Derivation and Structure in Categorial Grammar

Guy David Barry

PhD

University of Edinburgh

1992

EX BIBL. UNIV. EDINBURGH

Declaration

I declare that this thesis has been composed by myself and that the research reported therein has been conducted by myself unless otherwise indicated.

Edinburgh, 29 February 1992

Guy Barry

Acknowledgements

First of all I would like to thank my supervisor Robin Cooper for a great deal of support and encouragement during my time as a PhD student, and especially for being extraordinarily patient with me while I passed through an unnerving sequence of mental blocks towards the end of writing up. Thanks also to my examiners Mark Steedman and Ewan Klein for pointing out several areas which merited clarification.

My closest colleagues during the development of the thesis have been Glyn Morrill, Mark Hepple, Martin Pickering, Neil Leslie, Martin Emms and David Milward. Almost all of the present work has been greatly influenced by animated discussions with the above people in various combinations, and indeed some of it (as indicated in the text) has already been published as joint work. Special thanks to Martin Pickering and David Milward for commenting on the draft as it progressed.

Other sources of inspiration at various stages have included Steven Bird, Ted Briscoe, Matt Crocker, Elisabet Engdahl, Steve Finch, Norman Fraser, Herman Hendriks, Dick Hudson, Janne Johannessen, Michael Moortgat, Dick Oehrle, Paul Schweizer (my second supervisor), Anna Szabolcsi, Henry Thompson, Pete Whitelock and Mary Wood.

The thesis has perhaps benefited most of all from the exceptionally stimulating academic and social environment at the Edinburgh University Centre for Cognitive Science. The Centre has virtually been a home to me for the last three years, and I would like to thank all the students and staff there who have combined to make it such an enjoyable working-place.

Note

In accordance with regulation 3.5.8 of the Regulatory Standards for Theses I have included copies of the following three papers as an appendix:.

Morrill, G., Leslie, N., Hepple, M. and Barry, G. (1990). Categorial deductions and structural operations. In G. Barry and G. Morrill (eds), *Edinburgh Working Papers in Cognitive Science, Volume 5: Studies in Categorial Grammar*, Centre for Cognitive Science, University of Edinburgh, 1–21.

Barry, G. and Pickering, M. (1990). Dependency and constituency in categorial grammar. In G. Barry and G. Morrill (eds), *Edinburgh Working Papers in Cognitive Science, Volume 5: Studies in Categorial Grammar*, Centre for Cognitive Science, University of Edinburgh, 23–45.

Barry, G., Hepple, M., Leslie, N. and Morrill, G. (1991). Proof figures and structural operators for categorial grammar. In *Proceedings of the Fifth Conference of the European Chapter of the Association for Computational Linguistics*, 198–203.

The permission of the other authors has been obtained.

Abstract

Categorial grammar is an approach to language description that may be distinguished by the following two features: firstly classification of linguistic expressions is achieved by means of a system of recursively defined *types*, and secondly the grammar itself is lexicalist, in that all the information about the combining properties of words is in their lexical entries. This thesis considers ways in which categorial grammars can be adapted in order to make them more flexible and more linguistically expressive.

The thesis takes as its starting-point the established frameworks known as **AB** (developed by Ajdukiewicz and Bar-Hillel) and **L** (developed by Lambek). The linguistic coverage of each is discussed, and it is shown that **AB** generates structures analogous to those in a phrase-structure grammar, whereas **L** gives a 'flat' structure to expressions in which any string of words can receive a type. From this it is argued that neither framework as it stands assigns structures which are entirely suitable for linguistic description, **AB** being too restrictive and **L** being too general.

This leads into a discussion of the notion of constituency, in which it is argued (contrary to traditional analyses) that a variety of linguistic data supports a 'flexible' notion of constituency where it is permissible for constituents to overlap. In order to define such a notion of constituency rigorously, the framework of dependency grammar is introduced, and the concept of a *dependency constituent* is defined with respect to this framework. This leads to the definition of a new categorial framework **D**, within which only dependency constituents receive types. Some mathematical properties of **D** are discussed, along some of its deficiencies and some sample linguistic applications.

Attention is then turned to linguistic phenomena whose description is outside the scope of the frameworks so far discussed. These include constructions involving commutation, iteration and optionality. A hierarchy of logical systems called the 'categorial hierarchy' is used as motivation for a system of operators called *structural modalities*, which allows

such constructions to be described. Two systems of structural modalities are introduced, and their logical and linguistic applications are discussed.

To follow this, a more general network of categorial systems is introduced which contains both **D** and the categorial hierarchy, and the possibility of modalities to encode dependency is discussed. The approach in the thesis is then compared and contrasted with the alternative approach of Steedman's Combinatory Categorial Grammar, which makes use of a selected set of type-combining rules. The thesis concludes with a general overview of the work, and some pointers to further research.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

This thesis considers ways in which categorial grammars can be adapted in order to make them more linguistically expressive. It has two main aims, the first being to formulate a better motivated concept of 'constituent' than is generally considered, and the second being to show how the categorial formalism can be extended to accommodate linguistic phenomena outside the coverage of the original system. The thesis is not primarily concerned with providing a precise characterization for particular linguistic phenomena within categorial grammar; the emphasis is rather on exploiting the mathematical properties of the system so as to provide a richer framework for the theoretical linguist to work with.

Categorial grammar is an approach to language description that may be distinguished by the following two features. Firstly, classification of linguistic expressions is achieved by means of a system of recursively defined *types*, built from a set of ***primitive types*** and a number of *connectives*. The basic expressions are deemed to be phrasal rather than lexical (e.g. sentences and noun phrases); expressions in these categories receive primitive types. All other linguistic expressions (including words) receive complex types, based on their (syntactic and semantic) relation to expressions of primitive types.

Secondly, the grammar itself is lexicalist, in that all the information about the combining properties of words is in their lexical entries. The lexicon consists of a finite set of type-assignments to words, and the grammar takes the form of a general system of logical inference; there are no explicit grammar rules. Taken together, the lexicon and the infer-

ence procedure make it possible to assign the appropriate type or types to any grammatical expression of the language. Every current linguistic theory involves rich lexical entries, but theories vary on how much information is included in rules. Lexicalism has the advantage of putting all the possible sources of variation in the same place, and the rules can be defined as very general inference procedures with well-studied mathematical properties.

The framework of categorial grammar was chosen because of its distinctive approach to the classification of linguistic expressions, in two respects in particular. First, the system of syntactic classification is simultaneously a semantic classification, allowing a close correspondence between form and meaning to be defined. Second, because there are no separate grammar rules, the types to which expressions are assigned are completely determined by the combinatorial properties of those expressions, given a fixed meaning for the connectives. Hence categorial grammar is directly derived from logic and therefore gives rise to principled constraints on how it can be developed. This is in contrast to (say) a phrase-structure grammar, where the addition of a new grammar rule can make a difference to the set of expressions labelled by a particular category.

Categorial grammar also has the advantage over many linguistic systems of being able to deal with unbounded dependencies using the same machinery as is used for other constructions. Most other theories postulate a specialized mechanism whose main purpose is the analysis of unbounded dependencies (for example, *wh*-movement in GB or slash categories in GPSG). This is because there is no general analysis of unbounded dependencies using a phrase-structure grammar with conventional categories.

Some categorial grammars have a property which distinguishes them from the majority of linguistic theories: they allow more than one syntactically distinct analysis for a single semantic interpretation of a sentence. Such categorial grammars are sometimes known as *flexible*. Flexibility has been claimed to have many advantages, for examples in the analysis of coordination and in the modelling of human sentence processing. Categorial grammar is therefore an appropriate formal framework for the development of non-standard theories of constituency.

1.2 Overview

In chapter 2 the basic principles of categorial grammar are outlined, starting with the recursive definition of types and the categorial lexicon. The two simplest categorial frameworks, known as **AB** and **L**, are then introduced, using a notation known as *proof figures* for the presentation of derivations. The meaning of types in each of the two frameworks is then discussed, and their linguistic coverage is compared. This discussion is used as motivation for the rest of the material in the thesis.

In chapter 3 the notion of *constituency* is defined, and traditional 'rigid' constituency (as in a phrase-structure grammar) is contrasted with the less standard notion of 'flexible' constituency, where constituents are allowed to overlap. Linguistic arguments are put forward which suggest that a flexible notion of constituency is in fact to be favoured over the more usual rigid variety.

In order to develop a precise theory of flexible constituency, the framework of dependency grammar is introduced, in which syntactic relations hold between individual words rather than phrases. It is suggested that such relations can be motivated by considering examples for covariation between words, such as agreement and government. With this framework in place, the concept of a *dependency constituent* is defined as an expression whose words are connected by dependencies, and dependency constituency is shown to be a flexible form of constituency, satisfying the earlier requirements. A simplified formalization of dependency grammar is introduced which allows types to be given to dependency constituents.

A new categorial framework **D** is then presented, intermediate between **AB** and **L** in its coverage. The meaning of types in **D** is discussed, and it is shown that subject to certain assumptions the set of expressions derivable in **D** corresponds to the set of dependency constituents. Some sample linguistic applications of **D** are presented, along with a few of its deficiencies.

In chapter 4 attention is turned to those constructions which frameworks such as **AB**, **D** and **L** are inadequate to describe, in particular those involving commutation (i.e. non-canonical word orders), iteration and optionality. A solution to these problems is developed by way of the system of logics known as the 'categorial hierarchy', of which **L** is the bottom member. The higher logics in the hierarchy are formed from **L** by the addition of so-called *structural rules* manipulating sequences of assumptions. Structural rules are used to motivate new operators known as *structural modalities*, which permit the description of commutation, iteration and optionality. Two systems of structural modalities are introduced, and some sample linguistic applications are discussed.

In chapter 5 an attempt is made to combine the work of the previous two chapters, by presenting a more general network of categorial systems containing both **D** and the categorial hierarchy. The concept of modalities to encode dependency is also presented. The material in the thesis is then contrasted with the alternative categorial framework developed by Steedman, known as Combinatory Categorial Grammar, which makes use of a selected set of type-combining rules. The thesis concludes with a summary of the work, and some pointers to further research.

Chapter 2

Categorial Grammar

2.1 Principles of Categorial Grammar

In this section I shall first describe how types are defined in a categorial grammar and how the typing system may be used to construct a lexicon. Then I shall describe two standard frameworks of inference for categorial types and show how they may be used to generate grammatical expressions.

2.1.1 Types and the Lexicon

Syntactically, categorial grammar (CG) categorizes expressions according to how they combine with other expressions to form larger expressions. Thus in English a finite verb phrase (or intransitive verb) might be viewed as an expression that combines with a preceding noun phrase to form a sentence, and a transitive verb might be viewed as an expression that combines with a following noun phrase to form a finite verb phrase. This suggests a system of *directional* types as defined in (2.1):

(2.1) If X is a primitive type then X is a type.

If X and Y are types then X/Y and $Y\backslash X$ are types.

Here X/Y is the type of any linguistic expression that combines with a following adjacent expression of type Y to form an expression of type X, and $Y\backslash X$ is the type of any linguistic expression that combines with a preceding adjacent expression of type Y to form

an expression of type X.[1] The precise meaning of 'combines' has deliberately been left vague here, since it can be interpreted in different ways, giving rise to different varieties of categorial grammar. I shall go into more detail on this in the next section.

Semantically, CG adopts Frege's principle of *compositionality*; it assumes that the meanings of basic expressions are primitive, and that the meanings of other expressions are defined in terms of their relation to the meanings of the basic expressions. Thus, for instance, the meaning of a finite verb phrase might be viewed as a function from the meaning of its subject to the meaning of the sentence formed by combining it with its subject, and the meaning of a transitive verb might be viewed as a function from the meaning of its object to the meaning of the verb phrase formed by combining it with its object. This suggests a system of *functional* types as defined in (2.2):

(2.2) If X is a primitive type then X is a type.

If X and Y are types then $Y \rightarrow X$ is a type.

Here $Y \rightarrow X$ is the type of functions from meanings of type Y to meanings of type X. Note that throughout the thesis I shall only be concerned with those aspects of meaning that reflect function-argument structure; I shall not consider model-theoretic aspects of meaning such as the lexical semantics of individual words, or truth-conditions associated with expressions.

The Fregean principle allows us to state a direct correspondence between syntactic and semantic types in the manner of Montague (1970). Given a mapping τ_0 from primitive syntactic to semantic types such that if an expression has syntactic type X its semantic type is $\tau_0(X)$, τ_0 can be extended to a general mapping τ from all syntactic to semantic types in the following fashion:

(2.3) $\tau(X) = \tau_0(X)$ if X is a primitive type

---

[1]This is Lambek's (1958) notation for types. In the alternative notation used by Steedman (1987), what Lambek writes as $Y \backslash X$ would be written as $X \backslash Y$.

$$\tau(X/Y) = \tau(Y) \rightarrow \tau(X)$$

$$\tau(Y\backslash X) = \tau(Y) \rightarrow \tau(X)$$

Since an expression of syntactic type X/Y or $Y\backslash X$ can be combined with an expression of syntactic type Y to form an expression of syntactic type X, by the Fregean principle its meaning must be a function from meanings of semantic type $\tau(Y)$ to meanings of semantic type $\tau(X)$, and is thus of semantic type $\tau(Y) \rightarrow \tau(X)$. In the linguistic examples I shall use roman capital letters (e.g. S and NP) for names of primitive syntactic types, and primed versions of the same letters (e.g. S′ and NP′) for the corresponding semantic types, so that $\tau_0(\mathrm{S}) = \mathrm{S}'$, $\tau_0(\mathrm{NP}) = \mathrm{NP}'$, etc. (Italic capital letters (e.g. X and Y) will continue to be used for meta-variables over types.) This avoids argument over the precise nature of semantic types and the objects in those types.

For the purposes of illustration, assume that English has (at least) the primitive types given in (2.4), with their associated categories. Most of these categories are fairly standard in X-bar syntax (Jackendoff 1977), but note that the usual CG conventions assume N to be a phrasal rather than a lexical category (corresponding roughly to $\overline{\mathrm{N}}$ in X-bar syntax), and that finite verb phrases are not generally assumed to form a basic category. (The form of linguistic expressions will be represented by means of a sans-serif typeface, e.g. the expression "John walks" is represented as John walks.)

(2.4)

Syntactic type	Semantic type	Example members of category
S (sentences)	S′	**John walks, Mary walks, the man walks, John likes Mary, John likes the man, John speaks to Mary, John must walk, John says that Mary walks**
NP (noun phrases)	NP′	**John, Mary, the man**
N (common nouns)	N′	**man, woman**
PP (prepositional phrases)	PP′	**to Mary, for the man**
VP (non-finite verb phrases)	VP′	**walk, like Mary**
SP (complementized sentences)	SP′	**that John walks, that John likes Mary**

(This classification into types ignores many finer-grained distinctions of lexical categorization, such as number and case.)

Such a set of basic expressions suffices to infer lexical type-assignments. For instance, the word "walks" forms a sentence whenever it is preceded by a noun phrase, and so **walks** receives the syntactic type NP\S. Suppose we uniformly represent the meaning of a word by printing the same word in boldface (for example, the form of the word "walks" is represented as **walks**, and its meaning as **walks**). Then by the Fregean principle the semantic constant **walks** receives the semantic type NP′→S′. Similarly "likes" forms a finite verb phrase whenever it is followed by a noun phrase which in turn forms a sentence whenever it is preceded by a noun phrase; thus the syntactic object **likes** receives the syntactic type (NP\S)/NP, and the semantic object **likes** receives the semantic type NP′→(NP′→S′). On these principles we may construct a lexicon such as the following:

(2.5)

Word	Syntactic type	Meaning	Semantic type
walks	NP\S	**walks**	NP′→S′
runs	NP\S	**runs**	NP′→S′
likes	(NP\S)/NP	**likes**	NP′→(NP′→S′)
speaks	(NP\S)/PP	**speaks**	PP′→(NP′→S′)
must	(NP\S)/VP	**must**	VP′→(NP′→S′)
says	(NP\S)/SP	**says**	SP′→(NP′→S′)
John	NP	**John**	NP′
Mary	NP	**Mary**	NP′
the	NP/N	**the**	N′→NP′
man	N	**man**	N′
to	PP/NP	**to**	NP′→PP′
for	PP/NP	**for**	NP′→PP′
walk	VP	**walk**	VP′
like	VP/NP	**like**	NP′→VP′
that	SP/S	**that**	S′→SP′

Since semantic types are directly derivable from syntactic types, I shall occasionally associate meanings with syntactic types, which is not strictly correct (e.g. **walks** may be said to be of type NP\S rather than NP′→S′), but only where this does not lead to confusion.

Modifiers are usually treated as having *endocentric* functor types (i.e. functions from a type to itself). For instance, since **old man**, **old woman**, ... all have syntactic type N, **old** may be classified as having syntactic type N/N (and hence semantic type N′→N′); similarly, since **man from London**, **woman from London**, ... all have syntactic type N, **from London** may be classified as having syntactic type N\N (and semantic type N′→N′). On the same principles, VP-adverbials will have type (NP\S)/(NP\S) or (NP\S)\(NP\S) depending on whether they precede or follow their VPs. We shall see in the next chapter that this analysis of the types of modifiers is at odds with other aspects of the analysis to be adopted, but I shall keep to it for the time being.

The above lexicon makes certain tacit assumptions about syntactic structure, which will become clearer when we consider the interpretation of types in the next section. For the moment we shall assume that these types are fixed, and consider what inference procedures are needed to derive the types of other expressions.

2.1.2 The Categorial Frameworks AB and L

A categorial logic can be seen as consisting of a set of theorems or validities which determine how types can be combined. Coupled with this is an inference procedure which allows us to determine what the theorems of the logic are. This means that different formulations of the same logic are possible. Here I shall present two basic categorial logics, known as **AB** and **L**, using the *proof figure* notation developed in Barry, Hepple, Leslie and Morrill (1991), which is based on Prawitz' (1965) systems of 'natural deduction'.

Natural deduction was developed by Gentzen (1936) to reflect the natural process of mathematical reasoning in which one uses a number of *inference rules* to justify a single *conclusion* on the basis of a number of propositions, called *assumptions*. During a proof one may *temporarily* make a new assumption if one of the rules licenses the subsequent withdrawal of this assumption. The rule is said to *discharge* the assumption. The conclusion is said to *depend* on the undischarged assumptions, which are called the *hypotheses* of the proof.

A proof is usually represented as a tree with the assumptions as leaves and the conclusion at the root. Finding a proof is then seen as the task of filling this tree in, and the inference rules as operations on the partially completed tree. One can write the inference rules out as such operations, but as these are rather unwieldy it is more usual to present the rules in a more compact form as operations from a set of subproofs (the *premises*) to a conclusion, as follows (where $m \geq 1$ and $n \geq 0$):

(2.6)
$$\frac{\begin{array}{ccccccc} \vdots & & \vdots & [Y_1]^i & & [Y_n]^i \\ X_1 & \cdots & X_m & X_{m+1} & \cdots & X_{m+n} \end{array}}{Z} R^i$$

This states that a proof of Z can be obtained from proofs of $X_1, \ldots, X_{m+n}$ by discharging appropriate occurrences of assumptions $Y_1, \ldots, Y_n$ (if n is zero then no assumptions are discharged). The use of square brackets around an assumption indicates its discharge. R is the name of the rule, and the index i is included to disambiguate proofs, since there may be an uncertainty as to which rule has discharged which assumption.

As propositions are represented by formulas in logic, so linguistic categories are represented by type formulas in CG. The left-to-right order of types indicates the order in which the forms of subexpressions are to be concatenated to give a composite expression derived by the proof. Thus we must take note of the order and place of occurrence of the premises of the rules in the proof figures. Proofs are conventionally annotated with the words in left-to-right order along the top, so that each word is directly above its type.

The simplest framework of inference for directional categorial types is known as **AB** (after Ajdukiewicz (1935) and Bar-Hillel (1953)), or *classical categorial grammar.* **AB** can be formulated in the proof figure notation by means of just two inference rules, one for each of the connectives / and \. These are known as *elimination* rules since they determine how a type containing the connective may be used.

The elimination rule for / states that a proof of type X/Y followed by a proof of type Y yields a proof of type X. Similarly the elimination rule for \ states that a proof of type $Y\backslash X$ preceded by a proof of type Y yields a proof of type X. Using the notation above, we may write these rules as follows:

(2.7) a. $\dfrac{\begin{array}{cc}\vdots & \vdots \\ X/Y & Y\end{array}}{X}\,/\text{E}$ b. $\dfrac{\begin{array}{cc}\vdots & \vdots \\ Y & Y\backslash X\end{array}}{X}\,\backslash\text{E}$

By the Fregean principle, the semantic operation associated with each of the above rules is *functional application*; that is to say, in each case the meaning associated with the result type X is produced by applying the meaning associated with the *functor* type X/Y or $Y\backslash X$ to the meaning associated with the *argument* type Y. (Notation: meta-variables over meanings will be represented by lower-case Greek letters (e.g. ϕ and α). I adopt the usual notation of concatenation for function application, so that $\phi\alpha$ means ϕ applied to

α; however to aid clarity, where functions and/or arguments contain multiple characters, I shall add a space between the function and the argument, e.g. **runs Mary** means **runs** applied to **Mary**.) Thus an expression with meaning ϕ of type X/Y followed by an expression with meaning α of type Y gives an expression with meaning $\phi\alpha$ of type X, and an expression with meaning α of type Y followed by an expression with meaning ϕ of type $Y\backslash X$ gives an expression with meaning $\phi\alpha$ of type X. This correspondence makes it possible to derive the meaning representation for an expression directly from the structure of the proof.

For instance, the expression Mary likes John is derived as a sentence as follows:

(2.8)
$$\dfrac{\dfrac{\text{Mary}}{\text{NP}} \quad \dfrac{\dfrac{\text{likes}}{\text{(NP}\backslash\text{S)/NP}} \quad \dfrac{\text{John}}{\text{NP}}}{\text{NP}\backslash\text{S}}\,/\text{E}}{\text{S}}\,\backslash\text{E}$$

This gives rise to the meaning representation in (2.9):

(2.9) **(likes John) Mary**

Similarly, the sentence Mary says that John speaks to the man has the derivation in (2.10), and hence the meaning representation in (2.11):

(2.10)
$$\dfrac{\dfrac{\text{Mary}}{\text{NP}} \quad \dfrac{\dfrac{\text{says}}{\text{(NP}\backslash\text{S)/SP}} \quad \dfrac{\dfrac{\text{that}}{\text{SP/S}} \quad \dfrac{\dfrac{\text{John}}{\text{NP}} \quad \dfrac{\dfrac{\text{speaks}}{\text{(NP}\backslash\text{S)/PP}} \quad \dfrac{\dfrac{\text{to}}{\text{PP/NP}} \quad \dfrac{\dfrac{\text{the}}{\text{NP/N}} \quad \dfrac{\text{man}}{\text{N}}\,/\text{E}}{\text{NP}}\,/\text{E}}{\text{PP}}\,/\text{E}}{\text{NP}\backslash\text{S}}\,\backslash\text{E}}{\text{S}}\,/\text{E}}{\text{SP}}\,/\text{E}}{\text{NP}\backslash\text{S}}\,\backslash\text{E}}{\text{S}}\,\backslash\text{E}$$

(2.11) **(says (that ((speaks (to (the man))) John))) Mary**

The most general system of inference for directional categorial types is known as **L** (after Lambek (1958)), or *Lambek categorial grammar.*[2] The proof figure formulation contains an elimination rule for each connective, exactly as in **AB**, and so **L** subsumes the whole of **AB**. Thus the derivations in (2.8) and (2.10) are valid in **L**, yielding the same meaning representations. However, **L** also contains an *introduction* rule for each connective which states how a type containing that connective may be derived.

The introduction rule for / states that where the rightmost assumption in a proof of the type X is of type Y, that assumption may be discharged to give a proof of the type X/Y. Similarly, the introduction rule for \ states that where the leftmost assumption in a proof of the type X is of type Y, that assumption may be discharged to give a proof of the type $Y\backslash X$. Using the notation above, we may write these rules as follows:

(2.12) a. $\dfrac{\begin{array}{c}[Y]^i \\ \vdots \\ X\end{array}}{X/Y}/\text{I}^i$ b. $\dfrac{\begin{array}{c}[Y]^i \\ \vdots \\ X\end{array}}{Y\backslash X}\backslash\text{I}^i$

(The off-centre positioning of the Ys in the above rules is meant to indicate that in /I Y is the rightmost undischarged assumption in the proof of X, and in \I Y is the leftmost undischarged assumption in the proof of X.) In addition, **L** carries the condition that in both /I and \I the sole assumption in a proof cannot be withdrawn, so that no types are assigned to the empty string.

For both /I and \I, the meaning of the result is given by *functional abstraction* over the meaning of the discharged assumption, which is given by a variable of the appropriate type. I shall represent variables of a given type by an italic lower-case letter with the type name as a superscript, e.g. $x^{\text{NP}'}$ will represent a variable of type NP′, although I shall usually omit the superscript where the type is apparent from the context. I shall also adopt

---

[2]The term *Lambek calculus* is also widely used, but this should strictly refer only to Lambek's original sequent calculus formulation of the system.

the convention of using f, g, h to represent variables which act as functions, and x, y, z to represent variables which do not.[3]

Functional abstraction is represented in the usual way by means of the lambda operator. For example, the functional abstraction of (**likes** x) **Mary** over the variable x, written λx[(**likes** x) **Mary**], is the function that when applied to any term α gives (**likes** α) **Mary**. This general relation between abstraction, application and substitution is expressed by the law of *β-equality*:

(2.13) $\quad (\lambda y[\sigma])\alpha = \sigma[\alpha|y]$

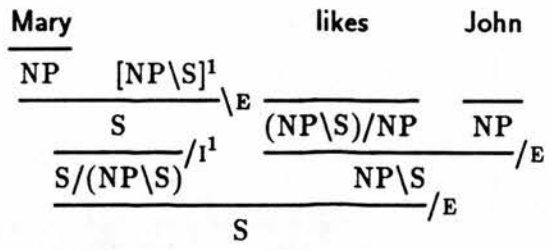
Here $\sigma[\alpha|y]$ means 'σ with α substituted for (each occurrence of) y'. The rules in (2.12) are analogous to the usual natural deduction rule of *conditionalization*, except that the latter allows withdrawal of any number of assumptions, in any position. The semantic algebra defined by the above set of inference rules corresponds to the subset of the typed lambda-calculus in which each binder binds exactly one variable (van Benthem 1983).

In **L**, unlike **AB**, it is possible to give more than one proof for a single interpretation of a string. For example, contrast the derivation of Mary likes John in (2.8), and the term in (2.9), with the derivations in (2.14) and the terms in (2.15):

(2.14) a.

$$\dfrac{\dfrac{\text{Mary}}{\text{NP}} \quad \dfrac{\dfrac{\dfrac{\text{likes}}{(\text{NP}\backslash\text{S})/\text{NP}} \quad [\text{NP}]^1}{\text{NP}\backslash\text{S}}/\text{E}}{\text{S}}\backslash\text{E}}{\dfrac{\dfrac{\text{S}}{\text{S/NP}}/\text{I}^1 \quad \dfrac{}{\text{NP}}}{\text{S}}/\text{E}}$$

[3]This convention is independent of the type of the variable, so that a variable of a functional type which acts as an argument to another function will be notated as x, y or z rather than f, g or h. The reason for this convention will become clear in chapter 3.

b. 

(2.15) a. $(\lambda x[(\textbf{likes}\ x)\ \textbf{Mary}])\ \textbf{John}$

b. $(\lambda f[f\ \textbf{Mary}])\ (\textbf{likes}\ \textbf{John})$

Note that the terms in (2.15) are both β-equal to the term in (2.9) (by the definition in (2.13)). Each of the proofs in (2.14) is thus said to be *derivationally equivalent* to the proof in (2.8). This property of derivational equivalence and has certain ramifications for theories of parsing (see Barry *et al.* 1991; cf. also Pareschi and Steedman 1987, Hepple and Morrill 1989). In the present context, it will be of most concern when we consider 'flexible constituency' in the next chapter.

2.1.3 Sequent Formulation of L

As an illustration of the fact that categorial frameworks can be given other equivalent formulations, I shall briefly present Lambek's original sequent formulation of **L**. **L** can also be formulated in other equivalent fashions, such as the *axiomatic* formulation of Zielonka (1981).

In sequent calculus (originally developed by Gentzen (1936)), the objects reasoned about are not individual types but *sequents*, corresponding to entire proofs in the natural deduction style. A sequent calculus consists of axioms and inference rules; axioms state that certain primitive sequents are valid, and inference rules allow new valid sequents to be derived from old ones. A sequent calculus proof may thus be seen as the instructions for the construction of a natural deduction proof. Let $\Gamma, \Delta, \Phi \ldots$ stand for sequences of types (and $X, Y, Z \ldots$ stand for single occurrences of a type as usual). A (single-conclusioned) sequent is a statement $\Gamma \Rightarrow X$; it asserts that there is a deduction of X depending on the

hypotheses Γ. In the sequent formulation of **L** both the order of the types in Γ and the number of occurrences of each type are relevant, and there is a restriction that Γ must be non-empty. A rule such as (2.16) states that Z follows from Γ if X follows from Δ and Y follows from Φ. It does not matter which order we solve the sub-problems in, so long as we solve them all.

(2.16) $$\begin{array}{ll} \Gamma \Rightarrow Z & \text{[Rule]} \\ \Delta \Rightarrow X & \\ \Phi \Rightarrow Y & \end{array}$$

In standard sequent systems there is a single schematic axiom saying that every type yields itself, and for each connective there are two inference rules; a *left rule*, which allows us to derive a sequent where the connective newly appears on the left-hand side of the arrow, and a *right rule*, which allows us to derive a sequent where the connective newly appears on the right-hand side of the arrow. The Gentzen-style formulation of **L** follows this pattern:

(2.17) $$X \Rightarrow X \quad \text{[Ax]}$$

$$\begin{array}{ll} \Gamma\ X/Y\ \Delta\ \Phi \Rightarrow Z & \text{[/L]} \\ \Delta \Rightarrow Y & \\ \Gamma\ X\ \Phi \Rightarrow Z & \end{array} \qquad \begin{array}{ll} \Gamma\ \Delta\ Y\backslash X\ \Phi \Rightarrow Z & \text{[\L]} \\ \Delta \Rightarrow Y & \\ \Gamma\ X\ \Phi \Rightarrow Z & \end{array}$$

$$\begin{array}{ll} \Gamma \Rightarrow X/Y & \text{[/R]} \\ \Gamma\ Y \Rightarrow X & \end{array} \qquad \begin{array}{ll} \Gamma \Rightarrow Y\backslash X & \text{[\R]} \\ Y\ \Gamma \Rightarrow X & \end{array}$$

There is thus a close analogy between the use of left rules in sequent calculus and elimination rules in natural deduction, and similarly between right rules in sequent calculus and introduction rules in natural deduction.

L may be formulated with one additional rule, the *Cut rule*, which combines two proofs by identifying the conclusion of one with one of the premises of the other:

(2.18) $$\frac{\Delta\ \Gamma\ \Phi \Rightarrow Y \quad [\text{Cut}] \qquad \Gamma \Rightarrow X}{\Delta\ X\ \Phi \Rightarrow Y}$$

Lambek shows that the sequent formulation of **L** has the property of *Cut-elimination*, i.e. every **L** proof can be transformed into an **L** proof with the same end-sequent, in which no Cuts occur. Cut-elimination in sequent calculus corresponds (approximately) to normalization in natural deduction (see Moortgat 1990a). The Cut-free formulation provides a decision procedure. A step in the search for a proof of a sequent can only be an axiom matching, or one of a finite number of possible rule applications, these creating strictly simpler subgoals since each rule application removes a connective.

2.2 Categorial Grammar and Linguistic Theory

In this section I shall consider what models of linguistic structure are implicitly entailed by frameworks such as **AB** and **L**, by looking more closely at the definition of syntactic types. This leads to some observations about the linguistic coverage of the frameworks and their inadequacies, which serve as motivation for the rest of the material in the thesis.

2.2.1 Interpretation of Types

The category symbols of (say) a phrase-structure grammar are by definition labels for linguistic units. Strictly speaking, the type symbols of categorial grammar have no such significance, since they merely reflect general combining properties of expressions. Nevertheless, there is a strong implicit assumption that the assignment of a type to an expression gives it some status as a linguistic unit (see for example Steedman 1987). Under this assumption, the set of expressions labelled by a type has a similar status to a linguistic category. I shall refer to this set as the *extension* of the type, and to the system for determining extensions of types as their *(syntactic) interpretation.*

Given some interpretation of types, we say that the sequence of types $X_1, \ldots, X_n$ *entails* the type Y iff, for any sequence of expressions $a_1, \ldots, a_n$ of types $X_1, \ldots, X_n$ respectively, the expression formed by combining $a_1, \ldots, a_n$ in that order is always of type Y. For an inference calculus to be appropriate to a given interpretation of types, it should therefore have the properties of *soundness* and *completeness* as defined in (2.19):

(2.19) a. A categorial inference calculus is *sound* with respect to a given interpretation of types iff, whenever a type Y can be derived from a sequence of types $X_1, \ldots, X_n$, it is the case that Y is entailed by $X_1, \ldots, X_n$.

b. A categorial inference calculus is *complete* with respect to a given interpretation of types iff, whenever a type Y is entailed by a sequence of types $X_1, \ldots, X_n$, it is the case that Y can be derived from $X_1, \ldots, X_n$.

Soundness ensures that no 'wrong' type-assignments are obtained by using the calculus; completeness ensures that no possible type-assignments are missed. If the calculus is sound and complete, and if it were somehow possible to ensure that all lexical type-assignments were consistent with the given interpretation, then the grammar would generate all and only grammatical expressions of the language. With these principles in mind, let us look at two possible interpretations for types. (Recall that in the discussion of (2.1) the operation of syntactic combination was deliberately left underspecified, in order to allow for different modes of interpretation for types.)

Suppose we assume that non-lexical expressions all have a hierarchical structure as generated by a binary-branching phrase-structure grammar, i.e. one in which each non-terminal node has exactly two daughters. In addition, assume (as in X-bar syntax) that one daughter in each local tree is distinguished as the 'head'. For example, consider the sentence **John spoke to Mary**. Here the standard assumption would be that the sentence divides into head daughter **spoke to Mary** and non-head daughter **John**; that **spoke to Mary** divides into head daughter **spoke** and non-head daughter **to Mary**; and that **to Mary** divides into head daughter **to** and non-head daughter **Mary**. This can be indicated by means of the bracketing in (2.20), where round brackets indicate head daughters and square brackets indicate non-head daughters:

(2.20) [[John] ((spoke) [(to) [Mary]])]

In such a system, then, there are two possible modes of combination for expressions: head followed by non-head, or non-head followed by head. Furthermore, combination of expressions is non-associative, i.e. structures cannot be rebracketed; *a* combined with (*b* combined with *c*) is not the same as (*a* combined with *b*) combined with *c*. This suggests the following interpretation of types:

(2.21) An expression is of type X/Y (resp. $Y\backslash X$) iff, whenever it is combined with a following (resp. preceding) non-head of type Y, the result is of type X.

So, for example, **spoke to Mary** has the type NP\S since it forms an S when combined with a preceding non-head of type NP; **spoke** has the type (NP\S)/PP since it forms an expression of type NP\S when combined with a following non-head of type PP. **John** has the type NP, but no other type, since it cannot be combined with a non-head to produce a grammatical expression. **Spoke to** has no type, since it does not form a unit in the phrase-structure tree.

As might be expected, **AB** is both sound and complete with respect to this interpretation of types. This is clearly reflected in the structure of **AB** proofs: /E inferences represent the combination of a head with a following non-head, and \E inferences represent the combination of a non-head with a following head.

Note that this definition implicitly identifies categorial functors (which are generally semantically motivated) with syntactic heads. This begs the question of what is the traditionally accepted notion of head, about which there is widespread disagreement. We shall not go into the issues here; see Zwicky (1985) and Hudson's (1987) reply to Zwicky for a comprehensive discussion. Hudson gives a great deal of evidence which suggests that, for constructions involving complementation, most syntactically motivated definitions of head in fact coincide with the notion of semantic functor. However, the reverse appears to be true in modifier constructions, where according to Hudson's criteria for headship (and nearly all linguistic theories) the modified element appears to be the head, whereas on semantic grounds the modifier is standardly assumed to be the functor.

There are several possible solutions to this problem. We could claim that modifiers are heads, but this leads to inconsistencies in syntactic description, as discussed in the next chapter. Alternatively, we could change the definition of head so that it did not always coincide with the semantic functor. This approach is adopted by Bouma (1988), following Vennemann and Harlow (1977), where the following definitions are proposed:

(2.22) a. Head: in a construction consisting of a functor F and an argument A, the head constituent is F, unless it is endotypic (i.e. F is of the form X/X or $X\backslash X$, in which case A is the head.

b. Complement: An argument which is not a head is a complement.

c. Modifier: A functor which is not a head is a modifier.

However, this approach would remove the direct correspondence between categorial grammar and X-bar syntax, and considerably complicate many of the arguments in this and the next chapter. The only alternative is to claim that modified elements are functors. This cannot be achieved in a general fashion within **AB**, since modified elements may combine with arbitrarily many modifiers, but will turn out to be possible when we consider extensions to the categorial machinery in later chapters. I shall eventually adopt this solution, but for the time being I shall ignore modifier constructions wherever possible.

Let us now change our hypothesis about syntactic structure and assume that expressions have a purely linear structure, i.e. every string is a linguistic unit. This has been termed *structural completeness* (Moortgat 1988). There is now only one notion of syntactic combination, namely concatenation. In addition, the operation is associative, so that bracketing is redundant; a concatenated with (b concatenated with c) is the same as (a concatenated with b) concatenated with c. This suggests a different interpretation of types:

(2.23) An expression is of type X/Y (resp. $Y\backslash X$) iff, whenever it is concatenated with a following (resp. preceding) expression of type Y, the result is of type X.

Clearly expressions receive all the types under this interpretation as they did under (2.21), but there are many other possibilities as well. For example, spoke again has the type

(NP\S)/PP, but it also has the type NP\(S/PP) since the expressions **John spoke, the man spoke**, ..., all have type S/PP. Similarly **John** has types other than NP, such as S/(NP\S), since whenever it is followed by an expression of type NP\S it forms an expression of type S. **Spoke to** can be given the type (NP\S)/NP, since whenever it is followed by an expression of type NP it forms an expression of type NP\S.

L is both sound and complete with respect to this interpretation; for the proof see Buszkowski (1983). Note particularly that **AB** is incomplete with respect to the interpretation in (2.23), and **L** is unsound with respect to the interpretation in (2.21). The former should be clear since (e.g.) **John** cannot be given the type S/(NP\S) in **AB**, and the latter conversely since **John** *can* be given the type S/(NP\S) in **L**.

2.2.2 Linguistic Coverage of AB and L

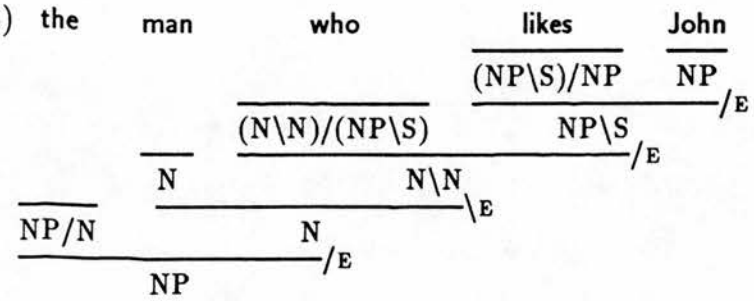
In assessing the linguistic coverage of categorial frameworks, there are two distinct questions to be answered, corresponding roughly to the notions of weak and strong generative capacity (Chomsky 1965). The first (linguistically less interesting) question is what set of strings can be generated by the grammar; the second is what structures it is capable of assigning to them.

To answer the first question, **AB** has been proved to have context-free generative power (Bar-Hillel, Gaifman and Shamir 1960), and it is conjectured that **L** has also. Hence we may presume that they are capable of generating the same set of strings. However, as we saw in the last section, **AB** is capable of assigning many fewer structures than **L**. In practice, this makes a significant difference to the linguistic coverage of the two frameworks, as I shall now illustrate.

Consider English relative clause constructions using **who** and **whom**. Both **AB** and **L** are capable of generating simple (non-embedded) subject relative constructions in a reasonably straightforward fashion. To see this, note first that restrictive relative clauses, being postmodifiers of nouns, can receive the type N\N. This makes it possible to assign a type to the subject relative pronoun **who** given a lexicon such as in (2.5), since it combines with verb phrases, of type NP\S, to yield relative clauses. We may therefore give **who** the

type (N\N)/(NP\S). This gives rise to derivations such as the following:[4]

(2.24)



(2.25) **the ((who (likes John)) man)**

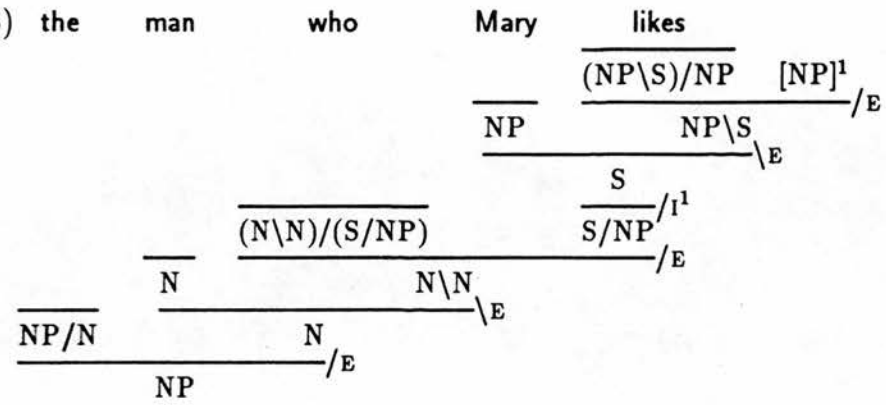
However, it is not possible given the type-assignments in (2.5) to derive object relative clauses in a simple fashion in **AB**, whatever the type of the relative pronoun. For example, in the man who Mary likes, Mary likes cannot be given a single type, since the types NP and (NP\S)/NP cannot be combined. Thus it is necessary either to give who a type that combines with Mary and likes separately, or to give either Mary or likes an additional lexical type. This suggests that it would be desirable for the grammar to generate Mary likes as a unit.

In contrast, observe that the introduction rules of **L** give us an apparatus for this and similar constructions which is lacking in **AB**. Since we can derive Mary likes as an expression of type S/NP (as in (2.14a)), assignment of the type (N\N)/(S/NP) to the object relative pronoun **who(m)** yields the derivation in (2.26) and the term in (2.27) (cf. Ades and Steedman 1982):

---

[4]Note once more that by representing the meaning of who as **who** we are not concerning ourselves with the truth conditions associated with relative clauses, but merely with their function-argument structure.

(2.26)



(2.27) **the (who (λx[(likes x) Mary]) man)**

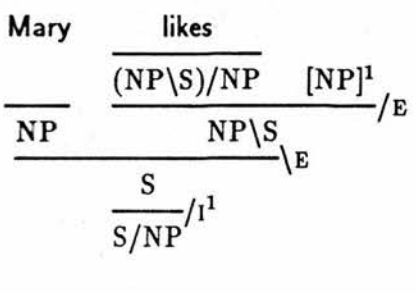
In fact, with no further apparatus we now have a simplified account of some unbounded dependencies, which is impossible in **AB** without an unprincipled set of extra lexical type assignments. For example, all the expressions in (2.28) may be given the type S/NP, as demonstrated in (2.29); the corresponding lambda-terms are given in (2.30):

(2.28) a. Mary likes

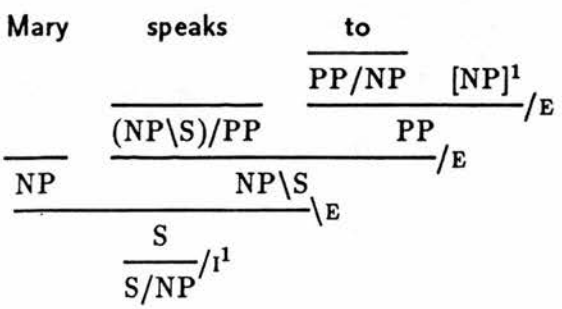
b. Mary speaks to

c. Mary says that John likes

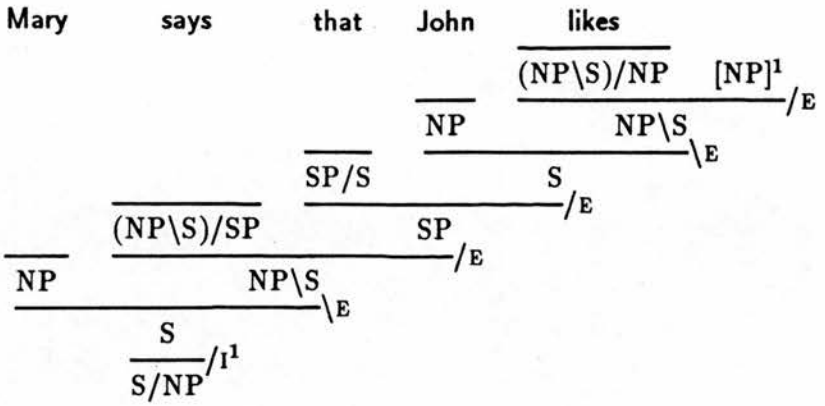
(2.29) a.



b.



c.



(2.30) a. λx[(**likes** x) **Mary**]

b. λx[(**speaks (to** x)) **Mary**]

c. λx[(**says (that** ((**likes** x) **John**))) **Mary**]

Thus, given the lexical type (N\N)/(S/NP) for the object relative pronoun **who(m)**, we may derive all the relative clause constructions in (2.31) in a similar fashion to (2.26):

(2.31) a. the man who Mary likes

b. the man who Mary speaks to

c. the man who Mary says that John likes

The reason why this works is that, as noted earlier, **L** can generate *any* string as a unit, and hence the proposed account will always allow relativization when the extraction site is *peripheral* (i.e. at leftmost or rightmost position in its clause), since the body of the relative clause forms a continuous string in the canonical sentence.[5]

This raises two questions which form the main topics of this thesis. First, is it necessary or desirable to use a framework which generates all strings in order to deal with these and

---

[5]This type for the relative pronoun therefore leads to island constraint violations such as *the man who Mary saw a woman who likes; see 5.2.3 for a brief discussion of this.

other constructions? Second, is the restriction to concatenative operations sufficient to capture all such constructions? In chapter 3 I shall argue that the answer to the first question is no, and instead propose a new framework **D** intermediate between **AB** and **L** which appears to capture the notion of a linguistic unit in a more satisfactory fashion. Then in chapter 4 I shall argue that the answer to the second question is also no, and propose the addition of extra operators to the previously proposed categorial frameworks to deal with this. In chapter 5 I shall use the material in chapters 3 and 4 to motivate a more general system of categorial logics, make some comparison with other categorial frameworks, and discuss directions in which further research might be pursued.

Chapter 3

Flexible Constituency and Dependency

3.1 Rigid and Flexible Constituency

So far I have been using the term 'linguistic unit' rather loosely. In this section I shall attempt to make this more precise by discussing the notion of *constituency* in syntax. In particular, I shall argue that the standard 'rigid' notion of constituency as assumed in phrase-structure grammar is inadequate for linguistic purposes, and that a more general 'flexible' notion appears to be required.

3.1.1 Definitions of Constituency

In order to define constituency in a suitably general fashion, we first need the notion of *subexpression*. (This definition assumes that an expression always consists of a string of words.)

(3.1) Given two expressions $\mathcal{E}$ and $\mathcal{F}$, $\mathcal{E}$ is a ***subexpression*** of $\mathcal{F}$ if $\mathcal{E}$ consists of a (non-empty) subset of the words in $\mathcal{F}$ in the same linear order. If the words in $\mathcal{E}$ form a continuous substring of $\mathcal{F}$, $\mathcal{E}$ is called a *continuous subexpression* of $\mathcal{F}$; otherwise it is called a *discontinuous subexpression* of $\mathcal{F}$.

So, for example, if A, B, C, D, E are (distinct) words in the given language, then the expression $ABCDE$ contains (among others) the continuous subexpressions B, CD and $ABCDE$, and the discontinuous subexpressions AC, BDE and ACE (but not, for example, CB or $BCCD$).

We now define constituency as follows:

(3.2) A *constituent representation* for an expression $\mathcal{E}$ is a set of subexpressions of $\mathcal{E}$, called the *constituents* of $\mathcal{E}$. Constituents may be referred to as *continuous* or *discontinuous* as defined for subexpressions in general.

(3.3) A theory of *constituency* is any scheme that assigns a constituent representation to each grammatical expression of a language.

I shall discuss the linguistic motivation for assigning constituent status to a subexpression below. For the moment, let us assume that a theory of constituency is free to assign constituent status to any subexpression whatsoever.

A notion of constituency is of course central to all theories of grammar based on phrase-structure. However, the version of constituency usually assumed in context-free phrase-structure grammar is considerably more restrictive than that given in (3.3), in at least two respects. Firstly, in context-free grammar all constituents are continuous, whereas the above definition also permits discontinuous constituents. Secondly, context-free grammar always assumes that no two constituents in an expression may *overlap*. Here two expressions are defined to overlap if their intersection is non-empty and one is not a subexpression of the other. For example, in the three-word expression ABC, the subexpressions AB and BC overlap, but ABC and BC do not, nor do AB and C. Because of the nature of rewrite rules, a context-free phrase-structure grammar would not permit both AB and BC to be constituents, but since (3.3) allows any subexpressions whatsoever to be constituents, overlapping constituents are of course permitted. I shall therefore make a distinction between *rigid* theories of constituency, which do not allow overlapping constituents, and *flexible* ones, which do.

For example, consider the sentence John loves Mary. In a traditional phrase-structure grammar assuming rigid constituency, this might receive the constituent representation in (3.4):

(3.4) {John loves Mary, John, loves Mary, loves, Mary}

No two of these constituents overlap. However, a rigid theory of constituency would not allow the representation in (3.5), since **John loves** and **loves Mary** overlap:

(3.5) {**John loves Mary, John, John loves, loves Mary, loves, Mary**}

In contrast, a flexible theory of constituency might allow either (3.4) or (3.5), and indeed many other representations.

In both (3.4) and (3.5), all individual words are constituents, but of course this is not a necessary consequence of constituency either. I shall refer to theories of constituency that impose this requirement as *exhaustive*. For example, the constituent representation in (3.6) is compatible with a theory of constituency that is rigid, but not exhaustive:

(3.6) {**John loves Mary, John, Mary**}

It should be noted that the existence of overlapping constituents in an expression is not strictly speaking incompatible with rigid constituency, since it could be that the expression is structurally ambiguous (i.e. has several different representations in a rigid system). For instance, suppose the expression ABC contains the set of constituents in (3.7). This set constitutes a single representation in a flexible system, but the same set could also arise within a rigid system from the two representations in (3.8):

(3.7) $\{ABC, AB, BC, A, B, C\}$

(3.8) a. $\{ABC, AB, A, B, C\}$

b. $\{ABC, BC, A, B, C\}$

However, the representation in (3.8) is considerably less parsimonious than that in (3.7), since in (3.8) the constituenthood of ABC, A, B and C is represented twice. On purely theoretical grounds, therefore, I shall assume (in the absence of evidence to the contrary) that the existence of overlapping constituents in an expression is a result of flexible constituency rather than structural ambiguity.

The above definitions have been purely mathematical and have not considered any linguistic restrictions on what may act as a constituent. We turn to this next.

3.1.2 Linguistic Arguments for Flexible Constituency

In phrase-structure grammar, the generally assumed linguistic motivation behind assigning constituent status to a subexpression is *distributional*, i.e. governed by standard syntactic tests such as the ability to be replaced by a proform, to be fronted or to be coordinated. For example, consider the sentence **John saw the woman in the red dress.** Here there is distributional evidence that **the woman in the red dress** is a constituent. It can be replaced by a proform as in (3.9), it can be fronted as in (3.10), and it can be coordinated as in (3.11):

(3.9) John saw her.

(3.10) The woman in the red dress, John saw.

(3.11) John saw the woman in the red dress and the man in the grey suit.

I shall now argue that distributional tests such as these are at least as good a motivation for flexible constituency as for rigid constituency. I shall consider a number of possible constituency tests, many of which have been employed standardly within rigid constituency theory in order to distinguish between competing rigid analyses. The survey is not exhaustive, but is intended to show in varied ways that rigid constituency is empirically too weak. It is important to realize that a linguistic theory may reject a particular test, but that the systematic use of at least some of these tests is necessary for any theory based on distributional evidence at all.

Consider coordination first. A conjunct can consist of either subject and verb, or verb and object:

(3.12) a. John loves and Bill hates Mary.

b. John loves Mary and hates Sue.

This is obviously problematic data for any theory assuming rigid constituency (defined distributionally) which incorporates coordination into the grammar, since we can infer

that **John loves** and **loves Mary** are both constituents in **John loves Mary** (and not merely in the coordinated sentences, which would be far less interesting). Coordination provides perhaps the clearest case for assuming flexible constituency in general, and the number of strings that can serve as conjuncts can be very great.

Fronting is another standard test. For example, pairs like (3.13) are standardly used to support the claim that **Mary** is a constituent:

(3.13) a. **John loves Mary.**

b. **Mary, John loves.**

The usual argument is roughly that **Mary** appears in a different position in (3.13b) from its position in the canonical sentence (3.13a), and hence must be a distributional unit. However, exactly the same data and the same argument could also be used to support the claim that **John loves** is a constituent, since neither of the two parts of (3.13b) has any distinguished status on purely distributional grounds. By the same argument, then, (3.14) shows that **I think John loves** and **I think loves Mary** are both constituents in **I think John loves Mary**, which again requires flexibility:

(3.14) a. **Mary, I think John loves.**

b. **John, I think loves Mary.**

(Note that in this case one of the proposed constituents is discontinuous.)[1]

Other less standard tests for constituency also support a flexible analysis. For example, we might reasonably claim that the material missing from a 'gapped' conjunct must be

---

[1]In fact in languages with complex fronting, such as German, the fronting test alone would appear to support flexible constituency. Consider the following (taken from Uszkoreit 1987):

(i) a. **Das Buch schenken wollte Peter dem Jungen**
the book-ACC give wanted Peter the boy-DAT
'Peter wanted to give the boy the book'

a constituent, such as **eats chocolate** in (3.15a). In (3.15b), the missing material is **eats constantly** (which is discontinuous in the original); again the two expressions overlap.

(3.15) a. **John eats chocolate constantly and Bill, occasionally.**

b. **John eats chocolate constantly and Bill, strawberries.**

A final test worth mentioning is replacement. It is sometimes proposed that if a set of words can be replaced by a constituent then those words are also a constituent (see e.g. Radford 1988, pp. 90–93). Of course this presupposes that we know that some sets are constituents already; in particular, if we assume that all single words are constituents, the claim follows that any string that can be replaced by a single word must be a constituent. In fact this is a flawed test as it stands, since for example in **I saw Mary yesterday** the substring **Mary yesterday** can be replaced by **Mary**, and **Mary yesterday** is not a constituent under normal analyses. It might be possible to save this test by going beyond strictly syntactic criteria and enforcing some sort of semantic substitutability, but even if this were possible the test would still support flexible constituency. For example, in **I will see Mary**, **will see** can be replaced by **saw**, and **see Mary** can be replaced by **arrive**, so we can infer that **will see** and **see Mary** are both constituents.

If these tests for constituent structure are to be used at all, then, the data strongly suggest that a flexible model of constituent structure is to be preferred. Indeed, in certain restricted cases, flexible constituency (or its equivalent, structural ambiguity) is standardly assumed within some phrase-structure-based theories, under what is termed *reanalysis*. Consider (3.16):

(3.16) a. **At Mary, John stared.**

---

b. **Dem Jungen schenken wollte Peter das Buch**
the boy-DAT give wanted Peter the book-ACC
'Peter wanted to give the boy the book'

In these examples the fronted elements **das Buch schenken** and **dem Jungen schenken** clearly overlap.

b. **Mary was stared at.**

(3.16a) suggests that **at Mary** is a prepositional phrase, but (3.16b) suggests that **stared at** is a phrasal verb (for the reasons for these claims, see e.g. Radford 1988). One standard conclusion is that **stare at Mary** has the structure [**stare** [**at Mary**]], but that it can be reanalysed as [[**stare at**] **Mary**]. This is simply an assumption of restricted flexibility; from this, it is not clear why flexibility should not be made into a general principle. Whatever the status of this analysis, we can conclude that the case for flexible constituency is very strong.

However, I have not so far argued for any particular flexible analysis. One (extreme) possibility is that all strings are constituents, i.e. structural completeness (see 2.2.1). There are a number of reasons to argue that this is not a good assumption, since some strings do not appear to serve as distributional units under any test. For instance, consider the subexpression **Mary Bill** in (3.17):

(3.17) **John told Mary Bill liked Sue.**

The tests considered above do not generally support the constituent status of this subexpression. For example, (3.18a–b) show that it cannot be replaced by a proform or fronted. The coordination test is more liberal, since it appears that a larger class of expressions can be coordinated (see 3.3.4), but (3.18c) is at least questionable:

(3.18) a. ***John told ?? liked Sue.**

b. ***Mary Bill, John told liked Sue.**

c. **?John told [Mary Bill] and [Jane Fred] liked Sue.**

In conclusion, it appears that any reasonable theory of flexible constituency will have some restriction on the degree of flexibility; in general (though not necessarily for every sentence), there will be some subexpressions which should not be treated as constituents. In the next section I shall describe a theory of flexible constituency, based on dependency grammar, which meets this requirement.

3.2 Dependency Grammar and Dependency Constituents

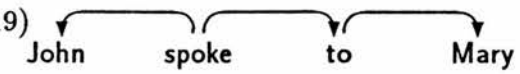
In this section I shall make a brief excursion into dependency grammar, outlining the general framework and giving some linguistic motivation. I shall then show that it is straightforward to derive a linguistically plausible notion of flexible constituency from the structures in a dependency grammar; this notion of constituency, referred to as *dependency constituency*, will play an important role in the rest of this chapter. I also proposed a simplified formal framework to generate dependency constituents, in preparation for the categorial model to be presented in the next section.

3.2.1 Dependency Structures

Dependency grammar (DG) is a grammatical framework in which the primitive syntactic links hold between individual words (rather than phrases). These links, called *dependencies*, relate two words called the *head* and the *dependent.* If a dependency links a head H and a dependent D, we say that D is *a dependent of* H (or *depends on* H), and that H is *a head of* D. A *dependency structure* for an expression consists of all the dependencies between the words in the expression. Thus dependency structures have the form of *directed graphs* where the words are viewed as nodes and the dependencies as directed edges, leading from the head to the dependent.

A *dependency diagram* for an expression is a diagram giving both the linear form of the expression and its dependency structure. For example, in the sentence John spoke to Mary, suppose that John and to both depend on spoke, that Mary depends on to, and that there are no other dependencies. (I shall discuss the linguistic motivation for these dependencies in 3.2.2 below.) Then the dependency diagram will be as in (3.19):

(3.19)



The notation is straightforward: we simply write the words in their usual linear order, and draw a directed edge between each pair of words in a dependency, with an arrow from the head to the dependent. Note if the directions on the edges in (3.19) are ignored then it is always possible to find a continuous chain of edges from any node to any other node. This is referred to in graph-theoretic terms as *connectedness.*

In many DGs, dependency structures are restricted to have the form of *directed trees.* A *directed tree* (or *rooted tree*) is a directed graph with the following properties:

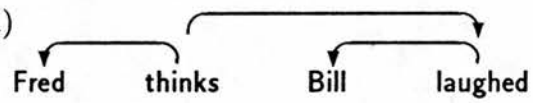
(3.20) a. The graph is connected.

b. There is a single node, called the *root*, which has no head.

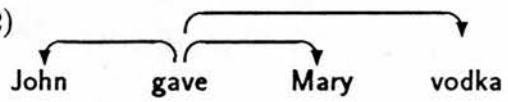
c. All nodes except the root have exactly one head (the *single-head requirement*).

For example, the structure in (3.19) is a directed tree with root **spoke.** Similarly the structures in (3.21) and (3.22) are directed trees with roots **thinks** and **gave** respectively:

(3.21)

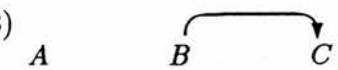


(3.22)

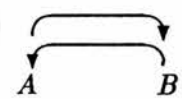


However, none of the three structures below is a directed tree; (3.23) because it is not connected, (3.24) because there is no node without a head, and (3.25) because B has two heads:

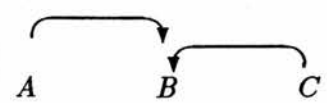
(3.23)



(3.24)



(3.25)



It is also common to make an assumption about the relation between dependency structure and linear order of words, which is usually called the *adjacency requirement.* To define this we need the notion of *subordinacy*, which is the transitive closure of dependency, i.e.:

(3.26) Given two words A, B in a dependency structure, B is *subordinate* to A if it is a dependent either of A or of some other word subordinate to A.

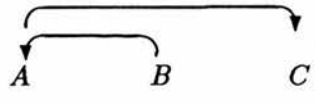
For example, in (3.21) **Fred**, **laughed** and **Bill** are all subordinate to **thinks**. The adjacency requirement runs as follows:

(3.27) No word may intervene between two words in a dependency relation unless it is subordinate to one of them.

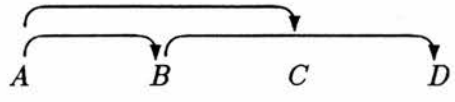
All the above dependency structures satisfy this requirement. (3.19) satisfies it trivially since all dependencies are between adjacent words. (3.21) satisfies the requirement because **Bill**, which intervenes between **thinks** and **laughed**, is subordinate to **laughed** (as well as to **thinks**). (3.22) also satisfies the requirement because **Mary**, which intervenes between **gave** and **vodka**, is subordinate to **gave**.

On the other hand, the structures in (3.28) and (3.29) violate the adjacency requirement:

(3.28)



(3.29)



From now on I shall assume the adjacency requirement, although it will not play a central role in this section since I shall mostly be ignoring word order.

The assumption that dependency structures are trees, plus the adjacency requirement, yields the version of DG which is sometimes known as *classical dependency grammar* (Gaifman 1965, Robinson 1970; see Fraser 1989). Classical DG has been shown to be (weakly)

equivalent to context-free phrase-structure grammar (Gaifman 1965). Of course this means that classical DG has similar generative inadequacies to context-free phrase-structure grammar, or indeed **AB**; for instance, it is impossible to formulate a principled account of unbounded dependencies. (The only way that this has been achieved within a DG framework is by violating the single-head requirement (see Hudson 1984).) However, I shall continue using classical DG for the time being because of its straightforward formal properties. (A brief proposal for refining the framework will be given in 3.3.2.)

It is always possible to derive a classical dependency structure from a headed binary-branching phrase-structure tree (as defined in 2.2.1). To do this we need some more terminology. Define *phrasal* heads and dependents as follows:

(3.30) Let $\mathcal{E}$ be an expression with two daughters $\mathcal{E}_1$ and $\mathcal{E}_2$, of which $\mathcal{E}_1$ is the head daughter. We shall say that $\mathcal{E}_1$ is the *phrasal head* of $\mathcal{E}$, and $\mathcal{E}_2$ is the *phrasal dependent* of $\mathcal{E}_1$.

So, for example, consider the sentence John spoke to Mary with the structure in (3.31) (repeated from (2.20)). As there, round brackets indicate head daughters and square brackets indicate non-head daughters:

(3.31) [[John] ((spoke) [(to) [Mary]])]

By the above definition, the phrasal head of the sentence John spoke to Mary is spoke to Mary, which has phrasal dependent John. In turn, spoke to Mary has phrasal head spoke, which has phrasal dependent to Mary. Similarly, to Mary has phrasal head to, which has phrasal dependent Mary.

Given the above definition of phrasal head, we may now define *lexical* heads and dependents in such a way that they induce dependency structures compatible with classical DG. The definitions run as follows:

(3.32) The *lexical head* of an expression $\mathcal{S}$ is $\mathcal{S}$ if $\mathcal{S}$ is lexical; otherwise it is the lexical head of the phrasal head of $\mathcal{S}$.

(3.33) Let $\mathcal{S}_1$ be a phrasal head with phrasal dependent $\mathcal{S}_2$; let $\mathcal{W}_1$ be the lexical head of $\mathcal{S}_1$, and $\mathcal{W}_2$ be the lexical head of $\mathcal{S}_2$. Then $\mathcal{W}_2$ is a *lexical dependent* of $\mathcal{W}_1$.

By (3.32), in the sentence **John spoke to Mary** it follows that **John** is the lexical head of **John** (and of nothing else), and that **Mary** is the lexical head of **Mary**; **to** is the lexical head of **to**, and hence also of **to Mary**; similarly **spoke** is the lexical head of **spoke**, hence also of **spoke to Mary**, hence also of **John spoke to Mary**.

Then, by (3.33), since **spoke to Mary** has the phrasal dependent **John**, **spoke** has the lexical dependent **John**. Similarly, since **spoke** has the phrasal dependent **to Mary**, **spoke** also has the lexical dependent **to**. Finally, since **to** has the phrasal dependent **Mary**, **to** has the lexical dependent **Mary**. Thus we have generated the same dependency structure as in (3.19).

So far I have presented the account purely formally. I shall now discuss the linguistic motivation both for postulating individual dependency relations and for making the above assumptions about dependency structures.

3.2.2 Linguistic Motivation for Dependency Structures

The linguistic motivation for assuming a dependency between two words within classical DG is essentially that the head specifies the category of its dependent, that is to say *subcategorizes* for it. For example, in **John spoke to Mary**, **spoke** may be said to determine the category of **John** and **to**, while **to** determines the category of **Mary**. Note that in other dependency-based frameworks such as Word Grammar (Hudson 1984) additional dependencies are allowed. For example, Word Grammar postulates dependencies between words in a control relation (e.g. **John** and **run** in **John wants to run**) or in a predication relation (e.g. **John** and **happy** in **John is happy**). However, such dependencies can be seen as semantic rather than syntactic, and therefore not appropriate to the present account. Hence I shall assume from now on that subcategorization is the sole criterion for dependency.

This assumption already gives us good grounds for assuming that that dependency structures are directed trees. Connectedness is straightforwardly justifiable, since otherwise

there would be no reason to regard the expression as a syntactic unit. The root, being the unique element with no head, does not have its category determined by any other word, and so its category is free to determine the category of the whole expression (e.g. the root of a sentence will always be a verb). Each other word has exactly one head, since its category must be determined by some word, but cannot be determined by more than one word.

In general, there is no completely reliable means of determining dependency relations directly from the form of sentences, and subcategorization must be regarded as a strictly theoretical construct. However, there are several features of syntax which suggest strongly that some form of connection between individual words is well motivated. These are traditionally classed under such headings as 'agreement' and 'government'. (Here the term 'agreement' refers purely to 'syntactic' rather than 'semantic' agreement, i.e. agreement constrained solely by syntactic features; for a discussion see Corbett 1979.) A characterization of agreement and government in a categorial framework is given by Bach (1983) (see also Keenan 1974); here, however, I am concerned purely with a theory-neutral characterization of the concepts.

Palmer (1971) offers a distinction between two kinds of linkage between words: (i) a word or class of word requiring a particular form of another word, and (ii) a form of one word requiring a corresponding form of another. He proposes the labels 'government' and 'agreement' to describe (i) and (ii) respectively, even though this use does not always correspond to the traditional use of the terms. For example, he cites the Latin sentence given below (note that the verb **pareo** ('obey') takes its object in the dative):

(3.34) **Viro bono paruit.**
Man-SING-DAT good-MASC-SING-DAT obey-PERF-3RD-SING
'He obeyed the good man'

Under Palmer's definition we would say that:

1. The verb (**paruit**) governs the noun (**viro**) in case (dative).

2. The adjective (**bono**) agrees with the noun in case.

3. The adjective also agrees with the noun in number (singular).

4. The noun governs the adjective in gender (masculine).

In the first three of these the uses of the terms 'agreement' and 'government' coincide with their usual uses, but the last would traditionally be referred to as agreement, even though it is a property of the noun **vir** itself, and not the particular form of the noun, that gives rise to the masculine inflection on **bono**.

Palmer's definition turns out to be a convenient one from the point of view of dependency, and so I shall adopt it here. In fact both agreement and government can be seen as special cases of a more general phenomenon called *covariation*. A formal definition of all three is given in (3.35).

(3.35) a. Suppose $\mathcal{E}$ is a grammatical expression containing two words X and Y. If there exist other words X_1 and Y_1, of the same categories as X and Y respectively, such that $\mathcal{E}$ is ungrammatical when either X_1 is substituted for X alone or Y_1 is substituted for Y alone, but grammatical when both X_1 is substituted for X and Y_1 is substituted for Y alone, then we shall say that X and Y *covary*.

b. If in (a) X and X_1 are forms of the same word, and Y and Y_1 are forms of the same word, then we shall say that X and Y *agree*.

c. If in (a) Y and Y_1 are forms of the same word, but X and X_1 are not forms of the same word, then we shall say that X *governs* Y.

If X and Y covary, and more specifically if they agree, we may take this as evidence of a dependency between X and Y, but since the relation is symmetric it is not possible from covariation or agreement alone to determine the direction of the dependency, and other factors must be brought into account. If X governs Y we may take this as evidence that Y depends on X. However, these conditions should not be thought of as either necessary or sufficient; I shall discuss exceptions towards the end of this subsection.

So, for example, in (3.36a) **relies** and **on** covary; in (3.37a) **he** and **likes** covary, and in addition they agree; in (3.38a) **will** and **see** covary, and in addition **will** governs **see.**

(3.36) a. John relies on Mary.

b. John trusts in Mary.

c. *John trusts on Mary.

d. *John relies in Mary.

(3.37) a. He likes Mary.

b. They like Mary.

c. *They likes Mary.

d. *He like Mary.

(3.38) a. John will see Mary.

b. John has seen Mary.

c. *John has see Mary.

d. *John will seen Mary.

In (3.36), therefore, we may postulate a dependency between the verb and the preposition, and in (3.37) a dependency between the subject and the verb, although in neither case is the direction determined. (3.38) gives evidence that the non-finite verb depends on the auxiliary.[2]

Note that where morphological marking is not explicit, tests such as the above will not always work. For example, contrast (3.37) and (3.38) with (3.39) and (3.40):

(3.39) a. He liked Mary.

---

[2]Some authors (e.g. Matthews 1981, pp. 155–156) have proposed the opposite direction of dependency for auxiliary constructions, but this position is less consistent with the above criteria.

b. **They liked Mary.**

(3.40) a. **John will hit Mary.**

b. **John has hit Mary.**

In (3.39) agreement cannot be directly demonstrated because of the lack of number marking on English past tenses. Similarly, in (3.40), because of the coincidence of the base and perfect forms of **hit**, government cannot be directly demonstrated. In such cases appeal must be made to analogous examples such as (3.37) and (3.38).

Sometimes, indeed, it is not possible to demonstrate certain examples of agreement and government in a given language at all. For instance, since all transitive verbs in English take accusative objects, it is impossible to demonstrate covariation between a transitive verb and its object, unless one considers 'cognate objects' such as in the following example:

(3.41) a. **John sneezed a loud sneeze.**

b. **John coughed a loud cough.**

c. ***John coughed a loud sneeze.**

d. ***John sneezed a loud cough.**

On the other hand, it is possible to do so in German, where most verbs, such as **sehen** ('see'), take an accusative object, but some, such as **helfen** ('help'), take a dative object:

(3.42) a. **Ich sah ihn.** (I saw him-ACC)

b. **Ich half ihm.** (I helped him-DAT)

c. ***Ich half ihn.**

d. ***Ich sah ihm.**

From such examples we see that verbs govern their objects for case in German, and may reasonably conclude that objects depend on verbs in general.

A less standard example:

(3.43) a. **Every boy laughed.**

b. **Many boys laughed.**

c. ***Many boy laughed.**

d. ***Every boys laughed.**

This example appears to demonstrate that determiners govern nouns in English, suggesting that nouns depend on determiners. This position is not universal within dependency grammar (it appears commoner to assume to the opposite direction of dependency), but is advocated by Hudson (1984), and turns out to be compatible with the categorial grammar account of dependency to be presented in the next section, so I shall adopt it here.

Finally, consider 'sequence of tenses':

(3.44) a. **Fred thinks Bill has left.**

b. **Fred thought Bill had left.**

c. ***Fred thought Bill has left.**

d. ***Fred thinks Bill had left.**

In this case the tense of the embedding verb has an effect on the tense of the embedded verb; by the strict definition in (3.35) this would actually count as agreement rather than government.

In order to determine the direction of dependency in cases where it is unclear, such as between subjects and verbs, we must consider other facts about dependency structure. For example, in **Fred thinks Bill laughed** we have already ascertained that **laughed** depends on **thinks**; by the single-head requirement **laughed** cannot therefore be dependent on **Bill**, so **Bill** must be dependent on **laughed**. We may thus assume in general (in main as well as subordinate clauses) that subjects depend on verbs rather than vice versa.

Generally, when words are not in a dependency relation they do not exhibit covariation (and hence do not exhibit agreement or government). For example, the subject and object of a verb do not in general covary; given a sentence like **John loves Mary**, there is no possible

change in the subject that could force a change in the object, or vice versa. Similarly **spoke** and **Mary** do not covary in **John spoke to Mary**, and **thinks** and **Bill** do not covary in **Fred thinks Bill laughed**; in each case there is no way of changing one in such a way as to affect the form of the other.

But in fact covariation cannot be regarded as a sufficient test for dependency, since it is possible for words to covary when there is no subcategorization relation, as in (3.45):

(3.45) a. **Mary respects herself.**

b. **John respects himself.**

c. ***Mary respects himself.**

d. ***John respects herself.**

Here the subject and the object of the verb covary, and indeed the subject governs the object in gender by the strict definition above. This might be argued to provide evidence for a dependency link from **Mary** to **herself**. However, the category of **herself** (as well as other features such as case) is still determined by the verb **respects**, and so by strict subcategorization the head of **herself** should be taken to be **respects**.

Similarly, in languages with a richer agreement system than English, such as French, covariation can be exhibited between subjects and predicative adjectives:

(3.46) a. **Marie est heureuse.** (Mary is happy-FEM)

b. **Jean est heureux.** (John is happy-MASC)

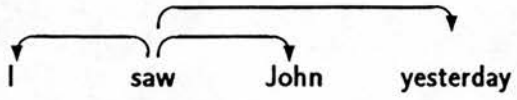
c. ***Marie est heureux.**

d. ***Jean est heureuse.**

Some work in dependency grammar has argued for a dependency link between subjects and predicative adjectives (see e.g. Matthews 1981), but in line with subcategorization it must be assumed that the verb is the head of both the subject and the predicative adjective, and hence that there is no dependency link between the two.

None of the above discussion has addressed modifier constructions. DG standardly deals with modifiers by assuming that they are optionally subcategorized for. For instance, in **I saw John yesterday**, **saw** is assumed to subcategorize for **yesterday** as well as **I** and **John**. This means that the dependency structure for this sentence is as below:

(3.47)



This is consistent with the usual assumptions of X-bar syntax, but at odds with the interpretation of categorial functors as heads, as we shall see in 3.3.3 when we try to combine the two approaches (cf. the discussion in 2.2.1). Nevertheless, I shall retain the standard dependency analysis of modifier constructions, rather than make concessions to CG. (Later we shall see how CG can be adapted to accommodate this change.) The reason for this will become clearer when we discuss dependency constituency, but for the moment note that if we assumed modifiers to be heads of the words they modify, then in the above example **yesterday** would be the head of **saw**, and so the root of the sentence would be **yesterday**, contradicting the general principle that the root of the sentence is a verb. Furthermore, if the sentence were embedded, e.g. **Fred thinks I saw John yesterday**, then the single-head requirement would force **yesterday** rather than **saw** to be dependent on **thinks**, for which there is no motivation (e.g. **thinks** and **yesterday** do not covary).

3.2.3 Dependency Constituents

We are now in a position to see how a linguistically motivated notion of flexible constituency can be derived from dependency structures. Previously, the only form of constituency considered in relation to DG was a rigid form (see e.g. Matthews 1981, pp. 84–93), which I shall refer to as *full constituency*; this will be discussed below. The new flexible form of constituency, called *dependency constituency*, was first proposed in Barry and Pickering (1990), and is discussed further in Pickering (1991).

The concept of a dependency constituent is straightforward. Given an expression and an associated dependency structure, the words in a dependency constituent must lie at

the nodes of a connected substructure (i.e. subgraph). This can be expressed formally as follows:

(3.48) Let $\mathcal{E}$ be an expression with an associated dependency structure $\mathcal{S}$, and $\mathcal{F}$ be a subexpression of $\mathcal{E}$. $\mathcal{F}$ is a *dependency constituent* of $\mathcal{E}$ iff there is a connected substructure (subgraph) $\mathcal{T}$ of $\mathcal{S}$ such that $\mathcal{F}$ consists of the words at the nodes of $\mathcal{T}$.

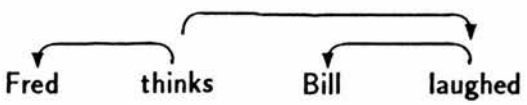
As with other forms of constituency, a dependency constituent is ***continuous*** if its words form a continuous substring of the original expression, and *discontinuous* otherwise. Clearly single words will always be dependency constituents, as will the entire expression; I shall refer to these particular examples as *trivial* and to all others as *non-trivial.*

To illustrate, let us return to the dependency structures for the three sample sentences given at the beginning of the section, and work out the associated sets of dependency constituents. First John spoke to Mary:

(3.19)

Here the non-trivial dependency constituents are John spoke, spoke to, to Mary, John spoke to and spoke to Mary. (These are all continuous, but this will not be the case in general.) Some of these (to Mary and spoke to Mary) are standard phrase-structure constituents, but the rest are not. Since (e.g.) spoke to and to Mary are both dependency constituents but overlap, dependency constituency is clearly flexible. In this case, in fact, all continuous strings are dependency constituents, which would be compatible with structural completeness. However, this is not in general true, as we can see by looking at Fred thinks Bill laughed:

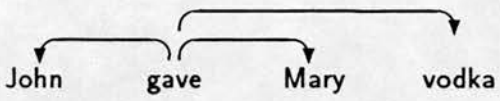
(3.21)



Here the non-trivial dependency constituents are **Fred thinks, thinks laughed, Bill laughed, Fred thinks laughed** and **thinks Bill laughed.** Again we have dependency constituents which are not phrase-structure constituents (e.g. **Fred thinks, thinks laughed**) and overlapping constituents (e.g. **Fred thinks** and **thinks Bill laughed**); in addition this time two of the dependency constituents are discontinuous (**thinks laughed, Fred thinks laughed**). However, there are now continuous substrings which are not dependency constituents, namely **thinks Bill** and **Fred thinks Bill**, which shows that dependency constituency is not structurally complete.

Most of the same points arise from the dependency structure for **John gave Mary vodka:**

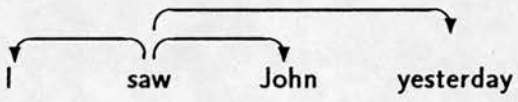
(3.22)



The non-trivial dependency constituents are **John gave, gave Mary, gave vodka, John gave Mary, John gave vodka** and **gave Mary vodka.** Again, **Mary vodka** is continuous but not a dependency constituent.

Our final example:

(3.47)



The non-trivial dependency constituents are **I saw, saw John, saw yesterday, I saw John, I saw yesterday** and **saw John yesterday.** In fact, if **yesterday** were the head of **saw**, the set of dependency constituents would be the same. However, there would be a difference if the sentence were embedded; for instance, in **Fred thinks I saw John yesterday**, with **saw** as the head of **yesterday**, **thinks saw** is a dependency constituent and **thinks yesterday** is not. On the other hand, if **yesterday** were the head of **saw**, **thinks saw** would not be a dependency constituent, but **thinks yesterday** would be. Since **yesterday** relates in meaning to **saw** and not to **thinks**, the version with modified elements as heads makes more sense linguistically.

Note that every dependency constituent contains a unique word that has no head within that dependency constituent; we may naturally extend the earlier terminology and refer to this as the *root* of the dependency constituent, since the root of the entire expression when viewed as a dependency constituent is the same as the root under the original definition. (The root of a single-word dependency constituent is of course that word.)

For instance, in (3.19) **John spoke, spoke to, John spoke to** and **spoke to Mary** all have root **spoke**, but **to Mary** has root **to**. Similarly in (3.21) **thinks laughed** has root **thinks**; **Bill laughed** has root **laughed**. In fact we may use the notion of root to construct an alternative definition of dependency constituent:

(3.49) Given a dependency structure and a word W in that dependency structure, a *dependency constituent rooted at* W is any expression $\mathcal{D}$ such that W is in $\mathcal{D}$, and the head of each word in $\mathcal{D}$ except W is also in $\mathcal{D}$.

Equivalently, there must be a head-dependent chain from W to each other word in $\mathcal{D}$, and every word in this chain must be in $\mathcal{D}$.

We may now use dependency constituency to motivate the rigid form of constituency referred to above as *full constituency*. Note first that for any word W the union of any two dependency constituents rooted at W is also a dependency constituent rooted at W. It follows that there must be a dependency constituent rooted at W of which all the others are subexpressions. We refer to this as the *full constituent rooted at* W. Formally:

(3.50) For any word W, the *full constituent rooted at* W is the (unique) expression $\mathcal{F}$ such that W is in $\mathcal{F}$, and any word except W is in $\mathcal{F}$ if and only if its head is in $\mathcal{F}$.

Equivalently, the full constituent rooted at W consists of W and every word subordinate to W. It follows that in a dependency structure that obeys the adjacency requirement all full constituents will be continuous. Note that the entire expression is a full constituent by this definition, but single words in general will not be.

For example, return to the expression **John spoke to Mary**, with the dependency structure in (3.19). There, the full constituent rooted at **John** is **John**, the full constituent rooted at **Mary** is **Mary**, the full constituent rooted at **to** is **to Mary**, and the full constituent rooted at **spoke** is **John spoke to Mary**. As this example illustrates, full constituency is rigid, but division into subconstituents is not exhaustive (since e.g. **spoke** is not a full constituent). Note that the set of full constituents is the same as the set of expressions that served as phrasal dependents in the phrase-structure representation (3.31) above. It will always be the case that when a dependency structure is derived from such a phrase-structure representation, the resulting set of full constituents will be the same as the original set of phrasal dependents. This will be discussed further in 3.3.2.

Note also the following:

(3.51) Every full constituent consists of its root plus zero or more full constituents.

For example, the full constituent **John spoke to Mary** consists of the root **spoke** plus the two full constituents **John** and **to Mary**; in turn, the full constituent **to Mary** consists of the root **to** plus the single full constituent **Mary**, and the full constituent **Mary** consists of the root **Mary** alone.

Finally, the following relationship holds between dependency constituency and full constituency:

(3.52) An expression is a dependency constituent iff it can be combined with zero or more full constituents to form a full constituent.

This is trivial for those dependency constituents that are also full constituents. Some examples: **Fred thinks** can be combined with **Bill laughed** to form **Fred thinks Bill laughed**, and **spoke to** can be combined with **John** and **Mary** to form **John spoke to Mary**. On the other hand, **Fred thinks Bill** cannot be combined with full constituents to form a full constituent (since **laughed** is not a full constituent).

3.2.4 A Formalization of Dependency Grammar

The above account of DG and dependency constituents has been purely descriptive. I shall conclude this section by proposing a simplified framework which might form the basis of a generative theory of DG. I shall not go into great detail since the purpose is mostly to motivate the connection with the categorial grammar account in the next section.

There is no standard formalism for representing DGs, although one has been proposed by Gaifman (1965). I shall now propose a grammatical framework which will generate dependency structures, based directly on subcategorization relations. The mechanisms resemble those of categorial grammar in that the combining properties of each word are completely encoded in its lexical entry, and the permissible modes of combination are entirely governed by general schemas. However, for simplicity's sake I shall completely ignore linear order and assume that expressions are multisets of words. (A multiset is an unordered list, so that order is irrelevant but number of occurrences is relevant.) We thus have no need for the adjacency requirement. Since dependency constituency is independent of word order, this simplified account will suffice for illustrative purposes. I shall introduce linear order when the CG formulation is introduced in the next section.

We assume some set of lexical categories. For illustrative purposes the following three will suffice: V (verb), PN (proper name), P (preposition). Each word has a *dependency type* of the form $H\{D_1;\ldots;D_n\}$, for some lexical categories $H, D_1, \ldots, D_n$ ($n \geq 0$). H (the *head category*) is the category of the word itself, and $D_1, \ldots, D_n$ (the *dependent categories*) are the categories of the words it subcategorizes for. For example:

(3.53)

John	PN{}
Mary	PN{}
Bill	PN{}
vodka	PN{}
laughed	V{PN}
to	P{PN}
spoke	V{PN;P}
thinks	V{PN;V}
gave	V{PN;PN;PN}

Expressions are generated by matching dependent categories against head categories. This can be stated by means of the following type-combining rule, where the commas between the dependency types are used to signify that order is irrelevant (cf. the ID/LP format for phrase structure rules (Pullum 1982)).

(3.54) $H\{D_1;\ldots;D_n\}\ ,\ D_1\{\}\ ,\ \ldots\ ,\ D_n\{\} \Rightarrow H\{\}$

The head category of the whole expression will be that of its root, so that in particular if the resultant category is V{}, the expression is a sentence. This can be illustrated by using a proof-figure-like notation for derivations:

(3.55)

John	spoke	to	Mary
		P{PN}	PN{}
PN{}	V{PN;P}	P{}	
V{}			

(3.56)

Fred	thinks	Bill	laughed
		PN{}	V{PN}
PN{}	V{PN;V}	V{}	
V{}			

(3.57)

John	**gave**	**Mary**	**vodka**
PN{}	V{PN;PN;PN}	PN{}	PN{}
	V{}		

On the same principles, it is possible to extend the system of categorization for words in a natural way so that it covers all dependency constituents. A dependency constituent will receive the dependency type $R\{D_1;\ldots;D_n\}$ where R is the category of the root and $D_1,\ldots,D_n$ are the categories of its missing dependents. So, for instance, **to Mary** would receive the type P{}; **John spoke**, V{P}; **spoke to**, V{PN;PN}. Note that a dependency constituent whose dependency type has no dependent categories is a full constituent.

This definition enables us to give a rule for deriving the dependency type for any dependency constituent from the types of its component parts (given that all words are dependency constituents). The rule says that two dependency constituents $\mathcal{D}_1$ and $\mathcal{D}_2$ may be combined by matching the head category of $\mathcal{D}_2$ against one of the dependent categories of $\mathcal{D}_1$; the head category of the result is the head category of $\mathcal{D}_1$, and the dependent categories of the result are the union of the remaining dependent categories of $\mathcal{D}_1$ and all the dependent categories of $\mathcal{D}_2$. Formally:

(3.58) $$X\{Y_1;\ldots;Y_k;\ldots;Y_m\}\ ,\ Y_k\{Z_1;\ldots;Z_n\}$$
$$\Rightarrow X\{Y_1;\ldots;Y_{k-1};Y_{k+1};\ldots;Y_m;Z_1;\ldots;Z_n\}$$

where $m \geq 1$, $n \geq 0$. Since the order of the Y_i is irrelevant, we may in fact state this slightly more simply:

(3.59) $$X\{Y_1;\ldots;Y_m\}\ ,\ Y_m\{Z_1;\ldots;Z_n\}$$
$$\Rightarrow X\{Y_1;\ldots;Y_{m-1};Z_1;\ldots;Z_n\}$$

So, for example, we may use this rule to derive all the following expressions as dependency constituents of the given types:


EX BIBL. UNIV. EDINBURGH

(3.60) a.

to	Mary
P{PN}	PN{}
P{}	

b.

John	spoke	to
PN{}	V{PN;P}	
V{P}		P{PN}
V{PN}		

(3.61) a.

thinks	Bill	laughed
	PN{}	V{PN}
V{PN;V}	V{}	
V{PN}		

b.

thinks	laughed
V{PN;V}	V{PN}
V{PN;PN}	

(3.62) a.

John	gave
PN{}	V{PN;PN;PN}
V{PN;PN}	

b.

gave	vodka
V{PN;PN;PN}	PN
V{PN;PN}	

On the other hand, we cannot perform the combinations in (3.63):

(3.63) a.

Fred thinks	Bill
V{V}	PN{}
***	

b.

Mary	vodka
PN{}	PN{}
***	

This is exactly in keeping with the analyses directly derived from the dependency structures.

The above formalization of course has many deficiencies. Not only does it ignore word order, but it cannot deal with constructions such as unbounded dependencies which are outside the scope of classical DG. It may be possible to add a linear precedence component to the grammar (see e.g. Pullum 1982), and to develop a richer type structure to deal with 'dependencies within dependencies', but this would take us outside the scope of this thesis. Rather than try to develop this formalization any further, it seems appropriate to use a categorial model similar to those already developed. The next section suggests how this might be achieved.

3.3 Dependency in Categorial Grammar

In this section I shall define a new categorial grammar **D**, which lies between **AB** and **L** in respect of the set of strings for which it generates types. I shall show that given certain assumptions about the relation between dependency types and categorial types, the legal combinations in **D** correspond to those operations which form continuous dependency constituents, leading to a more principled model of flexible constituency within categorial grammar.

3.3.1 A New Categorial Framework D

In the formalization of dependency grammar presented above, the key intuition was that dependent categories were cancelled against head categories to generate the types of dependency constituents. In much the same way, **D** generates the categorial types of dependency constituents by cancelling the 'argument' part of types against the 'value' part, in an analogous fashion to the procedure used in simplifying multiplication of fractions. For instance, in **D** the inference from X/Y and Y to X is valid, as is the inference from X/Y and Y/Z to X/Z; in each case the Y is cancelled. Slightly less obviously, in **D** the inference from $(Y\backslash X)/Z$ to $Y\backslash(X/Z)$ is valid; here, although no type symbols have been cancelled, no new type symbols have been introduced. But the inference from X to $Y/(X\backslash Y)$ is not valid in **D**, even though it is valid in **L**, since the new symbol Y has appeared in the conclusion. This is not meant to be a precise specification of **D**-valid inferences, but should provide a plausible motivation for the formalization to be presented below.

D can be formulated in the proof figure notation in a similar fashion to **L**, but with an important restriction. The /E and \E rules are exactly the same as in **L** (or **AB**):

(3.64) a. $\dfrac{\overset{\vdots}{X/Y} \quad \overset{\vdots}{Y}}{X}\,/\text{E}$ b. $\dfrac{\overset{\vdots}{Y} \quad \overset{\vdots}{Y\backslash X}}{X}\,\backslash\text{E}$

However, the /I and \I rules have an extra condition on their use: the discharged assumption must not serve as the functor in a /E or \E inference. Thus the rules may be stated as follows (cf. (2.12)):

(3.65) a. $\dfrac{\begin{array}{c}[Y]^i \\ \vdots \\ X\end{array}}{X/Y}/\mathrm{I}^i$ where Y does not serve as the functor in a /E or \E inference

b. $\dfrac{\begin{array}{c}[Y]^i \\ \vdots \\ X\end{array}}{Y\backslash X}\backslash\mathrm{I}^i$ where Y does not serve as the functor in a /E or \E inference

The semantic operation associated with the /I and \I rules is still lambda-abstraction, but in **D** it is only permitted to abstract over a variable if it does not act as a function over anything else.

Clearly all **AB** derivations are also valid in **D**, and yield the same meaning representations. For example:

(3.66) a. $\dfrac{\dfrac{\text{Bill}}{\text{NP}} \quad \dfrac{\text{laughed}}{\text{NP}\backslash\text{S}}}{\text{S}}\backslash\text{E}$

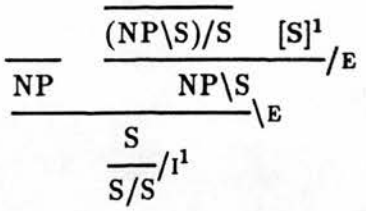
b. $\dfrac{\dfrac{\text{spoke}}{(\text{NP}\backslash\text{S})/\text{PP}} \quad \dfrac{\dfrac{\text{to}}{\text{PP/NP}} \quad \dfrac{\text{Mary}}{\text{NP}}}{\text{PP}}/\text{E}}{\text{NP}\backslash\text{S}}/\text{E}$

(3.67) a. **laughed Bill**

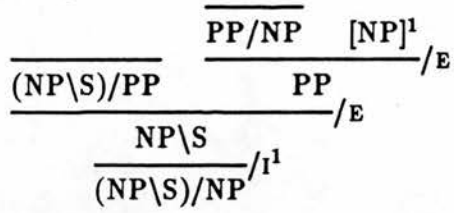
b. **spoke (to Mary)**

Also, derivations such as those in (3.68) are valid in **D**, since the assumptions conditionalized by the /I rule in each case serve as the argument in a /E inference, not the functor. Note also that in each of the meaning representations the abstracted-over variable is an argument, not a function.

(3.68) a.



b.

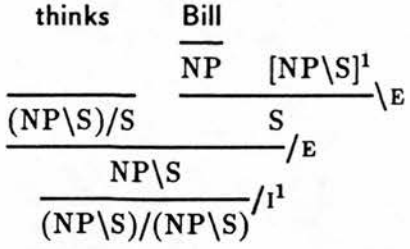


(3.69) a. λx[**(thinks** x**) Fred**]

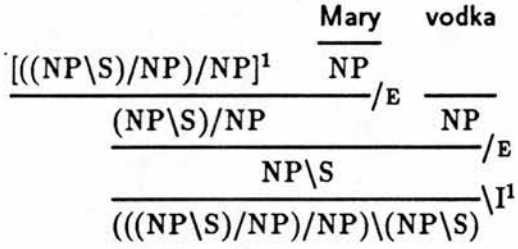
b. λx[**spoke (to** x**)**]

But the derivations in (3.70), though valid in **L**, are not valid in **D**, since they involve conditionalization of types that serve as the functor in /E or \E inferences:

(3.70) a.



b.



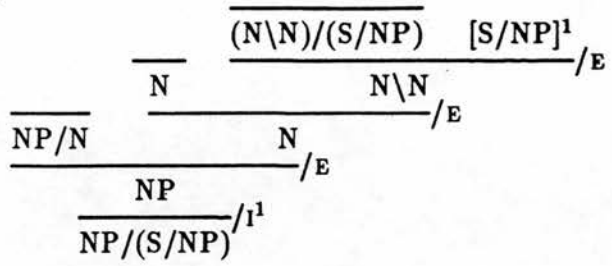
(3.71) a. λf[**thinks** (f **Bill**)]

b. λf[f **Mary John**]

Indeed, given the above lexical type-assignments there is *no* analysis of thinks Bill or Mary vodka within **D**, as should be evident from the 'cancellation' intuition.

Note carefully that it is permissible for derivations in **D** to involve conditionalization of functional *types*, so long as these types (and hence the associated variables) serve as arguments rather than functions. For instance, the following derivation is valid in **D**:

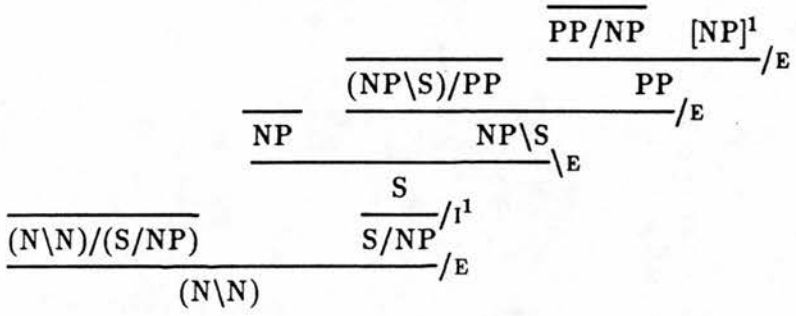
(3.72)



(3.73) $\lambda x^{\mathrm{NP}' \rightarrow \mathrm{S}'}$[**the** ((**who** x) **man**)]

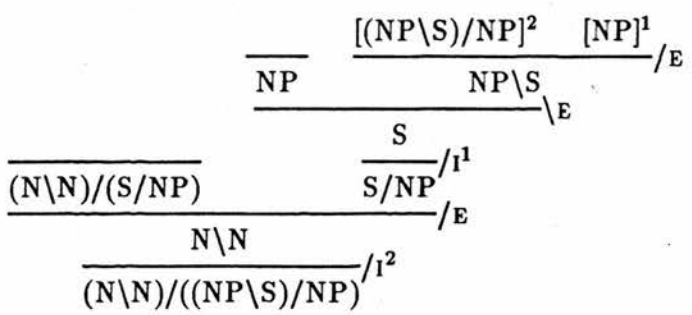
Finally, note that the restriction on /I and \I prohibits conditionalization even if the conditionalized type is a function over another conditionalized type. Thus, for instance, although the derivation in (3.74) is valid in **D**, those in (3.76) are not:

(3.74)

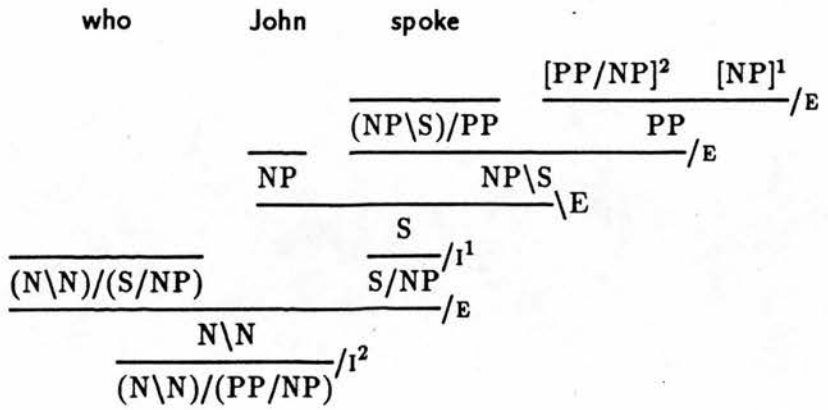


(3.75) **who** (λx[**spoke** (**to** x) **John**])

(3.76) a.



b.



(3.77) a. λf[**who** (λx[f x **John**])]

b. λg[**who** (λx[**spoke** (g x) **John**])]

The derivations given already suggest that expressions derivable by **D** correspond at least roughly to dependency constituents. I shall now consider what assumptions are needed to make this correspondence precise.

3.3.2 Interpreting Types in D

In the light of the discussion in 2.2.1, a natural question to ask is what model of syntactic structure we should assume in order to allow an interpretation of types with respect to which **D** is sound and complete. Recall that with **AB** it is appropriate to assume that heads are distinguished from non-heads and combination is non-associative, whereas with **L** there is no distinction between heads and non-heads and combination is associative. **D** stands between the two, in that heads are distinguished but combination is (at least partly) associative.

Let us return to John spoke to Mary. As noted in (2.20), **AB** assumes the following rigid structure, where round brackets indicate head daughters (phrasal heads) and square brackets indicate non-head daughters (phrasal dependents):

(3.78) [[John] ((spoke) [(to) [Mary]])]

Suppose we ignore heads and distinguish only non-heads. Then the structure is as follows:

(3.79) [[John] spoke [to [Mary]]]

Assuming an associative concatenation operation, this suggests the following interpretation of types (cf. (2.21) and (2.23)):

(3.80) An expression is of type X/Y (resp. $Y\backslash X$) iff, whenever it is concatenated with a following (resp. preceding) non-head of type Y, the result is of type X.

So, for example, spoke to Mary has the type NP\S since it forms an expression of type S when combined with a preceding non-head of type NP; spoke has both the types (NP\S)/PP and NP\(S/PP) since it can be combined with a following non-head of type PP to form an expression of type NP\S, or with a preceding non-head of type NP to form an expression of type S/PP, and so on. But not all **L**-valid types are possible; for instance John does not have the type S/(NP\S), since it cannot be combined with a non-head to form a grammatical expression.

I conjecture that **D** is sound and complete with respect to this interpretation of types. As with **AB**, /E inferences represent the combination of a head with a following non-head, and \E inferences represent the combination of a non-head with a following head; /I and \I inferences allow the withdrawal of a non-head in leftmost or rightmost position.

As mentioned earlier, the set of non-heads (or phrasal dependents) in (3.78), namely John, Mary, to Mary and John spoke to Mary, is in fact identical to the set of full constituents that would be generated under the corresponding dependency representation. This would seem to limit the grammar to generating expressions from which full constituents are 'missing'. Given the result in (3.52), namely that an expression is a dependency constituent iff it can be combined with zero or more full constituents to form a full constituent, we might expect **D** to generate dependency constituents. Further, since (like **AB** and **L**) **D** can only generate continuous expressions, we might conjecture that it generates all and only continuous dependency constituents. I shall now demonstrate that under certain assumptions about the relation between categorial type-assignments and dependency type-assignments, this is indeed the case.

Our assumption is that atomic CG types correspond one-to-one with DG lexical categories in the following way: there is a bijective function ρ from the set of atomic CG types to the set of DG lexical categories such that the root of an expression of atomic CG type X has the DG lexical category $\rho(X)$, and the full constituent rooted at a word with DG lexical category $\rho(X)$ has the atomic CG type X.

For example, in the sample dependency grammar in 3.2.4 there were three lexical categories V, PN and P. These corresponded respectively to the atomic CG types S, NP and PP; for example, the root of a sentence is a verb, and the full constituent rooted at a verb is a sentence. Thus we have $\rho(\mathrm{S}) = \mathrm{V}$, $\rho(\mathrm{NP}) = \mathrm{PN}$, $\rho(\mathrm{PP}) = \mathrm{P}$. A more general grammar might have the following correspondence:

(3.81)

Atomic CG type	DG lexical category
S (Sentence)	Finite verb
NP (Noun phrase)	Determiner or proper name
N (Common noun)	Lexical noun
PP (Prepositional phrase)	Preposition
VP (Non-finite verb phrase)	Non-finite verb
SP (Complementized sentence)	Complementizer

Thus this correspondence is reasonable for the types and categories already postulated (although it breaks down for modifiers, for reasons to be given in 3.3.3). From such a correspondence it follows that *all* full constituents receive atomic CG types. This is reasonable for the fragment of CG equivalent to classical DG (outside which full constituents are not defined anyway). It can now be shown that an expression can be derived within **D** iff it is continuous and can be given a dependency type within the framework of 3.2.4.

Suppose first that an expression $\mathcal{E}$ has the dependency type $H\{D_1, \ldots, D_n\}$, for some lexical categories $H, D_1, \ldots, D_n$ $(n \geq 0)$. Then for some atomic CG types $X, Y_1, \ldots, Y_n$ we have $\rho(X) = H, \rho(Y_1) = D_1, \ldots, \rho(Y_n) = D_n$. Since any expression of type Y_i is a full constituent with root of category D_i (for each i), $\mathcal{E}$ can be combined with full constituents of types $Y_1, \ldots, Y_n$ to form a full constituent. Furthermore, the resulting full constituent

will have a root of category H and hence will be of type X. Since $\mathcal{E}$ is continuous, it can therefore be generated within **D** by deriving type X and conditionalizing types $Y_1, \ldots, Y_n$.

Conversely, suppose that an expression $\mathcal{E}$ can be generated within **D** by deriving type X and conditionalizing types $Y_1, \ldots, Y_n$ ($n \geq 0$). $\mathcal{E}$ is continuous since it can be generated within **D**. $\mathcal{E}$ can be combined with full constituents of types $Y_1, \ldots, Y_n$ to form a full constituent of type X, $\mathcal{F}$ say. Let $\rho(X) = H, \rho(Y_1) = D_1, \ldots, \rho(Y_n) = D_n$. Then $\mathcal{F}$ has a root of type H, and so $\mathcal{E}$ can be given the dependency type $H\{D_1, \ldots, D_n\}$.

The main consequence of this can be stated as follows: the type given to a dependency constituent, within either the CG or the DG formalization, contains information about the categories of the root and the missing dependents, but no other information about dependency structure. (The CG type also contains information about linear order.) For example, the expression **John spoke to** receives the type V{PN} in DG and the corresponding type S/NP in CG. Either type tells us that the root is a verb and the missing dependent is nominal, but other aspects of the structure are hidden. This will play a role in the account of coordination to be presented at the end of the section.

Note that derivations involving conditionalization of functional types, such as (3.72), are outside the scope of this correspondence since they assume non-heads (corresponding to full constituents) which are not of atomic types. This is needed because of the presence of higher-order functions in the lexicon (i.e. functions which take other functions as arguments), such as **who** of type (N\N)/(S/NP). This makes the categorial treatment of dependency constituents more powerful than the straightforward DG treatment in this respect. However, the existence of such higher-order types in CG suggests that the DG type formalism might be extended in a similar fashion, so that **who** could be given a type of the form Rel{V{PN}} (where 'Rel' is a nonce-symbol for the lexical category of relative pronouns). I shall not pursue this here.

3.3.3 Deficiencies of D

There are still several deficiencies of the categorial account of dependency constituents just presented. The most obvious problem with **D** as it stands is that discontinuous depen-

dency constituents are not derivable, since all expressions must be continuous strings. For instance, it is impossible to give a type to the discontinuous dependency constituent **thinks laughed** in **Fred thinks Bill laughed**, since non-peripheral types cannot be conditionalized:

(3.82)
$$\dfrac{\dfrac{\textbf{thinks}}{(NP\backslash S)/S} \quad \dfrac{[NP] \quad \dfrac{\textbf{laughed}}{NP\backslash S}\backslash_E}{S}/_E}{\underline{NP\backslash S}_{***}}$$

A solution to this will be proposed when we consider commutation modalities in the next chapter.

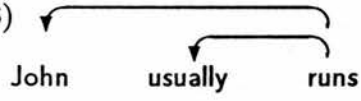
The other main problem is that (as already noted) modifiers must be taken as arguments rather than functors for the correspondence with DG to work. Consider for example:

(3.83) John runs fast

(3.84) John usually runs

Given that **fast** and **usually** are dependents of **runs** in the two cases, the dependency structures clearly show that the underlined expression in (3.83) is a dependency constituent, whereas that in (3.84) is not:

(3.85)


(3.86)


However, the standard categorial view of modifiers as functors means that under the usual type-assignments **D** makes the opposite prediction, since (3.87) is invalid in **D** but (3.88) is valid:

(3.87) **John** **runs**

$$\dfrac{\dfrac{}{\text{NP}}\quad\dfrac{\dfrac{\overline{\text{NP}\backslash\text{S}}\quad[(\text{NP}\backslash\text{S})\backslash(\text{NP}\backslash\text{S})]^1}{\text{NP}\backslash\text{S}}\backslash\text{E}}{\text{S}}\backslash\text{E}}{\text{S}/((\text{NP}\backslash\text{S})\backslash(\text{NP}\backslash\text{S}))}/\text{I}^1$$

(3.88) **John** **usually**

$$\dfrac{\dfrac{}{\text{NP}}\quad\dfrac{\dfrac{\overline{(\text{NP}\backslash\text{S})/(\text{NP}\backslash\text{S})}\quad[\text{NP}\backslash\text{S}]^1}{\text{NP}\backslash\text{S}}\backslash\text{E}}{\text{S}}/\text{E}}{\text{S}/(\text{NP}\backslash\text{S})}/\text{I}^1$$

The most transparent way of dealing with this is to reflect dependency relations by making modified elements into functors and modifiers into arguments, although this requires multiple type-assignments to modifiable elements. For example, suppose verbal modifiers have special atomic types instead of their usual CG types, say PoV for verbal postmodifiers like **fast** and PrV for verbal premodifiers like **usually**. Then **runs** would receive the type NP\S when not modified, (NP\S)/PoV when postmodified, and PrV\(NP\S) when premodified. Now (3.89) is valid in **D** and (3.90) is invalid, as desired:

(3.89) **John** **runs**

$$\dfrac{\dfrac{}{\text{NP}}\quad\dfrac{\dfrac{\overline{(\text{NP}\backslash\text{S})/\text{PoV}}\quad[\text{PoV}]^1}{\text{NP}\backslash\text{S}}/\text{E}}{\text{S}}\backslash\text{E}}{\text{S}/((\text{NP}\backslash\text{S})\backslash(\text{NP}\backslash\text{S}))}/\text{I}^1$$

(3.90) **John** **usually**

$$\dfrac{\dfrac{}{\text{NP}}\quad\dfrac{\dfrac{\overline{\text{PrV}}\quad[\text{PrV}\backslash(\text{NP}\backslash\text{S})]^1}{\text{NP}\backslash\text{S}}\backslash\text{E}}{\text{S}}/\text{E}}{\text{S}/(\text{PoV}\backslash(\text{NP}\backslash\text{S}))}/\text{I}^1$$

However, such multiple type-assignments are extremely inelegant. Furthermore, since a modifiable element may (at least in theory) combine with arbitrarily many modifiers, we

in fact need an infinite number of type-assignments for each modifiable element. For example, if nominal premodifiers such as **old** and **happy** have the type PrN, and nominal postmodifiers such as **with red hair** and **who likes cheese** have the type PoN, then a noun like **man** would have to have the alternative lexical types N, PrN\N, PrN\(PrN\N), ..., and also N/PoN, (N/PoN)/PoN, ..., as well as two-ways modifiable types like (PrN\N)/PoN. A solution will be proposed when we consider iteration and optionality modalities in the next chapter, but for the time being I shall simply assume that in any modifier construction the appopriate lexical type-assignment for the modified element is available.

3.3.4 Linguistic Applications of D

The clearest application of **D** is to coordination (see also Barry and Pickering (1990) and Pickering (1991)). Here I shall limit my attention to coordination of continuous expressions. Consider first the following 'naive' account of coordination based on **L** types. (An equivalent account is proposed by Moortgat (1988)).

(3.91) Two (or more) strings can be coordinated if and only if they can be given the same **L** type.

This holds in a large class of cases, covering both standard constituent coordination and many examples of what is generally termed 'non-constituent' coordination. To illustrate, examples will be annotated by placing square brackets round each conjunct and writing the minimal shared **L** type as a subscript to each. For example:

(3.92) a. John $[\text{loves Mary}]_{NP\backslash S}$ and $[\text{hates Sue}]_{NP\backslash S}$.

b. John $[\text{kissed Mary}]_{NP\backslash S}$ and $[\text{smiled}]_{NP\backslash S}$.

(3.93) a. John $[\text{will buy}]_{(NP\backslash S)/NP}$ and $[\text{may eat}]_{(NP\backslash S)/NP}$ the beans.

b. John $[\text{bought}]_{(NP\backslash S)/NP}$ and $[\text{may eat}]_{(NP\backslash S)/NP}$ the beans.

c. John $[\text{bought}]_{(NP\backslash S)/NP}$ and $[\text{paid for}]_{(NP\backslash S)/NP}$ the beans.

d. John $[\text{bought}]_{(NP\backslash S)/NP}$ and $[\text{thinks that Mary will eat}]_{(NP\backslash S)/NP}$ the beans.

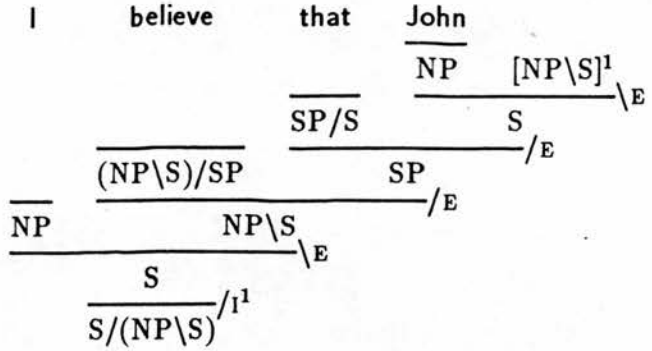
In examples (3.92) the conjuncts are phrase structure constituents, whereas in examples (3.93) they are not, but the same principle allows the coordinations in all of them. Furthermore, in each case the internal structure of the conjuncts appears to be irrelevant. However, this is not always the case, as demonstrated by the contrast in (3.94):

(3.94) a. **[I believe that John]$_{S/(NP\backslash S)}$ and [Harry thinks that Mary]$_{S/(NP\backslash S)}$ climbed the mountain.**

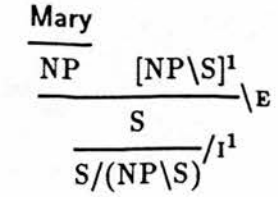
b. ***[John]$_{S/(NP\backslash S)}$ and [Harry thinks that Mary]$_{S/(NP\backslash S)}$ climbed the mountain.**

(3.94a) is correctly generated by (3.91), but (3.94b) is also incorrectly generated, since the type S/(NP\S) can be given to both conjuncts in each example:

(3.95) a.



b.



The data suggest that, contrary to the earlier examples and in defiance of (3.91), the internal structure of the conjuncts is relevant here. This contrast is repeated in examples with modifiers, such as the two pairs below (remember that modifiers in **L** are treated in the usual CG fashion as functions):

(3.96) a. **I ate [two small]$_{NP/N}$ and [three large]$_{NP/N}$ oranges.**

b. ***I ate [two small]$_{NP/N}$ and [three]$_{NP/N}$ oranges.**

(3.97) a. **John loves [Mary madly]$_{((NP\backslash S)/NP)\backslash(NP\backslash S)}$ and [Sue passionately]$_{((NP\backslash S)/NP)\backslash(NP\backslash S)}$.**

b. ***John loves [Mary madly]$_{((NP\backslash S)/NP)\backslash(NP\backslash S)}$ and [Sue]$_{((NP\backslash S)/NP)\backslash(NP\backslash S)}$.**

(3.91) correctly generates the first of each pair, where the conjuncts have a 'parallel' structure, but incorrectly generates the second, where they do not.

(3.91) also incorrectly generates examples such as the following (the so-called 'Dekker Paradox'):

(3.98) *[I think that]$_{(S/(NP\backslash S))/NP}$ and [a friend of]$_{(S/(NP\backslash S))/NP}$ Mary left.

The minimal type for **I think that** is S/S, whereas the minimal type for **a friend of** is NP/NP, but both of these types can be mapped to the same type (S/(NP\S))/NP, as the derivations below demonstrate:[3]

(3.99) a. **I think that**

$$\dfrac{\dfrac{\dfrac{}{NP}\quad\dfrac{\dfrac{}{(NP\backslash S)/SP}\quad\dfrac{\dfrac{}{SP/S}\quad\dfrac{[NP]^1\quad[NP\backslash S]^2}{S}\backslash E}{SP}/E}{NP\backslash S}\backslash E}{S}/I^2}{(S/(NP\backslash S))/NP}/I^1$$

b. **a friend of**

$$\dfrac{\dfrac{\dfrac{}{NP/N}\quad\dfrac{\dfrac{}{N/PP}\quad\dfrac{\dfrac{}{PP/NP}\quad[NP]^1}{PP}/E}{N}/E}{NP}\quad[NP\backslash S]^2}{\dfrac{S}{S/(NP\backslash S)}/I^2}\backslash E}{(S/(NP\backslash S))/NP}/I^1$$

---

[3]In fact this overgeneration is systematic, since (3.91) entails the following: for any types X, Y, Z and any (possibly empty) sequences of types Γ, Δ, if $\Gamma\ X\ \Delta \Rightarrow Z$ and $\Gamma\ Y\ \Delta \Rightarrow Z$ then $\Gamma\ X\ \text{Conj}\ Y\ \Delta \Rightarrow Z$ (where Conj represents the type of the conjunction).

However, if we turn to **D** rather than **L** as a basis for typing expressions, then it is possible to formulate a coordination schema which captures the data considerably better. The schema runs as follows:

(3.100) Two (or more) strings can be coordinated if and only if, for some sequence of **D** types $\langle X_1, \ldots, X_n \rangle$ $(n \geq 1)$, each string can be partitioned into a sequence of dependency constituents of those types.

In the case where $n = 1$, this reduces to the claim that two or more strings can be coordinated if they are dependency constituents of the same **D** type. This covers all the cases in (3.92) and (3.93) above, and many others besides. For instance:

(3.101) [John loves]$_{S/NP}$, and [Mary thinks Fred hates]$_{S/NP}$, sonatas by Mozart.

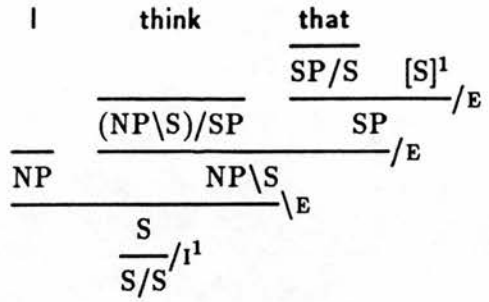
(3.102) That's a sonata which John [loves]$_{(NP\backslash S)/NP}$ and [thinks Fred hates]$_{(NP\backslash S)/NP}$.

(3.103) I [put one foot on]$_{(NP\backslash S)/NP}$, and [put the other foot on a box on]$_{(NP\backslash S)/NP}$, the large table in the hall.

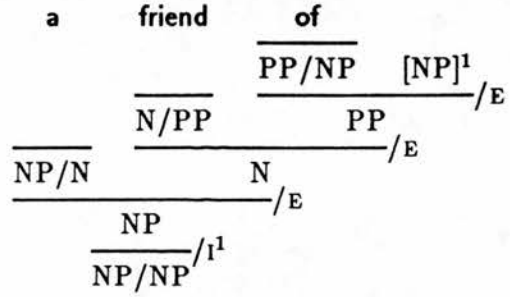
On the other hand, the claim that the conjuncts have to have the same **D** type correctly rules out (3.98) above (repeated with **D** types as (3.104)), since although the conjuncts there can be given a shared type within **L**, they must receive *different* types within **D**, as demonstrated by (3.105):

(3.104) *[I think that]$_{S/S}$ and [a friend of]$_{NP/NP}$ Mary left.

(3.105)a.



b.



Note that **D** cannot give any types to these expressions other than those illustrated. The easiest way to see this is to think in terms of cancellation; for example, in the sequence of types ⟨NP/N, N/PP, PP/NP⟩ the two N's and the two PP's can be cancelled, but nothing else.

The cases of (3.100) where $n \geq 1$ are based on a similar principle to the product-based coordination account of Wood (1988) (and cf. Williams 1978, Goodall 1987, Oehrle 1987, Bouma 1989). The important point to note here is that the conjuncts are not themselves dependency constituents; instead, two strings are coordinable if it is possible to subdivide each into a sequence of dependency constituents which can be matched type for type.[4] I shall illustrate such coordinations by exhaustively dividing each conjunct into the minimum number of dependency constituents (this division can be shown to be unique). Each conjunct is placed in square brackets, and additionally the individual dependency constituents within each conjunct are placed in round brackets and subscripted with their

[4]The schema as formulated here thus fails to generate such coordinations where one of the conjuncts is itself a coordination, e.g. John loves [Mary madly and Sue passionately] or [Jane intensely]. This is an oversight.

type in **D**. With this notation the earlier examples read as follows (treating modifiers as arguments):

(3.106) a. [(I believe that)$_{S/S}$ (John)$_{NP}$] and [(Harry thinks that)$_{S/S}$ (Mary)$_{NP}$] climbed the mountain.

b. *[(John)$_{NP}$] and [(Harry thinks that)$_{S/S}$ (Mary)$_{NP}$] climbed the mountain.

(3.107) a. I ate [(two)$_{NP/N}$ (small)$_{PrN}$] and [(three)$_{NP/N}$ (large)$_{PrN}$] oranges.

b. *I ate [(two)$_{NP/N}$ (small)$_{PrN}$] and [(three)$_{NP/N}$] oranges.

(3.108) a. John loves [(Mary)$_{NP}$ (madly)$_{PoV}$] and [(Sue)$_{NP}$ (passionately)$_{PoV}$].

b. *John loves [(Mary)$_{NP}$ (madly)$_{PoV}$] and [(Sue)$_{NP}$].

As a further example, contrast (3.103), where the conjuncts are dependency constituents, with (3.109), where they are not:

(3.109) I put [(one foot)$_{NP}$ (on)$_{PP/NP}$], and [(the other foot)$_{NP}$ (on a box on)$_{PP/NP}$], the large table in the hall.

There are a few possible exceptions to (3.100), for example:

(3.110) ?*John told [(Mary)$_{NP}$ (Bill)$_{NP}$] and [(Fred)$_{NP}$ (Sue)$_{NP}$] was coming.

In this case the problem may be due to the fact that two elements outside the coordination, told and was coming, are needed to complete a dependency constituent containing Mary and Bill (or Fred and Sue), whereas only one external element was needed in the previous examples: climbed the mountain in (3.106a), oranges in (3.107a), and loves in (3.108a). In contrast, sentences like (3.111) do seem acceptable:

(3.111) John told [(the mothers)$_{NP}$ (that)$_{\overline{S}/S}$ (their daughters)$_{NP}$] and [(the fathers)$_{NP}$ (that)$_{\overline{S}/S}$ (their sons)$_{NP}$] were all at the party.

Finally, note that we can only deal with 'unlike category coordination' (Sag, Gazdar, Wasow and Weisler 1985) if we assume that the conjuncts have some shared type. For instance, in (3.112), we might give the conjuncts the common type PredP:

(3.112) John is $[\text{lucky}]_{\text{PredP}}$ and $[\text{a rogue}]_{\text{PredP}}$.

On this assumption, we can generate the coordination below:

(3.113) John is $[(\text{lucky})_{\text{PredP}}$ (in the gambling hall)$_{\text{PP}}]$ and $[(\text{a rogue})_{\text{PredP}}$ (in the tavern)$_{\text{PP}}]$.

Other linguistic applications of **D** are more tentative. For example, the unacceptability of (3.114) suggests that extraction is subject to the restriction (at least in English) that the non-extracted part of the string must be a dependency constituent:[5]

(3.114) *How many boys do you think with red hair like football?

Applications of **D** have also been proposed to incremental interpretation (Pickering 1991) and to prosodic structure (van der Linden 1991, Bird 1991); in each case the claim is made that a dependency constituent forms a linguistic unit in some sense.

Finally, note that **D** may be of relevance in relation to computational parsing. Pure bottom-up parsing in **L** is in general impossible because infinitely many types can be derived from any given type (see Moortgat 1988). In contrast, it is evident, given the 'cancelling' intuition, that the set of types to which a sequence of types can be mapped within **D** is finite. This suggests that expressions can be parsed bottom-up within **D** even if the type of the entire expression is unknown. See Hepple (1991).

---

[5]Apparent counterexamples such as Which boy do you know that plays football? are presumably examples of extraposition.

Chapter 4

Extensions to Categorial Grammar

4.1 Linguistic and Logical Motivation

Throughout the last two chapters there have been several examples of constructions which it appeared that the mechanisms of frameworks such as **AB**, **L** and **D** were insufficient to describe. In this section I shall first present further linguistic evidence which suggests that the type system assumed in these frameworks is inadequate, and then present some logically inspired motivation for extending the frameworks. Together these will provide background for the extended categorial grammar to be presented in the next section.

4.1.1 Non-Sequential Constructions

The categorial frameworks we have considered so far (**AB**, **L** and **D**) have all used the same *directional* type system with the two connectives / and \. Although the interpretation of the connectives differs between the systems, in each case there is an assumption of *sequentiality*: expressions combine in a strict linear sequence. As we have already seen, this does not enable us to describe discontinuous constituents; in fact there are many different classes of constructions that cannot be described straightforwardly (or at all) if sequentiality is adhered to. These may be grouped under the headings of *commutation*, i.e. departures from canonical word order; *iteration*, i.e. use of an expression more than once, or combining with arbitrary numbers of complements; and *optionality*, i.e. non-use of an expression, or combining with optional complements.

We can find examples of all three of these by looking at extraction constructions. Consider commutation first. In chapter 2 an account of unbounded relativization within **L** (also valid within **D**) was proposed which involved giving the type (N\N)/(S/NP) to object relative pronouns such as **who(m)**, generating all the constructions in (4.1) (repeated from (2.31)):

(4.1) a. **the man who Mary likes**

b. **the man who Mary speaks to**

c. **the man who Mary says that John likes**

However, within either **L** or **D**, the type S/NP can only be given to expressions that consist of a sentence missing an NP from the rightmost position. Thus this type will not generate constructions such as in (4.2):

(4.2) a. **the man who Mary introduced to Sue**

b. **the man who Mary greeted warmly**

The only way of generating the constructions in (4.2) in **L** or **D** is to give an extra type to the relative pronoun so that it combines with both parts of the clause body separately (for instance **who** might have the type ((N\N)/PP)/((S/PP)/NP) in (4.2a)). However, this fails to capture the obvious generalization that, in all the constructions in both (4.1) and (4.2), the body of the relative clause consists of a sentence missing an NP from *some* position (not necessarily peripheral). Thus it would be desirable if we could find a single way of characterizing all expressions that have this property, which is impossible under an assumption of adjacency.

Turning to iteration, consider the problem of multiple extraction, such as in the parasitic gap construction in (4.3). (We are not concerned here with linguistic constraints on where parasitic gaps may occur.)

(4.3) **the man who Mary passed without greeting**

Here the body of the relative clause is missing two NPs. Again, this could be handled in an *ad hoc* fashion by giving extra types to the relative pronoun. However, the fact that such constructions can be recursed means that (at least in theory) the body of the relative clause may be missing arbitrarily many NPs (from one upwards):

(4.4) a. **the man who Mary passed**

b. **the man who Mary passed without greeting**

c. **the man who Mary passed without greeting after seeing**

d. **the man who Mary passed without greeting after seeing before recognizing**

Thus who would have to receive infinitely many lexical types under this analysis.

Finally, extraction may be optional, as in the following example:

(4.5) a. The lecture was too long for Mary to follow.

b. The lecture was too long for Mary to concentrate.

Once more, this could be handled by multiple type-assignments, but a generalization would be lost.

Commutation, iteration and optionality also occur in the description of complement constructions. For instance, a head may combine with a complement which is not adjacent to it, as in (4.6b):

(4.6) a. A critique of HPSG appeared.

b. A critique appeared of HPSG.

Assuming on the basis of (4.6a) that critique has the lexical type N/PP, (4.6b) could be dealt with within **AB**, **L** or **D** by giving appeared an additional type such as ((NP/PP)\S)/PP. However, this goes against normal assumptions about syntactic structure; specifically, the assignment of this type within **D** would mean that of HPSG was a dependent of appeared rather than critique. (This problem corresponds to a violation of the adjacency requirement within dependency grammar.)

Iterated and optional complements are also relatively easy to find. The simplest example of iterated complements occurs if we regard coordinators as heads taking arbitrary numbers of complements:

(4.7) a. John and Mary left.

b. Sue, John and Mary left.

c. Bill, Sue, John and Mary left.

d. Jane, Bill, Sue, John and Mary left.

Clearly **and** would require an infinite number of lexical types to deal with this. Similarly, optional complements are found in examples like (4.8) and (4.9), where **ate** and **thought** would require two types each:

(4.8) a. John ate the apple.

b. John ate.

(4.9) a. The thought that Mary was dead appalled him.

b. The thought appalled him.

Another instance of both iteration and optionality comes about when we try to incorporate dependency into categorial grammar, as in the previous chapter. Recall that modifiers are standardly treated as dependents by dependency grammar, which means that a strict dependency interpretation of **D** would require them to be treated as arguments rather than as functors. However, modifiers are well known to be both iterable and optional, as illustrated by (4.10) and (4.11) respectively:

(4.10) a. John arrived suddenly.

b. John arrived suddenly in a taxi.

c. John arrived suddenly in a taxi with a suitcase.

d. John arrived suddenly in a taxi with a suitcase at six o'clock.

(4.11) a. **John arrived suddenly.**

b. **John arrived.**

Thus if we want to avoid problems such as those illustrated in 3.3.3 we need some mechanism to deal with iteration and optionality.

The motivation for including some sort of 'non-sequential' mechanism in the grammar should be clear by now. Before moving on to the proposed linguistic account, I shall consider the logical motivation for extending frameworks such as **L** in this way.

4.1.2 The Categorial Hierarchy

From a logical perspective, **L** can be seen as the weakest of a hierarchy of logics which differ in the amount of freedom allowed in the use of assumptions. **L** as it stands forms a 'sequential logic', which is essentially the single-conclusioned, implicational fragment of *non-commutative linear logic* (see Girard 1989; Abrusci 1989).[1] The highest logic in the hierarchy corresponds to the implicational fragment of the calculus *LJ* introduced in Gentzen (1936). Gentzen formulated this calculus in terms of sequences of propositions, and then provided explicit *structural rules* to show the permitted ways to manipulate these sequences. The structural rules are *permutation*, which allows the order of the assumptions to be changed; *contraction*, which allows an assumption to be used more than once; and *weakening*, which allows an assumption to be ignored. The inclusion or exclusion of these rules results in a hierarchy of logics sometimes known as the 'categorial hierarchy', discussed in e.g. van Benthem (1987), Moortgat (1988), Ono (1988) and Wansing (1989).

In standard natural deduction style formulations of these logics, where the proof is organized as a tree, structural operations are implicit in the treatment of occurrences of assumptions and the conditions on their discharge (e.g. the usual natural deduction presentation of intuitionistic logic). However, for present purposes it will be more appropriate to

---

[1] However, proofs from no premises are permissible in non-commutative linear logic.

deal with structural rules by explicitly including them in proof figures (since it will enable a more direct motivation for the structural modalities to be described in the next section). This means that we have to extend the proof figure notation introduced in 2.1.2.

Recall that a tree structure was imposed on proof figures by requiring each inference rule to have a single formula below the line. However, suppose that we now allow inference rules to have an arbitrary number of formulas (zero or more) below the line, so that the general version of a rule is as in (4.12) (where $m \geq 0$, $n \geq 0$):

(4.12)
$$\frac{\vdots \qquad\qquad \vdots}{X_1 \quad \cdots \quad X_m \qquad\qquad Y_1 \quad \cdots \quad Y_n}R$$

However, we wish to retain the requirement that proofs should have a single conclusion, and so we impose a global condition on proofs by requiring that the final inference used in a proof must have exactly one formula below the line. Alternatively, we could view (4.12) not as an inference rule in itself, but as a shorthand for the following (for some formula Z):

(4.13)
$$\frac{\vdots \quad \frac{\vdots \qquad\qquad \vdots}{X_1 \quad \cdots \quad X_m \qquad\qquad Y_1 \quad \cdots \quad Y_n}R \quad \vdots}{Z}$$

(4.13) essentially says that if an inference rule has more than one formula below the line, all these formulas must all be used lower down the proof, so that the proof has a single conclusion.

The 'Lambek-van Benthem' calculus **LP** (van Benthem 1983, 1986) is like **L** except that the order of premises is irrelevant. **LP** is essentially the implicational fragment of Girard's (1987) *linear logic*. In the proof figure notation, we may obtain **LP** from **L** by adding a structural rule of *permutation* as follows:

(4.14)
$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ X & Y\end{array}}{\begin{array}{cc}Y & X\end{array}}\text{PRM}$$

(Since this rule is not single-conclusioned, it may not form the last inference in a proof.) Addition of the permutation rule means that the directional implicational types X/Y, $Y\backslash X$ are equivalent, since they are mutually derivable:

(4.15)
$$\dfrac{\dfrac{\dfrac{[Y]^1 \quad X/Y}{\begin{array}{cc}X/Y & Y\end{array}}\text{PRM}}{X}/\text{E}}{Y\backslash X}\backslash\text{I}^1 \qquad \dfrac{\dfrac{\dfrac{Y\backslash X \quad [Y]^1}{\begin{array}{cc}Y & Y\backslash X\end{array}}\text{PRM}}{X}\backslash\text{E}}{X/Y}/\text{I}^1$$

For the purposes of **LP** (and higher logics), therefore, we need use only a single type for the two, which I shall notate in the more conventional fashion as $Y{\rightarrow}X$. The elimination and introduction rules for $\rightarrow$ will be given as for $\backslash$ (though the rules for $/$ would have done as well).

(4.16)
$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ Y & Y{\rightarrow}X\end{array}}{X}{\rightarrow}\text{E} \qquad \dfrac{\begin{array}{c}[Y]^i \\ \vdots \\ X\end{array}}{Y{\rightarrow}X}{\rightarrow}\text{I}^i$$

Condition on $\rightarrow$I: Y is the leftmost undischarged assumption in the proof of X

Without loss of generality, rule applications can be kept subject to the previous ordering conditions for $\backslash$. For example, we may derive the type $Y{\rightarrow}(X{\rightarrow}Z)$ from $X{\rightarrow}(Y{\rightarrow}Z)$ in **LP** as follows:

(4.17)
$$\dfrac{\dfrac{\dfrac{\dfrac{[X]^1 \quad [Y]^2}{\begin{array}{cc}Y & X\end{array}}\text{PRM} \quad X{\rightarrow}(Y{\rightarrow}Z)}{Y{\rightarrow}Z}{\rightarrow}\text{E}}{Z}{\rightarrow}\text{E}}{X{\rightarrow}Z}{\rightarrow}\text{I}^1}{Y{\rightarrow}(X{\rightarrow}Z)}{\rightarrow}\text{I}^2$$

However, since the order of premises is irrelevant, the logic is unaffected by allowing the argument type Y in the $\rightarrow$E rule to occur either to the left or the right of the functor type

$Y{\rightarrow}X$, and Y to be *any* undischarged assumption above X in the $\rightarrow$I rule. This removes the need for an explicit permutation rule.

LPC is an extension of **LP** which allows assumptions to be used more than once, a property which it shares with *relevance logic* (Anderson and Belnap 1975). This may be achieved in the proof figure formulation by adding a structural rule of *contraction*:

(4.18)
$$\dfrac{\begin{array}{c}\vdots\\ X\end{array}}{X \quad X}\,\text{CON}$$

The $\rightarrow$E and $\rightarrow$I rules are as before in this formulation. By way of example of derivation in **LPC**, we may derive $X{\rightarrow}Y$ from $X{\rightarrow}(X{\rightarrow}Y)$ as follows:

(4.19)
$$\dfrac{\dfrac{\dfrac{[X]^1}{X \quad X}\,\text{CON} \qquad X{\rightarrow}(X{\rightarrow}Y)}{X{\rightarrow}Y}\,{\rightarrow}\text{E}}{Y}\,{\rightarrow}\text{E}$$
$$\dfrac{Y}{X{\rightarrow}Y}\,{\rightarrow}\text{I}^1$$

LPC could be alternatively formulated in a natural deduction style by relaxing the directionality conditions on **L** and allowing withdrawal of one or more occurrences of a conditionalized assumption. This removes the need for explicit rules of permutation and contraction.

Finally we may extend **LPC** to **LPCW**[2], where it is permissible for assumptions not to be used at all. It thus corresponds to the implicational fragment of *intuitionistic logic*. In the proof figures, this is achieved by adding a structural rule of *weakening*:

(4.20)
$$\dfrac{\begin{array}{c}\vdots\\ X\end{array}}{}\,\text{WKN}$$

Because this rule has no conclusion, in practice weakening will not be notated this way.

---

[2]This calculus is called **LPCE** in Moortgat (1988).

Instead the weakened assumption will be 'absorbed' from the left or the right as follows:

(4.21) a. $\dfrac{\begin{array}{cc}\vdots & \vdots \\ X & Y\end{array}}{Y}\,\mathrm{WKN_l}$ b. $\dfrac{\begin{array}{cc}\vdots & \vdots \\ Y & X\end{array}}{Y}\,\mathrm{WKN_r}$

For example, the derivation of $X{\rightarrow}Y$ from Y looks like this:

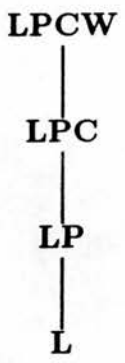
(4.22) $\dfrac{\dfrac{[X]^1 \quad Y}{Y}\,\mathrm{WKN_l}}{X{\rightarrow}Y}\,{\rightarrow}\mathrm{I}^1$

This derivation satisfies the ordering conditions on conditionalization since X is indeed the leftmost assumption dominating Y. **LPCW** could be alternatively formulated in a natural deduction style by relaxing the directionality conditions on **L** and allowing withdrawal of zero or more occurrences of a conditionalized assumption. This removes the need for explicit rules of permutation, contraction and weakening.

Note that for each of the three systems, if the $\rightarrow$E and $\rightarrow$I rules are interpreted as application and abstraction respectively, then we obtain a different subset of the lambda-calculus. This is a generalization of the Curry-Howard isomorphism for intuitionistic logic and full lambda-calculus (Curry and Feys 1958; Howard 1969). Exactly one assumption must be discharged by the $\rightarrow$I rule in **LP** (or **L**), and so we obtain the set of lambda-terms in which each variable occurrence is bound exactly once (the 'single-bind' lambda-calculus). In **LPC**, one or more assumptions may be discharged, and so the resulting fragment allows multiple binding but not vacuous abstraction. Finally **LPCW**, where zero or more assumptions may be discharged, corresponds to the full lambda-calculus with vacuous abstraction.

We may conveniently display the categorial hierarchy as in Figure 4–1 below:

Figure 4–1: The categorial hierarchy



4.2 Structural Modalities

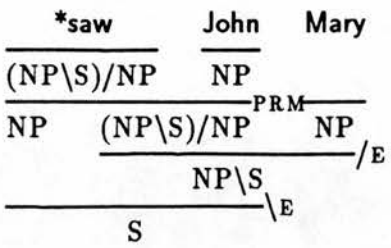
In this section I shall summarize work by Glyn Morrill, Neil Leslie, Mark Hepple and myself which extends the categorial type system to deal with non-sequentiality by defining operators called *structural modalities*, motivated by structural rules and the logics in the categorial hierarchy. Two alternative systems of modalities will be proposed, and some logical and linguistic properties of the two systems will be discussed.

4.2.1 Extending the Categorial Type System

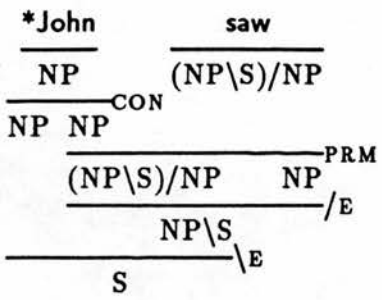
It is possible to deal with non-sequential linguistic phenomena in a very crude fashion simply by using structural rules; permutation for commutation, contraction for iteration, and weakening for optionality. But since these rules can be applied freely, the resulting calculi would overgenerate wildly. Permutation would imply that word order was irrelevant; contraction would imply that any word in a sentence could fulfil any number of roles; and weakening would imply that sentences could contain arbitrary words not contributing to their meaning. For example, (4.23a) would be generated by **LP**, (4.23b) by **LPC**, and

(4.23c) by **LPCW**:[3]

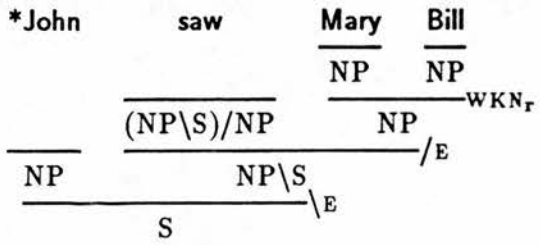
(4.23) a.



b.



c.



However, we have seen that individual linguistic elements may exhibit commutation, iteration or optionality, which suggests that we should be able to indicate that structural operations are permissible on *specific* types, while still forbidding them as general inference rules. There is a precedent for this in (commutative) linear logic (Girard 1987), which has a structural rule of permutation, but not contraction or weakening. In addition to the usual logical operators, linear logic has the so-called *exponentials* ! ('of course') and ? ('why not'), which bring back the structural rules of contraction and weakening in a controlled way. For instance, the formula $!X$ represents an item from which zero or more occurrences of the formula X may be derived, corresponding to both contraction and weakening. The structural modalities described here are in the spirit of Girard's exponentials, but are geared more to individual structural rules, and thus to specific linguistic operations.

Two different, though related, systems of structural modalities are proposed in Barry, Hepple, Leslie and Morrill (1991) and Morrill, Leslie, Hepple and Barry (1990). I shall

---

[3]For illustrative purposes we revert to a directional type system here, even though there is no difference in meaning between X/Y and $Y\backslash X$ in these systems.

describe both below, referring to them as the 'cumulative' system and the 'orthogonal' system respectively. The main difference is that the cumulative system defines a complete logic but its linguistic coverage is more restricted, whereas the orthogonal system has greater coverage but is not complete.

Structural modalities may be classified into *universal* and *existential* modalities. Broadly speaking the universal modalities may be said to represent 'any' (location, number of occurrences, etc.), and the existential modalities to represent 'some' (location, number of occurrences, etc.); in fact the terminology arises since the behaviour of the operators in each class bears a resemblance to that of the operators $\Box$ (universal modality) and $\Diamond$ (existential modality) in some modal logics (see below). The cumulative system as currently formulated contains only universal modalities; the orthogonal system contains both universal and existential modalities. For convenience I shall list the notation used for all the structural modalities in (4.24).[4]

(4.24)

	Cumulative	Orthogonal	
	Universal	Universal	Existential
Commutation	$\triangle$	$\triangleright$ and $\triangleleft$	$\rangle$ and $\langle$
Iteration	†	!	+
Optionality	‡	‖	?

Universal modalities have the common property that any item of a universally modal type also has the corresponding non-modal type, and the existential modalities have the property that any item of a non-modal type also has the corresponding existentially modal type. This means there are straightforward elimination rules for the universal modalities and introduction rules for the existential modalities, but the complementary rules in each case are less straightforward. Introduction rules have been formulated for the universal modalities in the cumulative system, making the logic complete, but universal introduction

[4]The notation for the cumulative iteration and optionality modalities has been changed from that given in Barry *et al.*, in order to avoid confusion with the notation for the corresponding orthogonal modalities.

and existential elimination rules have not been formulated for the orthogonal system. This does not appear to matter in practice, since it appears that the universal introduction and existential elimination rules are in fact rarely needed for linguistic purposes, as will become clear when we look at the examples below.

In general, universal modalities are appropriate for describing the behaviour of lexical items that subcategorize for expressions which are themselves missing commutable, iterable or optional arguments; such items typically receive types of the form $((\Box Z)\rightarrow Y)\rightarrow X$, where $\Box$ represents one of the universal modalities and $\rightarrow$ one of the implications. Conversely, existential modalities are appropriate for describing the behaviour of lexical items that subcategorize for commutable, iterable or optional arguments; such items typically receive types of the form $(\Diamond Y)\rightarrow X$, where $\Diamond$ represents one of the existential modalities. Note that all these operators are strictly formal devices and not geared towards specific linguistic phenomena, so that in many cases the modalities on their own will not be sufficient to capture all aspects of the behaviour of a given lexical item. Thus the proposed types may still lead to overgeneration in some cases; for instance the proposed types for relative pronouns do not take account of island constraints, and all notions of 'domains' or 'boundedness' will be ignored. (See Morrill (1989) for an approach to boundedness within CG.)

4.2.2 Cumulative Structural Modalities

For each of commutation, iteration and optionality, the cumulative system defines a single universal modality. For each universal modality there is an elimination rule, an introduction rule, and one or more so-called 'operational rules', which are essentially controlled versions of structural rules. The system is called 'cumulative' because as we proceed up the hierarchy each modality inherits all the inference rules of the previous one.

Commutation is achieved by means of the operator Δ. ΔX is the type of an item of type X which may be freely permuted; thus an occurrence of an item of type X in any position may be derived from an item of type ΔX. A type whose outermost operator is Δ will be referred to as a *Δ-type*. As we shall see, Δ-types (and other such modal types)

do not in general occur in the lexicon; they are usually only assumed within a derivation. Δ has the following inference rules:

(4.25)
$$\frac{\begin{array}{c}\vdots\\ \Delta X\end{array}}{X}\Delta\text{E} \qquad \frac{\begin{array}{c}\vdots\\ X\end{array}}{\Delta X}\Delta\text{I}$$

Condition on ΔI: every path to an undischarged assumption in the proof of X leads to an independent subproof of a Δ-type

$$\frac{\begin{array}{cc}\vdots & \vdots\\ \Delta X & Y\end{array}}{Y \quad \Delta X}\Delta\text{PRM} \qquad \frac{\begin{array}{cc}\vdots & \vdots\\ Y & \Delta X\end{array}}{\Delta X \quad Y}\text{PRM}\Delta$$

The ΔE rule is simply a consequence of the fact that an item of type ΔX is also of type X. The ΔPRM and PRMΔ rules are the usual structural rule of permutation, but restricted to cases where one of the types is a Δ-type. The ΔI rule is more complicated, since it has a side condition which says (essentially) that Δ-types can only be derived from assumptions which are all Δ-types.[5] The operator will be assumed to have identity semantics, so that the term associated with the input type to the ΔE rule will also be associated with the output type.[6]

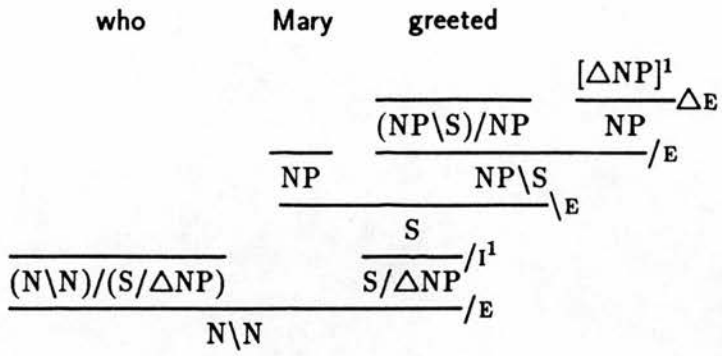
This operator makes it possible to deal with the earlier problems involving non-peripheral extraction. Recall that within **L** it is not possible to characterize expressions that consist of (say) a sentence missing an NP from *some* position. Within **L**+Δ, on the other hand, all such expressions may be given the type S/ΔNP (or equivalently ΔNP\S). This is because the type ΔNP may be assumed in rightmost (or leftmost) position and then permuted to whatever place is appropriate to the derivation. Thus assignment of the type (N\N)/(S/ΔNP) to the object relative pronoun **who(m)** allows both peripheral and non-peripheral extraction, as illustrated by the derivations in (4.26) and the terms in (4.27):

---

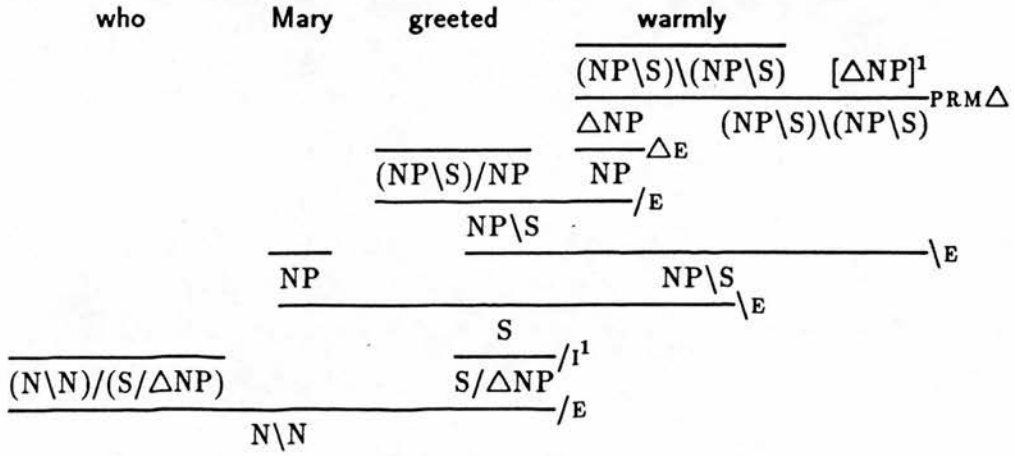
[5]I shall not be using the rule here, but for an illustration of its use see Hepple (1990).

[6]Notationally, unary operators will be assumed to bind tighter than binary ones, so that $\Delta X/Y$ means $(\Delta X)/Y$ rather than $\Delta(X/Y)$.

(4.26) a.



b.



(4.27) a. **who** (λx[(**greeted** x) **Mary**])

b. **who** (λx[(**warmly** (**greeted** x)) **Mary**])

This treatment partly resembles that of Moortgat (1988), who deals with such constructions by means of the extraction operator $\uparrow$; the type $X \uparrow Y$ is assigned to any string of type X missing a subpart of type Y somewhere within it. (In fact Moortgat later suggests (1990b) that the type $X \uparrow Y$ may in fact be definable in terms of permutation modalities.)

Iteration is achieved by means of the operator $\dagger$. $\dagger X$ is the type of an item of type X which may be freely permuted and iterated; thus one or more occurrences of items of type X in any position may be derived from an item of type $\dagger X$. $\dagger$ has the all the inference rules of $\triangle$, plus an analogue of the contraction rule which we shall call $\dagger$CON:

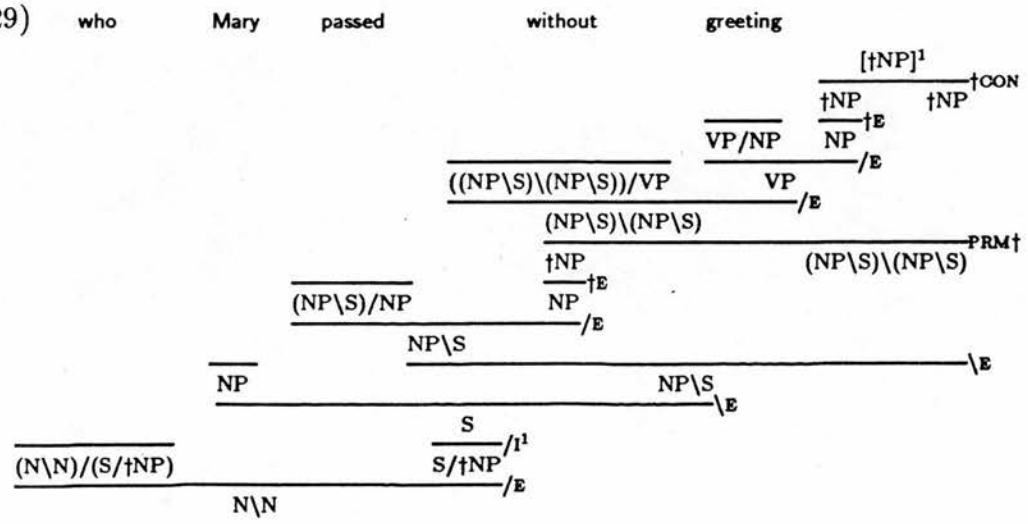
(4.28) $\dfrac{\overset{\vdots}{\dagger X}}{X}\dagger E$ $\qquad$ $\dfrac{\overset{\vdots}{X}}{\dagger X}\dagger I$ $\qquad$ Condition on †I: every path to an undischarged assumption in the proof of X leads to an independent subproof of a †-type

$$\dfrac{\overset{\vdots}{\dagger X}\quad\overset{\vdots}{Y}}{Y\quad\dagger X}\dagger PRM \qquad \dfrac{\overset{\vdots}{Y}\quad\overset{\vdots}{\dagger X}}{\dagger X\quad Y}PRM\dagger \qquad \dfrac{\overset{\vdots}{\dagger X}}{\dagger X\quad\dagger X}\dagger CON$$

The same remarks apply to these rules as to the rules for Δ, and again † is assumed to have identity semantics.

This gives us a simplified treatment of multiple extraction. If the type of **who** is refined to (N\N)/(S/†NP), it will now license extraction not just from any position in the body of a relative clause, but from *any number of* positions (greater than or equal to one). Since † has all the inference rules of Δ, the derivations in (4.26) will still go through with the new type. In addition, the †CON inference rule allows the derivation given in (4.29), and the corresponding term in (4.30):

(4.29)



(4.30) **who** (λx[((**without** (**greeting** x)) (**passed** x)) **Mary**])

As a linguistic treatment of parasitic gaps, this overgenerates in many ways (for instance it does not capture their asymmetry), but it demonstrates in principle how the type of a relative pronoun can be adapted to allow multiple extractions.

Finally, optionality is achieved by means of the operator ‡. ‡X is the type of an item of type X which may be freely permuted and iterated and omitted; thus zero or more occurrences of items of type X in any position may be derived from an item of type †X. ‡ has the all the inference rules of †, plus an analogue of the weakening rule which we shall call ‡WKN:

(4.31)
$$\frac{\overset{\vdots}{\ddagger X}}{X}\ddagger\text{E} \qquad \frac{\overset{\vdots}{X}}{\ddagger X}\ddagger\text{I}$$

Condition on ‡I: every path to an undischarged assumption in the proof of X leads to an independent subproof of a ‡-type

$$\frac{\overset{\vdots}{\ddagger X}\quad\overset{\vdots}{Y}}{Y\quad\ddagger X}\ddagger\text{PRM} \qquad \frac{\overset{\vdots}{Y}\quad\overset{\vdots}{\ddagger X}}{\ddagger X\quad Y}\text{PRM}\ddagger \qquad \frac{\overset{\vdots}{\ddagger X}}{\ddagger X\quad\ddagger X}\ddagger\text{CON} \qquad \frac{\overset{\vdots}{\ddagger X}\quad\overset{\vdots}{Y}}{Y}\ddagger\text{WKN} \qquad \frac{\overset{\vdots}{Y}\quad\overset{\vdots}{\ddagger X}}{Y}\text{WKN}\ddagger$$

Cases of extraction which allow all of commutation, iteration and optionality are not easy to find, but the following may serve as an example:

(4.32) a. The lecture was too long for Mary to follow.

b. The lecture was too long for Mary to follow easily.

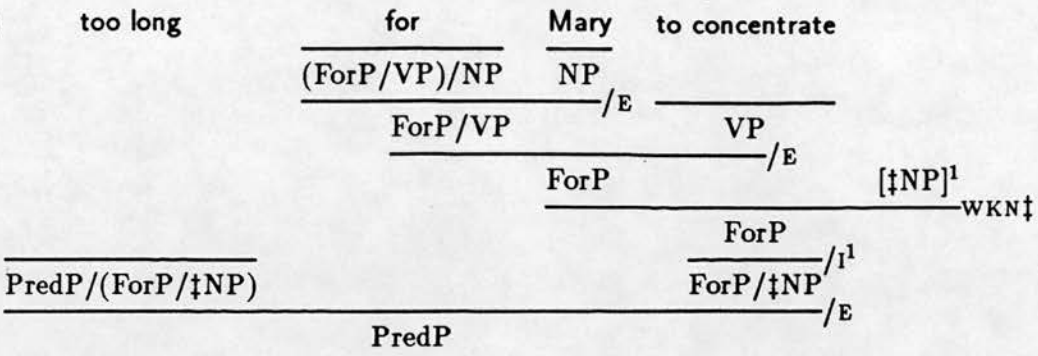
c. The lecture was too long for Mary to follow without transcribing.

d. The lecture was too long for Mary to concentrate.

In (a) too long licenses simple peripheral extraction, in (b) simple non-peripheral extraction, in (c) multiple extraction and in (d) no extraction. We may thus assume that too long licenses extraction from any number of positions in the embedded clause greater than or equal to zero. In the linguistic analysis, I shall assume for simplicity that to-infinitives are single lexical items of type VP, that for-to clauses have a special atomic type ForP (so that for has the type (ForP/VP)/NP), and that predicate phrases like too long to follow have a special atomic type PredP. Too long will therefore receive the type PredP/(ForP/‡NP).

Since ‡ has all the inference rules of †, (4.32a–c) may be generated in a similar way to the earlier examples. The WKN‡ rule allows (4.32d) to be derived as in (4.33), giving the term in (4.34):

(4.33)



(4.34) **too-long** (λx[**for (to-concentrate Mary)**])

The cumulative system is reasonably logically 'clean', in that its three operators define three distinct levels, analogous to **LP**, **LPC** and **LPCW** (and thus to a different subset of the lambda-calculus). In addition, the existence of introduction rules makes the system complete in a technical sense, since the elimination and introduction rules for each universal modality are identical to those for the universal modality in the modal logic S4 (see Dosen 1990). However, the expressive power of the cumulative system is severely limited, since firstly the operational rules are not independent, and secondly the universal modalities are only capable of dealing with commutable, iterable and optional *extractions* (see below for an explanation of this). For this reason it is usually preferable for linguistic purposes to use the orthogonal system, which I shall now discuss.

4.2.3 Orthogonal Structural Modalities

For each of commutation, iteration and optionality, the orthogonal system defines both universal and existential structural modalities. (For commutation it turns out to be more convenient to have two directional versions, though a single modality as with the cumulative system would also be possible.) For each universal modality there is an elimination rule and a *single* operational rule (analogous to a structural rule), but no introduction rule; symmetrically, for each existential modality there is an introduction rule and a single operational rule (analogous to an *inverted* version of a structural rule), but no elimina-

tion rule. The system is called 'orthogonal' because the operational rules for the various modalities are independent. The orthogonal system has greater expressive power than the cumulative system, but is less well motivated from a logical point of view; in particular it is not straightforward to assign meaning representations. (For this reason they will be omitted from the derivations below.)

The universal commutation modalities are notated as $\triangleright$ (written as a prefix operator) and $\triangleleft$ (written as a postfix operator). $\triangleright X$ is the type of an item of type X which may be freely permuted to the right, and $X\triangleleft$ is the type of an item of type X which may be freely permuted to to the left. Thus $\triangleright$ and $\triangleleft$ each have the following inference rules: a directional analogue of the permutation rule, and an elimination rule.

(4.35) a. $\dfrac{\vdots \quad \vdots}{\dfrac{\triangleright X \quad Y}{Y \quad \triangleright X}\,\triangleright\text{PRM}}$ b. $\dfrac{\vdots \quad \vdots}{\dfrac{Y \quad X\triangleleft}{X\triangleleft \quad Y}\,\triangleleft\text{PRM}}$

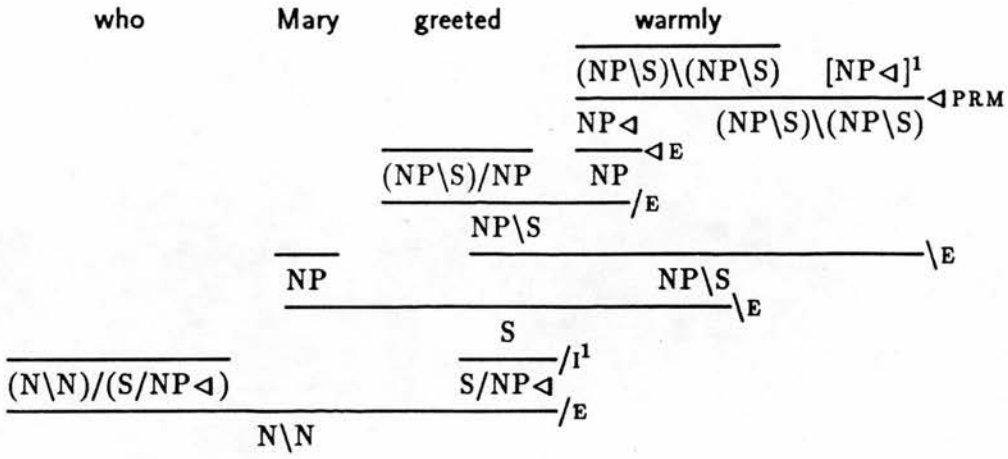
(4.36) a. $\dfrac{\vdots}{\dfrac{\triangleright X}{X}\,\triangleright\text{E}}$ b. $\dfrac{\vdots}{\dfrac{X\triangleleft}{X}\,\triangleleft\text{E}}$

So, for instance, the derivations in (4.26) can be achieved in this system using $\triangleleft$ instead of $\triangle$; the type assigned to who will now be $(N\backslash N)/(S/NP\triangleleft)$:

(4.37) a.

$$\dfrac{\dfrac{\text{who}}{(N\backslash N)/(S/NP\triangleleft)} \quad \dfrac{\dfrac{\text{Mary}}{NP} \quad \dfrac{\dfrac{\text{greeted}}{(NP\backslash S)/NP} \quad \dfrac{[NP\triangleleft]^1}{NP}\,\triangleleft\text{E}}{NP\backslash S}\,/\text{E}}{S}\,\backslash\text{E}}{\dfrac{S}{S/NP\triangleleft}\,/\text{I}^1}\,/\text{E}}{N\backslash N}$$

b.



Dual to these modalities are two existential commutation modalities, $\rangle$ (written postfix) and $\langle$ (written prefix). The type $X\rangle$ (resp. $\langle X$) is assigned to an item of type X either in its canonical position or somewhere to the right (resp. left) of its canonical position. This is achieved by means of the rules $\rangle$I and $\langle$I, which allow the appropriate modality to be added, and the operational rules $\rangle$EXC and $\langle$EXC ('exchange'), which allow types bearing the modality to be moved to any position on the given side. Note that the $\rangle$I and $\langle$I rules are inversions of the $\triangleright$E and $\triangleleft$E rules, and the $\rangle$EXC and $\langle$EXC rules are inversions of the $\triangleright$PRM and $\triangleleft$PRM rules. (Since permutation is symmetrical, the EXC rules are in fact structurally identical to the PRM rules, but this is a slightly misleading coincidence.)

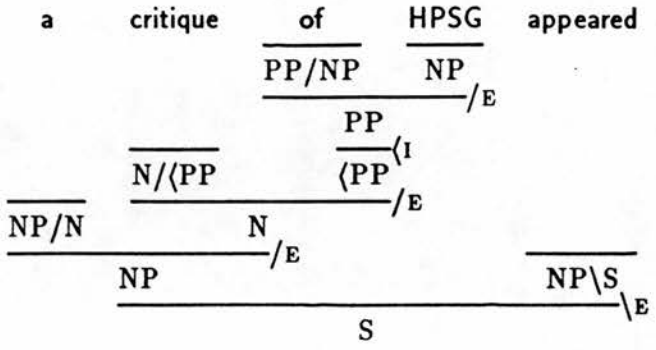
(4.38) a. $\dfrac{\begin{array}{c}\vdots\\ X\end{array}}{X\rangle}\rangle$I b. $\dfrac{\begin{array}{c}\vdots\\ X\end{array}}{\langle X}\langle$I

(4.39) a. $\dfrac{\begin{array}{cc}\vdots & \vdots\\ X\rangle & Y\end{array}}{\begin{array}{cc}Y & X\rangle\end{array}}\rangle$EXC b. $\dfrac{\begin{array}{cc}\vdots & \vdots\\ Y & \langle X\end{array}}{\begin{array}{cc}\langle X & Y\end{array}}\langle$EXC

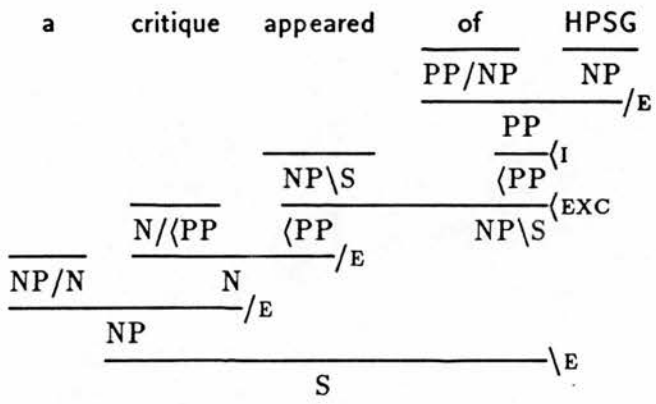
By way of example a type $X/\langle Y$ represents elements which are functors over Ys *somewhere* to the right, but not necessarily immediately to the right. In this way we can define

the 'long-distance' functors $X//Y$ and $Y\backslash\backslash X$ of Bach (1984).[7] This means that we can deal with examples like (4.6) by giving critique the type N/⟨PP:

(4.40) a.

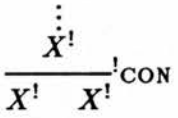


b.

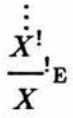


The universal iteration modality is notated as $^!$. $X^!$ is the type of an item from which one or more occurrences of the type X may be derived. Thus $^!$ has an elimination rule and an analogue of the contraction rule, but no other rules:

(4.41) a.



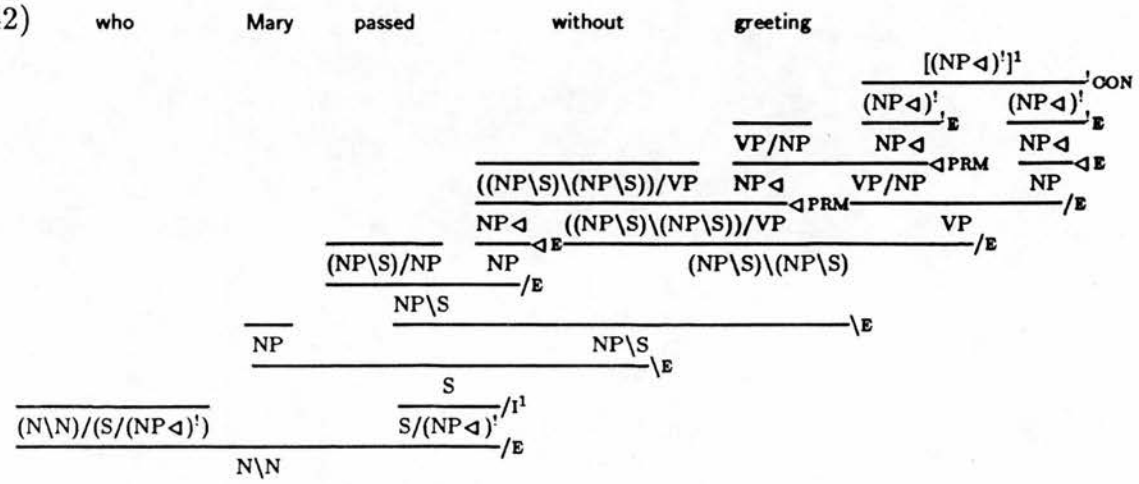
b.



[7]Typically such specifications would need to be made with reference also to the domains within which the word order variation is allowed, e.g. by using the modal treatment of boundedness introduced in Morrill (1989).

In order to perform multiple extraction within this framework, then, both permutation and iteration operators are needed. For instance, to generate (4.4), who must receive the type $(N\backslash N)/(S/(NP\triangleleft)^!)$:

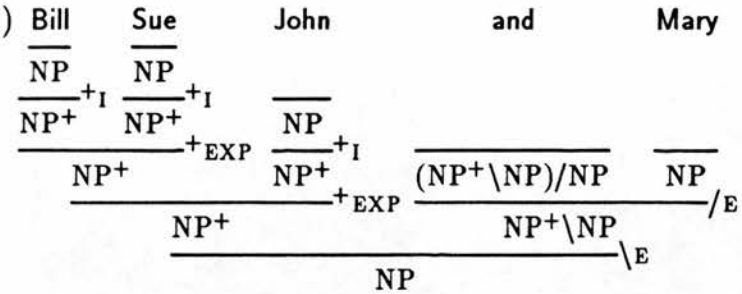
(4.42)



Dual to $^!$ is the existential iteration modality $^+$ (the usual 'Kleene plus' symbol). The type X^+ is assigned to an item formed by combining one or more items of type X. This is achieved by means of the $^+$I rule, which introduces the operator, and the $^+$EXP ('expansion') rule, which combines identical types bearing it. Again, note that the $^+$I rule is an inversion of the $^!$E rule, and the $^+$EXP rule is an inversion of the $^!$CON rule.

(4.43) a. $\dfrac{\vdots\ X}{X^+}\,{}^+\mathrm{I}$ b. $\dfrac{\vdots\ X^+ \quad \vdots\ X^+}{X^+}\,{}^+\mathrm{EXP}$

As an example, iterated coordination of NPs may be treated by giving coordinators the type $(NP^+\backslash NP)/NP$, which will allow one or more NPs to the left of the coordinator:

(4.44)


This approach may be compared to the iterating coordination schema of GPSG (Gazdar, Klein, Pullum and Sag 1985, pp. 169ff.). Compare also Wood's (1988) categorial treatment of coordination in which the coordinator receives a type notated as $X\mathrm{I}(X^{+} * X)$, where I is a so-called 'infix slash'; the intended meaning is that X may be any category, the conjuncts may be multiple, and the conjunction is infixed before the final conjunct.

Finally, the universal optionality modality is notated as $^{\|}$; $X^{\|}$ is the type of an item from which zero or one occurrences of the type X may be derived. $^{\|}$ has an elimination rule and an operational rule of weakening, but in keeping with the presentation in (4.21) I shall present two directional 'absorption' versions of the weakening rule, $^{\|}\mathrm{WKN_l}$ and $^{\|}\mathrm{WKN_r}$:

(4.45) a. $\dfrac{\vdots \quad \vdots}{\dfrac{X^{\|} \quad Y}{Y}\,\mathrm{WKN_l}}$ b. $\dfrac{\vdots \quad \vdots}{\dfrac{Y \quad X^{\|}}{Y}\,\mathrm{WKN_r}}$ c. $\dfrac{\vdots}{\dfrac{X^{\|}}{X}\,{}^{\|}\mathrm{E}}$

Note that this operator allows 'zero or one' gaps, not just 'zero or more', and so could be used in the type of a lexical item whose argument optionally contained a gap (perhaps as in languages with resumptive pronouns or optional clitic doubling). The modality can of course be combined with the universal commutation and iteration modalities to produce an operator with the power of ‡.

Dual to $^{\|}$ is the existential optionality modality $^{?}$; $X^{?}$ indicates zero or one items of type X. The $^{?}\mathrm{I}$ rule introduces the operator on a non-modal type as with other existential modalities; the $^{?}\mathrm{STN}$ ('strengthening') rule creates a modal type from nothing. As with the other universal-existential pairs, the $^{?}\mathrm{I}$ rule is an inversion of the $^{\|}\mathrm{E}$ rule, and the $^{?}\mathrm{STN}$ rule is an inversion of the $^{\|}\mathrm{WKN}$ rule.

(4.46) a. $\dfrac{\vdots\ X}{X^?}{}^{?_I}$ b. $\dfrac{}{X^?}{}^{?_{STN}}$

By way of example, the optionality of the sentential complement of **thought** may be characterized by assignment to $N/SP^?$:

(4.47) **the** **thought** **that Mary was dead**

$$\dfrac{\dfrac{}{NP/N}\quad\dfrac{\dfrac{}{N/SP^?}\quad\dfrac{SP}{SP^?}{}^{?_I}}{N}/_E}{NP}/_E$$

(4.48) **the** **thought**

$$\dfrac{\dfrac{}{NP/N}\quad\dfrac{\dfrac{}{N/SP^?}\quad\dfrac{}{SP^?}{}^{?_{STN}}}{N}/_E}{NP}/_E$$

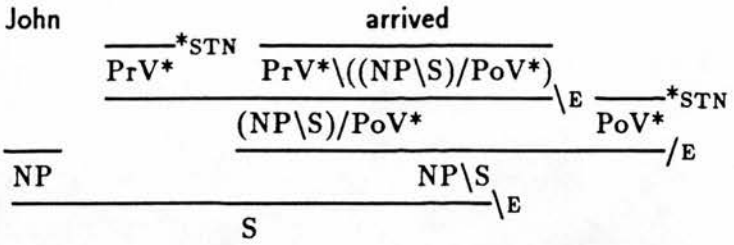
The existential iteration and optionality modalities can be combined to produce a 'zero or more' operator. Since this is the same as 'Kleene star', I shall notate it as *, where X^* is equivalent to $(X^+)^?$. * may be viewed as having the inference rules of both $^+$ and $^?$:

(4.49) a. $\dfrac{\vdots\ X}{X^*}{}^{*_I}$ b. $\dfrac{\vdots\ X^* \quad \vdots\ X^*}{X^*}{}^{*_{EXP}}$ c. $\dfrac{}{X^*}{}^{*_{STN}}$

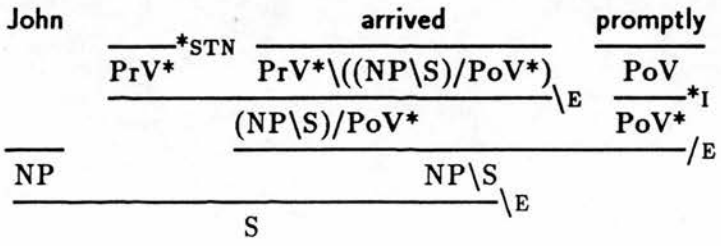
The use of this operator becomes appropriate in the light of the discussion of modifiers in 3.3.3, which requires us to view modifiers as arguments rather than functors to reflect dependency relations. There it was shown that under the normal categorial type system an infinite number of lexical type-assignments would be needed for each modifiable element to reflect the fact that modifiers are iterable and optional. However, the * operator allows us to give the lexical types (PrN*\N)/PoN* to nouns and PrV*\((NP\S)/PoV*) to intransitive verbs (cf. HPSG (Pollard and Sag 1987)), allowing derivations such as the

following:

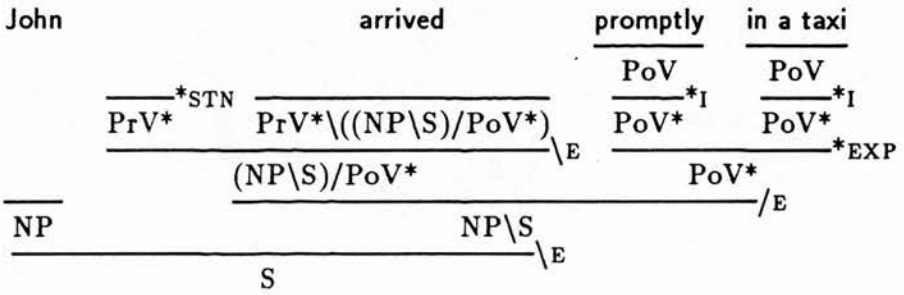
(4.50) a.



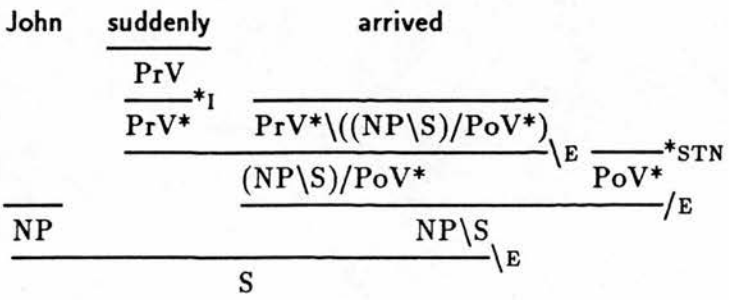
b.



c.



d.



In actual fact type-assignments to some modifiable elements may have to be more complex than this. For instance, if we take sentential postmodifiers and premodifiers into account as well as verbal ones (e.g. today John arrived, John arrived today), then the type of intransitive verbs will be of the form

$$PrV^*\backslash((NP\backslash((PrS^*\backslash S)/PoS^*))/PoV^*)\ ,$$

since modifiers for the resulting sentence must be taken into account. The types of all verbs will have to be refined similarly, e.g. transitive verbs will have the type

$$(PrV^*\backslash((NP\backslash((PrS^*\backslash S)/PoS^*))/PoV^*))/NP\ .$$

Such complexity in type-assignments is of course undesirable, but as things stand there is no straightforward way to avoid it because of the automatic correspondence between heads and functors, and because linear order is directly encoded into the type.

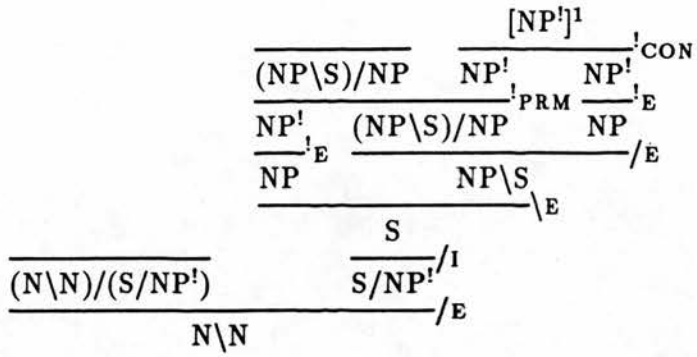
As mentioned above, neither of the two systems of operators is designed to address particular linguistic phenomena, which means that the sample type-assignments above are of necessity rather crude when it comes to dealing with individual constructions. For instance, the cumulative modality system, with the type $(N\backslash N)/(S/\triangle NP)$ for **who(m)**, would incorrectly generate the following construction (and similarly for the orthogonal system):

(4.51) ***who(m) I think that arrived**

$$\dfrac{\dfrac{}{(N\backslash N)/(S/\triangle NP)} \quad \dfrac{\dfrac{\dfrac{}{NP} \quad \dfrac{\dfrac{}{(NP\backslash S)/SP} \quad \dfrac{\dfrac{}{SP/S} \quad \dfrac{\dfrac{}{NP\backslash S}\ \dfrac{\dfrac{}{NP\backslash S} \quad \dfrac{\dfrac{}{\triangle NP} \quad \dfrac{}{NP\backslash S}}{NP}\triangle E}{S}\backslash E}{SP}/E}{S}/E}{NP\backslash S}/E}{S}/I}{N\backslash N}/E \qquad \dfrac{}{NP\backslash S} \quad \dfrac{}{[\triangle NP]^1}\ PRM\triangle$$

Similarly, the type $(N\backslash N)/(S/NP^!)$ for **who(m)**, designed to allow multiple extraction, allows extraction of any number of NPs in any positions and hence gives rise to overgenerations such as the following:

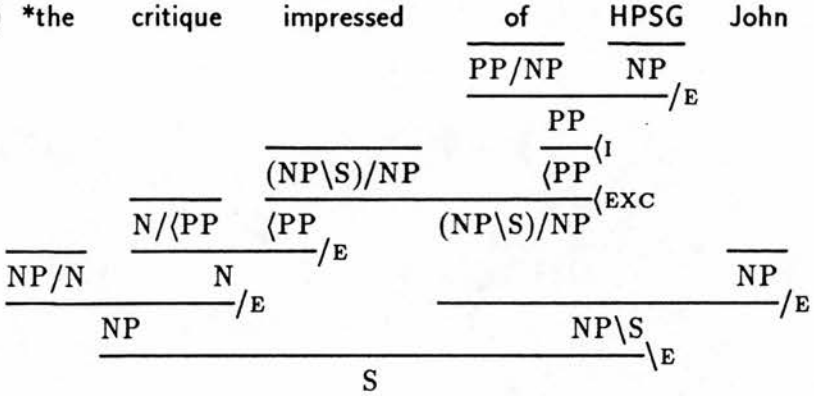
(4.52) *who(m) loves



However, in each of the above cases appeal could be made to case features on the extracted NP, since an object relative pronoun has been used to perform a subject extraction.

The applications presented for the existential modalities can also cause overgeneration. For example, the type N/⟨PP given for critique in (4.40) allows its complement to appear *anywhere* to its right, which would generate constructions such as the following:

(4.53) *the critique impressed of HPSG John



In general, therefore, all the above operators should be seen as tools for use in the description of constructions, rather than linguistic devices in their own right. To provide a more accurate characterization of these phenomena would require an account of boundedness, which is outside the scope of this thesis. See Hepple (1990) for extensive discussion of the interaction of commutation and boundary phenomena.

Chapter 5

Synthesis, Comparisons and Conclusions

5.1 Dependency and the Categorial Hierarchy

In this section I shall consider how the various categorial frameworks discussed so far may be viewed as part of a more general network of categorial systems. With this as background I shall then suggest that modalities analogous to structural modalities may be introduced into the Lambek calculus so that dependency may be dealt with in a controlled fashion.

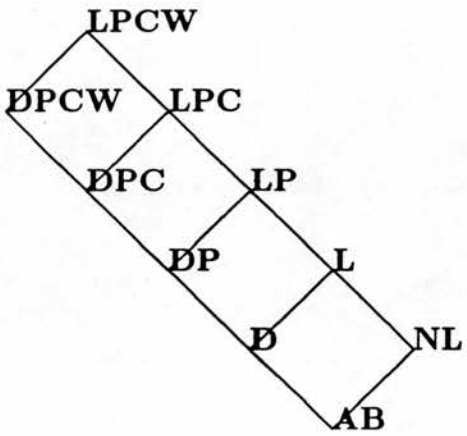
5.1.1 A Network of Categorial Systems

In chapter 3 we discussed the relation between **AB**, **D** and **L**, and in chapter 4 we looked at the 'categorial hierarchy' containing **L**, **LP**, **LPC** and **LPCW**. For convenience I shall repeat the interpretation of directional types in each of the frameworks **AB**, **L** and **D** in (5.1):

(5.1) a. In **AB**: an expression is of type X/Y (resp. $Y\backslash X$) iff, whenever it is combined (non-associatively) with a following (resp. preceding) non-head of type Y, the result is of type X.

b. In **L**: an expression is of type X/Y (resp. $Y\backslash X$) iff, whenever it is concatenated (associatively) with a following (resp. preceding) expression of type Y, the result is of type X.

Figure 5–1: The categorial network



c. In **D**: an expression is of type X/Y (resp. $Y\backslash X$) iff, whenever it is concatenated (associatively) with a following (resp. preceding) non-head of type Y, the result is of type X.

In fact we may view all of these systems as part of a more general network of categorial systems, as illustrated in Figure 5–1. Several of these calculi have not yet been discussed, and will be explained below. **NL** is the non-associative calculus of Lambek (1961); **DP**, **DPC** and **DPCW** are new calculi constructed by analogy with **LP**, **LPC** and **LPCW**. The calculi are arranged in a lattice, where X is above Y in the lattice iff X is more powerful than Y (i.e. all the theorems of X are also theorems of Y). Essentially one proceeds 'north-west' in the diagram by adding structural rules, and 'north-east' by removing the dependency constraint.

NL is the calculus that arises when expressions are deemed to have a binary-branching structure without any notion of head. Combination of expressions is non-associative, so that structures cannot be rebracketed, but neither daughter in a local tree is distinguished in any way. Essentially, **NL** contains all **AB**-valid inferences plus those **L**-valid inferences that do not require associativity. For example, the inferences in (5.2a) are valid in **NL**, but those in (5.2b) are not:

(5.2) a. $X/Y \;\; Y \Rightarrow X$

$Y \;\; Y\backslash X \Rightarrow X$

$X/Y \;\; Y/Z \;\; Z \Rightarrow X$

$Y \;\; (Y\backslash X)/Z \;\; Z \Rightarrow X$

$X \Rightarrow Y/(X\backslash Y)$

$X/Z \Rightarrow (Y/(X\backslash Y))/Z$

$Z/(Y/(X\backslash Y)) \Rightarrow Z/X$

b. $X/Y \;\; Y/Z \not\Rightarrow X/Z$

$Y \;\; (Y\backslash X)/Z \not\Rightarrow X/Z$

$(Y\backslash X)/Z \not\Rightarrow Y\backslash(X/Z)$

The following interpretation of types can be given for **NL** (cf. (5.1)):

(5.3) An expression is of type X/Y (resp. $Y\backslash X$) iff, whenever it is combined (non-associatively) with a following (resp. preceding) expression of type Y, the result is of type X.

NL is hard to formulate in the proof figure notation, and I shall not attempt to do so here. For an axiomatic formulation see Lambek (1961).

DP, **DPC** and **DPCW** are new calculi obtained from **D** in the obvious fashion by cumulatively adding structural rules of permutation, contraction and weakening; however there is a slight complication with their formulation in the proof figure notation since types can be copied by the structural rules. Thus it is no longer sufficient to require merely that the type conditionalized by an $\rightarrow$I inference does not itself occur as the functor in an $\rightarrow$E inference; we must also require that no copy of the conditionalized type occurs as a functor.

For example, consider the following derivation, which is valid in **LP** but not in **DP**:

$$\dfrac{\dfrac{\dfrac{\dfrac{[Z\rightarrow Y]^1 \quad Z}{Z \quad Z\rightarrow Y}\,\text{PRM}}{Y}\rightarrow\text{E} \qquad Y\rightarrow X}{X}\rightarrow\text{E}}{(Z\rightarrow Y)\rightarrow X}\rightarrow\text{I}^1$$

This is invalid in **DP** because the copy of the conditionalized type $Z\rightarrow Y$ created by the permutation rule acts as the functor in an $\rightarrow$E inference.

The addition of structural rules to **D** mirrors some of the extensions that have been proposed to classical dependency grammar to cover a wider range of linguistic phenomena. Essentially, adding permutation corresponds to removing the adjacency requirement, adding contraction corresponds to removing the single-headedness requirement, and adding weakening corresponds to removing the connectedness requirement. In such a way we may build up a 'dependency hierarchy' in the manner of Fraser (1989). As with the grammars in the categorial hierarchy, it is clear that adding freely applying structural rules to **D** would be linguistically undesirable. However, structural modalities can be added to **D** in an exactly analogous fashion to **L**, allowing controlled degrees of non-adjacency, multi-headedness and disconnectedness. I shall not give separate illustrations of this here since all the examples in chapter 4 in fact obey the dependency constraint.

5.1.2 Dependency Modalities

Given the above discussion, it is natural to ask whether it is possible to encode dependency by means of modalities within **L** rather than by means of a special calculus. It turns out that this can be achieved in a natural fashion by means of a single modality. This modality is unusual in that its sole inference rule is an introduction rule. I shall use the notation $\Diamond$ for it, so that the rule reads as follows:

(5.4) $$\frac{X}{\Diamond X}\Diamond I$$

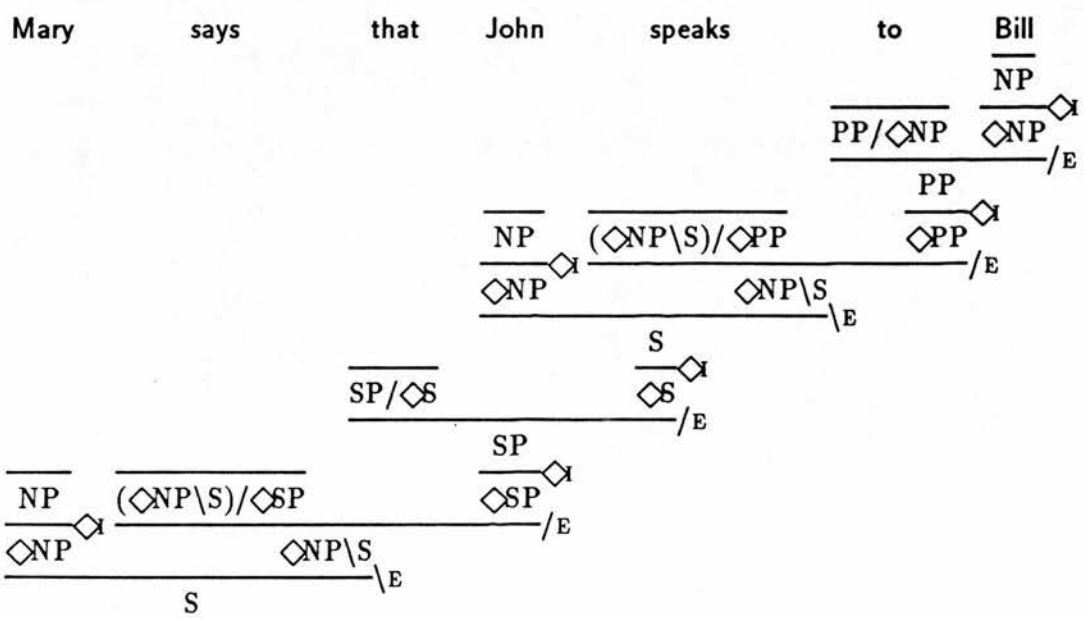
The intuition behind $\Diamond$ is that it always occurs on any type that is deemed to be a dependent in the particular analysis. Up until now we have assumed that all arguments in CG are dependents, so a straightforward mapping from types in **D** to types in **L**+$\Diamond$ would produce a lexicon such as the following (the usual **L**/**D** types are included for comparison):

(5.5)

Word	Syntactic type in **L**+◇	Syntactic type in **L/D**
walks	◇NP\S	NP\S
runs	◇NP\S	NP\S
likes	(◇NP\S)/◇NP	(NP\S)/NP
speaks	(◇NP\S)/◇PP	(NP\S)/PP
must	(◇NP\S)/◇VP	(NP\S)/VP
says	(◇NP\S)/◇SP	(NP\S)/SP
John	NP	NP
Mary	NP	NP
to	PP/◇NP	PP/NP
for	PP/◇NP	PP/NP
walk	VP	VP
like	VP/◇NP	VP/NP
that	SP/◇S	SP/S

The fact that ◇ can be freely introduced means that an expression may be analysed as a dependent whenever it is needed by a head. For instance:

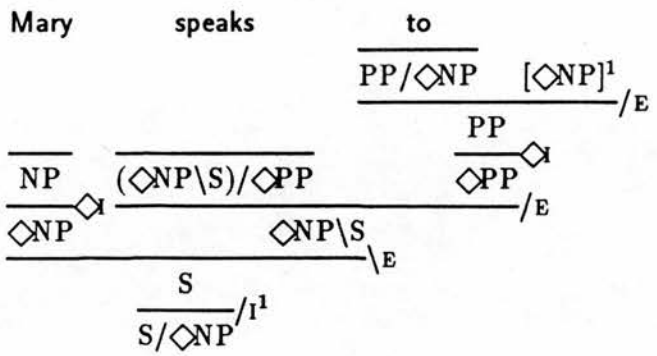
(5.6)



Furthermore, an expression that is missing a dependent or dependents can always be given a type where the type(s) of the missing dependent(s) are marked with ◇, since appropriate

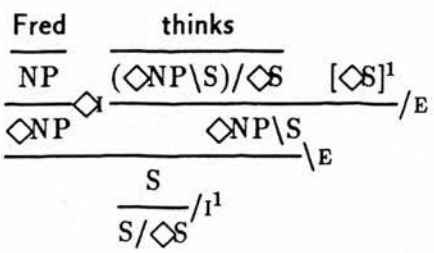
◇types can be assumed and conditionalized in the derivation. For example:

(5.7) **Mary** **speaks** **to**

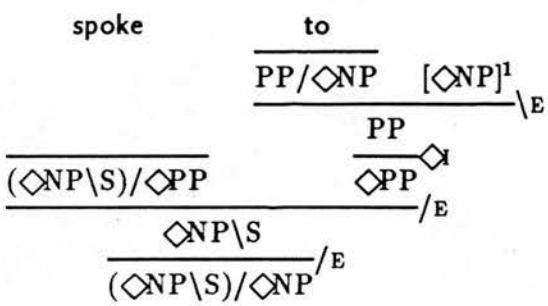


It follows that in this system dependency constituents will always receive types of the form $\Diamond Y_n \rightarrow (\ldots \rightarrow (\Diamond Y_1 \rightarrow X))$ (where $n \geq 0$). We can illustrate this by giving ◇-based analogues of the derivations in 3.3.1. Expressions such as **Fred thinks** and **spoke to**, which are dependency constituents, can receive types of this form, whereas expressions such as **thinks Bill** and **Mary vodka** cannot:

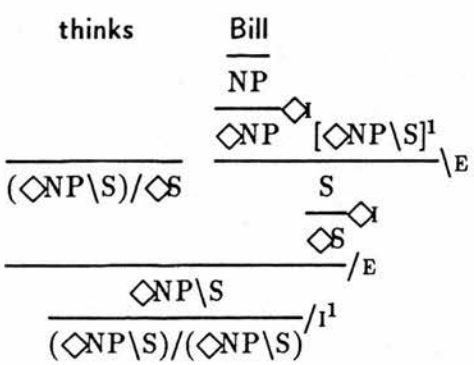
(5.8) **Fred** **thinks**



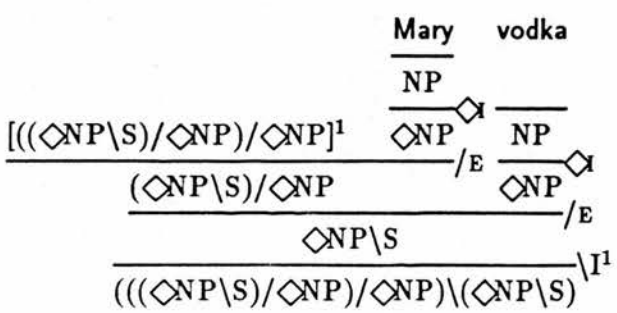
spoke **to**



(5.9) **thinks** **Bill**



Mary **vodka**



In order to be a dependency constituent, **thinks Bill** would have to be able to receive the type $(\Diamond NP\backslash S)/\Diamond(\Diamond NP\backslash S)$, and **Mary vodka** the type $\Diamond(((\Diamond NP\backslash S)/\Diamond NP)/\Diamond NP)\backslash(\Diamond NP\backslash S)$. Neither of these is possible, because the derivation would require assuming the type

$\Diamond(\Diamond NP\backslash S)$ in the first case and $\Diamond(((\Diamond NP\backslash S)/\Diamond NP)/\Diamond NP)$ in the second case. But since $\Diamond$ has no elimination rule, there is no way of removing the operator so that each of these functions can be applied to the appropriate argument. In essence, this enforces the 'no functor abstraction' constraint automatically.

In addition, the use of an explicit dependency modality gives us an extra degree of freedom, in that we can regard the arguments in some lexical entries as being non-dependent, i.e. of type $Y \rightarrow X$ rather than $\Diamond Y \rightarrow X$. This of course breaks with the strict dependency interpretation of CG assumed for **D**, but it might be useful in cases where it is unclear whether to regard functors or arguments as heads, as with determiners and nouns. Suppose we give determiners such as **the** and **a** the lexical type NP/N in this system (rather than NP/$\Diamond$N); note that this does not match the schema for types of dependency constituents given earlier. Then we can no longer derive strings such as **John saw the** as dependency constituents, since the conditionalized type in the derivation below is not a dependent:

(5.10) **John** **saw** **the**

$$\dfrac{\dfrac{\dfrac{}{NP}\Diamond I}{\Diamond NP} \quad \dfrac{}{(\Diamond NP\backslash S)/\Diamond NP} \quad \dfrac{\dfrac{\dfrac{}{NP/N} \quad \dfrac{}{[N]^1}}{NP}/E}{\Diamond NP}\Diamond I}{\dfrac{\Diamond NP\backslash S}{S}\backslash E}$$

$$\dfrac{S}{S/N}/I^1$$

Given the near-unacceptability of constructions such as ?*John saw the, and Bill saw a, woman, it seems at least plausible that John saw the should not have the status of a dependency constituent. In contrast, determiners such as two and many might retain the type NP/$\Diamond$N, since John saw two arguably has the properties of a dependency constituent (since e.g. John saw two, and Bill saw many, women appears to be acceptable).

Finally, note that there may be a correlation in the distribution of dependency modalities and structural modalities in a system that combines the two. This is because, as noted in 4.2.1, structural modalities only appear on arguments (or on arguments of arguments),

which suggests that structural modalities might conceivably be viewed as a special type of dependency modality. I shall not pursue this further here.

5.2 Comparisons with Combinatory Categorial Grammar

Many of the linguistic phenomena covered in the last two chapters have been dealt with in alternative fashions by other categorial frameworks. In this section I shall look in detail at one such framework, developed by Steedman (1987, 1990) and known as Combinatory Categorial Grammar (CCG), and compare its treatments of flexible constituency and non-sequentiality with the ones proposed here.

5.2.1 Combinatory Categorial Grammar

CCG resembles the categorial frameworks we have already considered in that it consists of a lexicon plus a system for combining types. However, it differs from them in that the system for combining types is not a general logical inference procedure but a specified set of rules. The rules of CCG are inspired by *combinatory logic* (Curry and Feys 1958), which is an alternative system to the lambda-calculus for defining functional application and functional abstraction. It does not make use of bound variables but instead uses operators called *combinators* to define functions in terms of other functions.

The simplest rules in CCG, corresponding to the /E and \E inferences in the proof figure formulation of **AB** or **L**, are the rules of *functional application.* They state that an expression of a functor type may be combined with an expression of an argument type on the appropriate side, and that the meaning of the result is the meaning of the function applied to that of the argument. This can be notated by means of the notation in (5.11), which uses colons to separate meanings from types and an arrow ($\Rightarrow$) to denote derivation. The notation means that a sequence of expressions with the meanings and types to the left

of the arrow can be combined (in that order) to produce an expression with the meaning and type to the right of the arrow.

(5.11) a. Forward application (indexed $\rangle$)

$\phi : X/Y \quad \alpha : Y \Rightarrow \phi\alpha : X$

b. Backward application (indexed $\langle$)

$\alpha : Y \quad \phi : Y\backslash X \Rightarrow \phi\alpha : X$

In addition to the application rules, there are a number of *combinatory rules* which allow more complex type-combining operations. Semantically, these rules correspond to certain primitive combinators; syntactically, they are constrained by three principles which Steedman claims to be universal.

Steedman formulates his semantics in terms of the three combinators **B**, **T** (or $\mathbf{C}_*$) and **S**. These can be defined by means of the equivalences in (5.12):

(5.12) a. $\mathbf{B}fgx = f(gx)$

b. $\mathbf{T}xf = fx$

c. $\mathbf{S}fgx = fx(gx)$

However, it will be more perspicuous for present purposes to regard **B** and **S** as two-place operators, and **T** as a one-place operator, so that the combinators can be viewed as corresponding to special lambda-expressions as in (5.13):

(5.13) a. $\mathbf{B}fg = \lambda x[f(gx)]$

b. $\mathbf{T}x = \lambda f[fx]$

c. $\mathbf{S}fg = \lambda x[fx(gx)]$

B thus performs the operation of functional composition, and **T** reverses the roles of function and argument (type-raising). **S** ('substitution') is harder to describe straightforwardly, but it can be thought of as taking two one-place functions and forming a new function which identifies their arguments.

The above equivalences automatically put constraints on the relations between the semantic types of terms that can be combined, as follows (where X, Y, Z are arbitrary semantic types):

(5.14) a. $\mathbf{B}f^{Y\rightarrow X}g^{Z\rightarrow Y} = \lambda x^{Z}[f(gx)]$

b. $\mathbf{T}x^{X} = \lambda f^{X\rightarrow Y}[fx]$

c. $\mathbf{S}f^{Z\rightarrow(Y\rightarrow X)}g^{Z\rightarrow Y} = \lambda x^{Z}[fx(gx)]$

Thus **B** combines functions of types $Y\rightarrow X$ and $Z\rightarrow Y$ to form a function of type $Z\rightarrow X$; **T** maps a term of type X to a function of type $(X\rightarrow Y)\rightarrow Y$; **S** combines functions of types $Z\rightarrow(Y\rightarrow X)$ and $Z\rightarrow Y$ to form a function of type $Z\rightarrow X$.

The syntactic types of expressions that can be combined are further constrained by the following three principles, which are claimed to be universal:

(5.15) a. *Adjacency*: Combinatory rules may only apply to entities which are linguistically realized and adjacent.

b. *Directional Consistency*: All syntactic combinatory rules must be consistent with the directionality of the principal function.

c. *Directional Inheritance*: If the category that results from the application of a combinatory rule is a function category, then the slash defining directionality for a given argument in that category will be the same as the one defining directionality for the corresponding argument(s) in the input function(s).

The principle of adjacency is akin to the assumption of 'sequentiality' that was earlier made for **L**, and the principle of directional consistency essentially ensures that the directionality of slashes is respected. (Note that the application rules in (5.11) both conform to these two principles.) Directional inheritance is a rather less obvious assumption, saying in essence that directionality is a property of arguments rather than functions, as we shall see in the examples below.

By placing these constraints on the combinations in (5.14) we obtain a set of possible syntactic combinatory rules. It is important to note that the licensing of a particular

syntactic rule does not entail that that rule will be used by a given language, so that in contrast to the categorial frameworks of the previous chapters there may be language-specific variation in the choice of grammar rules; the use of certain rules may be restricted to particular type instantiations, or even forbidden altogether for a particular language, as we shall see in 5.2.2 below. Ignoring such restrictions for the moment, the above principles license four different versions of the **B** (composition) rule, depending on the directionality of the 'main' and 'subsidiary' functions, as follows (rules are referred to as *harmonic* where the main and subsidiary functions have the same directionality, and *disharmonic* otherwise):

(5.16) a. Forward harmonic composition ($\rangle\mathbf{B}$)
$\phi : X/Y \quad \chi : Y/Z \Rightarrow \mathbf{B}\phi\chi : X/Z$

b. Backward harmonic composition ($\langle\mathbf{B}$)
$\chi : Z\backslash Y \quad \phi : Y\backslash X \Rightarrow \mathbf{B}\phi\chi : Z\backslash X$

c. Forward disharmonic composition ($\rangle\mathbf{B}\times$)
$\phi : X/Y \quad \chi : Z\backslash Y \Rightarrow \mathbf{B}\phi\chi : Z\backslash X$

d. Backward disharmonic composition ($\langle\mathbf{B}\times$)
$\chi : Y/Z \quad \phi : Y\backslash X \Rightarrow \mathbf{B}\phi\chi : X/Z$

Likewise there are four possible versions of the **S** (substitution) rule:

(5.17) a. Forward harmonic substitution ($\rangle\mathbf{S}$)
$\phi : (X/Y)/Z \quad \chi : Y/Z \Rightarrow \mathbf{S}\phi\chi : X/Z$

b. Backward harmonic substitution ($\langle\mathbf{S}$)
$\chi : Z\backslash Y \quad \phi : Z\backslash(Y\backslash X) \Rightarrow \mathbf{S}\phi\chi : Z\backslash X$

c. Forward disharmonic substitution ($\rangle\mathbf{S}\times$)
$\phi : Z\backslash(X/Y) \quad \chi : Z\backslash Y \Rightarrow \mathbf{S}\phi\chi : Z\backslash X$

d. Backward disharmonic substitution ($\langle\mathbf{S}\times$)
$\chi : Y/Z \quad \phi : (Y\backslash X)/Z \Rightarrow \mathbf{S}\phi\chi : X/Z$

Finally, there are arguably four possible versions of the **T** (type-raising) rule, though it is not entirely clear how the principles of Consistency and Inheritance should apply to

type-raising (see Steedman 1991 for discussion of this). I shall list all four below, using the terms 'symmetric' and 'asymmetric' to distinguish the variants:

(5.18) a. Forward symmetric type-raising ($\rangle\mathbf{T}$)
$\alpha : X \Rightarrow \mathbf{T}\alpha : Y/(X\backslash Y)$

b. Backward symmetric type-raising ($\langle\mathbf{T}$)
$\alpha : X \Rightarrow \mathbf{T}\alpha : (Y/X)\backslash Y$

c. Forward asymmetric type-raising ($\rangle\mathbf{T}\times$)
$\alpha : X \Rightarrow \mathbf{T}\alpha : Y/(Y/X)$

d. Backward asymmetric type-raising ($\langle\mathbf{T}\times$)
$\alpha : X \Rightarrow \mathbf{T}\alpha : (X\backslash Y)\backslash Y$

Type-raising was originally introduced as a lexical rule by both Steedman and Dowty, and its presentation as a syntactic operation in CCG is sometimes taken to be a notational convenience. However, for ease of comparison I shall assume here that type-raising is a syntactic operation on a par with composition and substitution.

Note that some of the above rules ($\rangle$, $\langle$, $\rangle\mathbf{B}$, $\langle\mathbf{B}$, $\rangle\mathbf{T}$ and $\langle\mathbf{T}$) are derivable as theorems of **L**, and thus their use will not give rise to any new word orders; we shall refer to these as *order-preserving*. The order-preserving rules form a subsystem which generates some, but by no means all, **L**-valid combinations. For example, CCG does not include either of the two rules below, and indeed cannot derive them by using any other rules:

(5.19) $X/Y \Rightarrow (X/Z)/(Y/Z)$ ('forward division')

(5.20) $(Y\backslash X)/Z \Rightarrow Y\backslash(X/Z)$ ('associativity')

However, both rules are valid in **L** and can be given a combinatory semantics; (5.19) as the 'unary' version of **B**, defined by $\mathbf{B}f = \lambda g\lambda x[f(gx)]$, and (5.20) as Curry and Feys' 'commuting' combinator **C**, defined by $\mathbf{C}f = \lambda y\lambda x[(fx)y]$.

The remaining rules of CCG are derivable as theorems of higher categorial logics: $\rangle\mathbf{B}\times$, $\langle\mathbf{B}\times$, $\rangle\mathbf{T}\times$ and $\langle\mathbf{T}\times$ are derivable in **LP** (and of course all higher logics), and $\rangle\mathbf{S}$, $\langle\mathbf{S}$, $\rangle\mathbf{S}\times$

and $\langle \mathbf{S}\times$ are derivable in **LPC**. The existence of non-order-preserving rules means that it is not straightforwardly possible in CCG to give 'extensional' interpretations of types as it was in **AB**, **L** and **D**, since for example a type of the form X/Y may be given to expressions which do not consist of an X missing a Y from the rightmost position (see Steedman 1991 for discussion of this). Note also that the composition and substitution rules, but not the type-raising rules, are dependency-preserving by the generalized definition given in the previous section, so that $\rangle$, $\langle$, $\rangle\mathbf{B}$ and $\langle\mathbf{B}$ are also derivable in **D**, $\rangle\mathbf{B}\times$ and $\langle\mathbf{B}\times$ in **DP**, and $\rangle\mathbf{S}$, $\langle\mathbf{S}$, $\rangle\mathbf{S}\times$ and $\langle\mathbf{S}\times$ in **DPC**. The consequences of this will become evident in the linguistic discussion below.

In fact Steedman regards the composition rule as a specific case of a 'generalized' composition rule, of which the forward harmonic version is given in (5.21):

(5.21) Generalized forward harmonic composition ($\rangle\mathbf{B}^n$)

$$\phi : X/Y \quad \chi : ((Y/Z_1)/\ldots)/Z_n \Rightarrow \mathbf{B}^n\phi\chi : ((X/Z_1)/\ldots)/Z_n$$

(Steedman in fact regards the n in this rule as being bounded by the highest valency in the lexicon.) The combinator $\mathbf{B}^n$ is recursively defined to be **B** for $n = 1$, and $\mathbf{BBB}^{n-1}$ for $n > 1$. All instances of generalized forward harmonic composition are derivable in **L** (and **D**). For $n = 1$ the above rule is just the ordinary forward harmonic composition rule. For $n = 2$ we obtain the following:

(5.22) Second-order forward harmonic composition ($\rangle\mathbf{B}^2$)

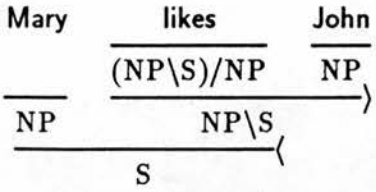
$$\phi : X/Y \quad \chi : (Y/Z_1)/Z_2 \Rightarrow \mathbf{B}^2\phi\chi : (X/Z_1)/Z_2$$

This is the only 'higher-order' version of generalized composition that will be considered from now on.

5.2.2 Linguistic Applications of CCG

Let us now consider how the above system is used in the analysis of English. Clearly all derivations in **AB** have direct analogues in CCG, using the two application rules (whose use is unconstrained). For example, the expression Mary likes John may be derived as a

sentence as follows (cf. 2.8):

(5.23)


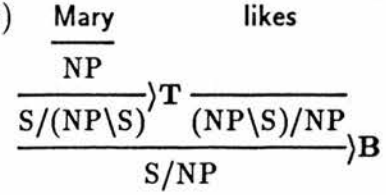
The notation for derivations is based on Steedman's, and resembles the proof figure notation to a large extent. The words are written in sequence at the top of the derivation; each is underlined and its type is written below. When two types are combined, a line is drawn underneath both of them and indexed with the appropriate rule index; the resultant category is then written below the line. The meaning is derived directly from the structure of the proof, just as with the proof figures for **L**, by interpreting the ⟩ and ⟨ rules as functional application (cf. (2.9)):

(5.24) **(likes John) Mary**

To generate constructions within the scope of **L**, CCG uses its order-preserving combinatory rules, constrained to a greater or lesser extent. For example, it is usual to restrict the use of the forward symmetric type-raising rule to subject NPs as follows:

(5.25) Subject type-raising (⟩**T**)
$\alpha : \mathrm{NP} \Rightarrow \mathbf{T}\alpha : \mathrm{S/(NP\backslash S)}$

Derivations that would be performed in **L** (or **D**) by means of the /I rule are performed in CCG using a combination of the ⟩**T** and ⟩**B** rules (for the moment assume ⟩**B** to be unconstrained). For example, we may assign the type S/NP to **Mary likes** by means of the derivation in (5.26), giving the term in (5.27):

(5.26)


(5.27) **B (T Mary) likes**

Using the equivalences in (5.13) and applying β-reduction, we see that this combinatory term is equivalent to the lambda-term that would be derived using **L**:

(5.28) **B (T Mary) likes** $= \lambda x[(\textbf{T Mary})\ (\textbf{likes}\ x)] = \lambda x[(\lambda f[f\ \textbf{Mary}])\ (\textbf{likes}\ x)] \rhd_\beta$ $\lambda x[(\textbf{likes}\ x)\ \textbf{Mary}]$

Repeated use of the $\rangle$**T** and $\rangle$**B** rules in this fashion allows arbitrary right-peripheral extraction:

(5.29) a.

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{Mary}}{\text{NP}}}{\text{S/(NP\S)}}\rangle\textbf{T} \quad \dfrac{\dfrac{\text{speaks}}{\text{(NP\S)/PP}} \quad \dfrac{\text{to}}{\text{PP/NP}}}{\text{(NP\S)/NP}}\rangle\textbf{B}}{\text{S/NP}}\rangle\textbf{B}$$

b.

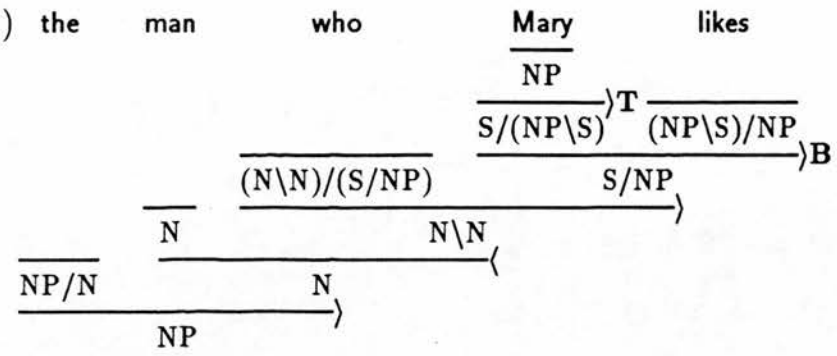
$$\dfrac{\dfrac{\dfrac{\dfrac{\text{Mary}}{\text{NP}}}{\text{S/(NP\S)}}\rangle\textbf{T} \quad \dfrac{\dfrac{\text{says}}{\text{(NP\S)/SP}} \quad \dfrac{\dfrac{\text{that}}{\text{SP/S}} \quad \dfrac{\dfrac{\dfrac{\text{John}}{\text{NP}}}{\text{S/(NP\S)}}\rangle\textbf{T} \quad \dfrac{\text{likes}}{\text{(NP\S)/NP}}}{\text{S/NP}}\rangle\textbf{B}}{\text{SP/NP}}\rangle\textbf{B}}{\text{(NP\S)/NP}}\rangle\textbf{B}}{\text{S/NP}}\rangle\textbf{B}$$

(5.30) a. **B (T Mary) (B speaks to)**

b. **B (T Mary) (B says (B that (B (T Mary) likes)))**

We thus have an account of unbounded extraction similar to the one put forward for **L**. For instance, the man who Mary likes has the derivation in (5.31) and the meaning representation in (5.32):

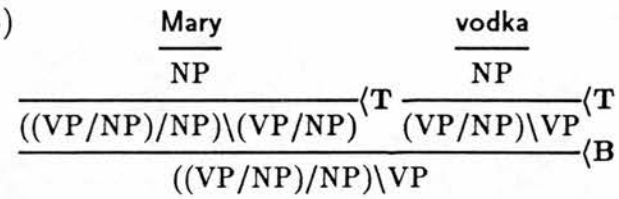
(5.31)



(5.32) **the ((who (B (T Mary) likes)) man)**

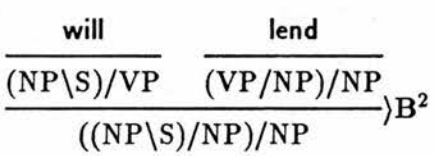
Similarly, versions of the ⟨**B** and ⟨**T** rules may be used where **L** would employ the \I rule. For instance, in order to account for coordinate constructions like give Mary vodka and Sue whisky, Dowty (1988) assigns Mary vodka the type ((VP/NP)/NP)\VP by the following analysis:

(5.33)



The $\rangle\mathbf{B}^2$ rule would be used to generate expressions such as will lend as constituents (Steedman gives the example a car which I will lend, and may sell, to that man as motivation):

(5.34)



The non-order-preserving rules are used for constructions outside the scope of **L**. For instance, Steedman proposes the following restriction on the ⟨**B**× rule (slightly simplified here):

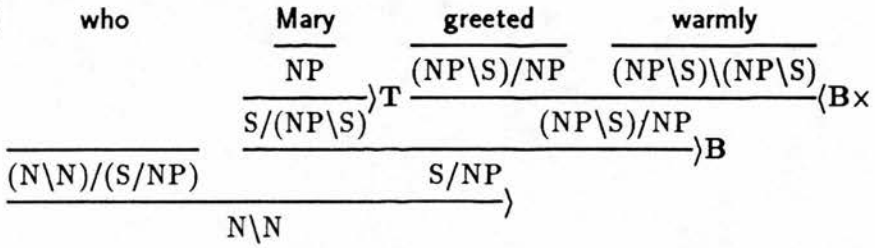
(5.35) English backward disharmonic composition ($\langle\mathbf{B}\times$)

$\chi : Y/Z \quad \phi : Y\backslash X \Rightarrow \mathbf{B}\phi\chi : X/Z$

where Y = NP\S or VP

This version of the rule may be used to assign the type S/NP to an expression consisting of an S missing an NP *internally*, so that the object relative pronoun type (N\N)/(S/NP) will serve for both peripheral and non-peripheral extraction:

(5.36)



The $\langle\mathbf{S}\times$ rule may be restricted in a similar fashion:

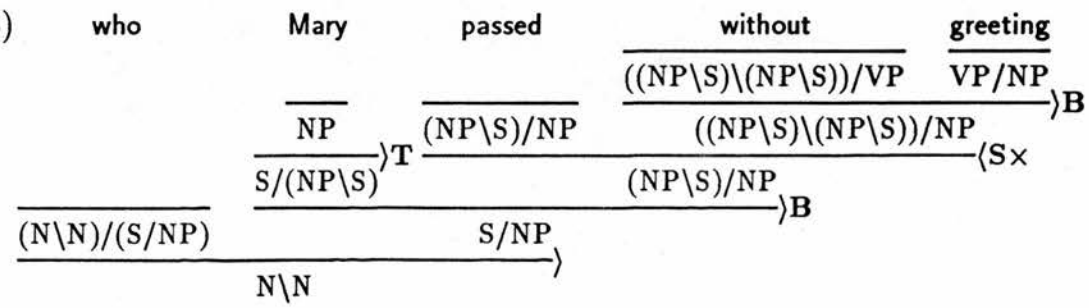
(5.37) English backward disharmonic substitution ($\langle\mathbf{S}\times$)

$\chi : Y/Z \quad \phi : (Y\backslash X)/Z \Rightarrow \mathbf{S}\phi\chi : X/Z$

where Y = NP\S or VP

This rule may be used to assign the type S/NP to an expression consisting of an S missing more than one NP, so that the same object relative pronoun type will also serve for parasitic extraction:

(5.38)



Use of the remaining non-order-preserving rules in English is generally forbidden, although Steedman (1987) suggests using a restricted version of the forward asymmetric

type-raising rule $\rangle\mathbf{T}\times$ to deal with topicalization, and Steedman (1990) proposes an account of gapping based on a very restricted use of the $\rangle\mathbf{B}\times$ rule, both of which we shall ignore here.

5.2.3 Comparisons

We have seen that the order-preserving rules (beyond application) of CCG are used to yield an account of flexible constituency, whereas the non-order-preserving rules are being used to deal with non-sequentiality. In view of this, it is natural to consider both how the version of flexible constituency implicit in CCG compares with dependency constituency, and how the accounts of non-sequentiality compare with the use of structural modalities.

Take flexible constituency first. As noted earlier, the operation of type-raising is not dependency-preserving, and so the rules as stated above can generate expressions that are not dependency constituents, e.g. Fred said Bill:

(5.39)
$$\dfrac{\dfrac{\dfrac{\text{Fred}}{\text{NP}}}{\text{S/(NP\S)}}\rangle\mathbf{T} \quad \dfrac{\dfrac{\text{said}}{\text{(NP\S)/S}} \quad \dfrac{\dfrac{\text{Bill}}{\text{NP}}}{\text{S/(NP\S)}}\rangle\mathbf{T}}{\text{(NP\S)/(NP\S)}}\rangle\mathbf{B}}{\text{S/(NP\S)}}\rangle\mathbf{B}$$

On the usual assumption that constituents of the same type can be coordinated, this allows the incorrect coordination in (5.40):

(5.40) *[Mary] and [Fred said Bill] laughed.

In fact Steedman agrees with the dependency constituency account in assuming that Fred said Bill should not be analysed as a constituent, and is thus forced to place an ad-hoc restriction on the forward harmonic composition rule as follows:

(5.41) Forward harmonic composition ($\rangle\mathbf{B}$)

$\phi : X/Y \quad \chi : Y/Z \Rightarrow \mathbf{B}\phi\chi : X/Z$ where $Z \neq \text{NP\S}$

Such a move would not be necessary if the non-dependency-preserving rule of type-raising were absent from the grammar, although a replacement operation would be needed to combine verbs (other than intransitive verbs) with subjects, such as the rule of associativity in (5.20) above, which is dependency-preserving.[1] Note also that with the restriction in (5.41) it is now not possible to derive **Fred said Bill and Tom said Mary laughed**, since there is no way within CCG of coordinating expressions that are not constituents.

It is also possible for an expression to be a dependency constituent but not a CCG constituent. For example, Steedman treats modifiers as functors in the usual categorial fashion but forbids modified elements to type-raise over them. In the absence of associativity, this means that an incomplete relative clause such as **a man who likes** cannot be analysed as a CCG constituent:

(5.42)

a	**man**	**who**	**likes**	
		$\overline{}$	$\overline{}$	
$\overline{}$	$\overline{}$	$(N\backslash N)/(NP\backslash S)$	$(NP\backslash S)/NP$	
NP/N	N	$\overline{}$		$\rangle \mathbf{B}$
		$(N\backslash N)/NP$		

In contrast, **a man who likes** would be analysed as a dependency constituent irrespective of whether modifiers or modified elements are viewed as functors. The CCG analysis is partly motivated by the desire to capture island constraints as in (5.43) (which cannot be derived from dependency constituency alone):

(5.43) ***Beans, I met a man who likes.**

However, it also incorrectly rules out coordinations such as (5.44):

(5.44) **Most people like, but I know a man who hates, sonatas by Mozart.**

---

[1]The rule could be restricted in a similar way to the type-raising rule, e.g. as in (i):

(i) $\alpha : (NP\backslash S)/X \Rightarrow \mathbf{C}\alpha : NP\backslash(S/X)$

(5.44) would be allowed in an account of coordination based on dependency constituency, because the conjuncts are both dependency constituents of type S/NP.

Even if the CCG rules were modified to generate only dependency constituents, it would still be impossible to account for the parallel coordination phenomena described earlier, since they require the generation of expressions that are not dependency constituents. For instance, consider (5.45):

(5.45) John loves [Mary madly] and [Sue passionately].

Dowty (1988) deals with such constructions by using the combinatory rules ⟨**B** and ⟨**T**; note that ⟨**T** is not dependency-preserving.[2]

(5.46)
$$\dfrac{\dfrac{\dfrac{\text{Mary}}{\text{NP}}}{(\text{(NP\S)/NP})\backslash\text{(NP\S)}}\langle\mathbf{T} \quad \dfrac{\text{madly}}{\text{(NP\S)}\backslash\text{(NP\S)}}}{\text{((NP\S)/NP)}\backslash\text{(NP\S)}}\langle\mathbf{B}$$

The conjuncts in (5.45) are constituents under Dowty's analysis, but not dependency constituents. However, this means that Dowty is unable to prevent the forbidden coordination in (5.47), since both Mary madly and Sue can be analysed as constituents of type ((NP\S)/NP)\(NP\S):

(5.47) *John loves [Mary madly] and [Sue].

Comparisons between the CCG treatment of non-sequentiality and the treatments using structural modalities are a little less straightforward, because they work on different principles. For example, return to the non-peripheral extraction example who Mary greeted warmly given earlier. Given the type (N\N)/(S/NP) for who, it is impossible to generate this construction using order-preserving rules, because S/NP is only assignable to expressions consisting of an S missing an NP at the rightmost end. The structural modality

---

[2]This example follows Dowty in taking modifiers to be functors, but a similar argument can be constructed if modified elements are taken to be functors.

approach solves this by refining the type of **who**, whereas the CCG approach solves it by (effectively) extending the interpretation of S/NP. The CCG approach thus appears to be at a disadvantage in one respect, since it allows overgenerations such as the following:

(5.48) ***Mary greeted warmly John**

$$\dfrac{\dfrac{\dfrac{}{\text{NP}}\quad\dfrac{\dfrac{\dfrac{\text{(NP\S)/NP}}{}\quad\dfrac{\text{(NP\S)\(NP\S)}}{}}{\text{(NP\S)/NP}}\langle\mathbf{B}\times\quad\dfrac{}{\text{NP}}}{\text{NP\S}}\rangle}{\text{S}}\langle$$

Steedman argues that such a mechanism is in fact desirable since it allows the generation of heavy-NP-shifted constructions such as **Mary greeted warmly the winner of the 1989 Booker Prize**, but this does not explain why the construction in (5.48) is ungrammatical, nor why the addition of another modifier is impossible (e.g.***Mary greeted warmly the winner of the 1989 Booker Prize with open arms**). Similar use of the $\langle\mathbf{S}\times$ rule incorrectly generates constructions such as ***Mary passed without greeting John.**

On the other hand, CCG's Principle of Inheritance prevents the generation of certain constructions that would be allowed by a naive structural modality approach, such as the following:

(5.49) ***who(m) I think that arrived**

$$\dfrac{}{\text{(N\N)/(S/NP)}}\quad\dfrac{\dfrac{\dfrac{\text{I}}{\text{NP}}}{\text{S/(NP\S)}}\rangle\mathbf{T}\quad\dfrac{\dfrac{\text{think}}{\text{(NP\S)/SP}}\quad\dfrac{\text{that}}{\text{SP/S}}}{\text{(NP\S)/S}}\rangle\mathbf{B}}{\text{S/S}}\rangle\mathbf{B}\quad\dfrac{}{\text{NP\S}}$$

Because the subject of **arrived** is a leftward-directional argument, by Inheritance there can be no combinatory rule that would turn it into the rightward-directional argument required within the argument of **who**. On the other hand, as we saw in (4.51), the structural modality system with the type $(N\backslash N)/(S/\triangle NP)$ for **who** would incorrectly generate this construction.[3] Similar remarks extend to multiple extraction. CCG's inheritance

[3]Such an analysis could however be blocked by case features on the NP, so there is no need to make an appeal to argument directionality in this case.

principle would automatically block **who loves** (where both subject and object have been extracted) as a relative clause, since the two extracted arguments have different directionalities, whereas the naive account based on structural modalities allows it if case is not taken into consideration (as shown in (4.52)).

Less obviously, CCG also correctly blocks certain multiple extractions where the extracted arguments have the *same* directionality, e.g. **who I showed** (where direct and indirect object have been extracted), since there is no rule of the form $(X/Y)/Y \Rightarrow X/Y$ (which would be derivable in **LPC**):

(5.50)
$$\frac{\text{*who(m)}}{(N\backslash N)/(S/NP)} \qquad \frac{\dfrac{\text{I}}{NP}}{S/(NP\backslash S)}\,\rangle\mathbf{T} \qquad \frac{\text{showed}}{((NP\backslash S)/NP)/NP}$$

It is not obvious how the present account would block this; the relevant intuition appears to be that the extracted arguments should be dependent on different heads, which cannot be captured by means of the devices introduced in earlier chapters. One possible approach is not to regard parasitic extraction as a special syntactic phenomenon at all, but to regard it as a form of control. For example, since **without** can control the subject of a subordinate verb as in **John left without apologizing**, it could be argued that it controls both the subject and the object in **the man who Mary passed without greeting**, and thus has the additional type $((NP\backslash S)\backslash(NP\backslash S))/VP$. This would remove the need to use the iteration modality in the type of the relative pronoun (and in fact would leave no obvious role for the iteration modality to play in syntax).

In conclusion, it seems that the rules of CCG capture many of the intuitions of chapters 3 and 4, and that each account has its positive and negative aspects from a linguistic point of view, but that the account presented here is mathematically better motivated. This is not of course to say that combinatory formulations of CG are in themselves less mathematically rigorous; in particular, a combinatory formulation of a dependency calculus closely related to **D** is given in Hepple (1991).

5.3 Further Remarks and Conclusions

In this final section I shall attempt to indicate how further research might proceed from the material in the earlier sections, and draw some conclusions.

Chapter 3 essentially put forward three arguments; that theories of rigid constituency are inadequate for linguistic description, that a linguistically appropriate notion of flexible constituency can be derived from a dependency grammar, and that (subject to certain assumptions) this notion of constituency can be captured in a categorial grammar with a suitable reinterpretation of types. Chapter 4 addressed the lack of coverage of categorial grammar by advancing the claim that the categorial hierarchy can be used to motivate the addition of operators to deal with non-sequential constructions. Section 5.1 made steps towards combining the approaches of chapters 3 and 4, and section 5.2 compared the entire approach with the rule-based alternative of CCG.

Each of these lines of enquiry gives rise to further questions. The case for flexible constituency has not often been put forward outside the context of CG, and consequently independent evidence for flexible constituency has rarely been explored, other theories of grammar almost always taking rigid constituency as axiomatic. In view of this, it would be advantageous if a more detailed linguistic study of the evidence for and against flexible constituency could be carried out in a theory-neutral setting. Similarly, the dependency grammar tradition has not generally been concerned with the idea of constituency (although the relation between dependency and full constituency has been pointed out in various places). It appears that the study of dependency grammar could benefit from an analysis of the forms of constituency that can be derived from it. In addition to this, a more thorough investigation of the correspondence between categorial grammar and dependency grammar demonstrated in section 3.3 would help to shed light on both formalisms, especially in respect of those cases such as modifier constructions where there is a mismatch between traditional notions of head and functor.

The material in chapter 4 presents both logical and linguistic questions. The main log-

ical question is whether an adequate interpretation can be given to the structural modalities in an analogous fashion to the interpretation of standard categorial types; this would presumably require the construction of models where the underlying algebra contained commutable, iterable or optional elements. The main linguistic question appears to be whether the tools presented there can be combined with other devices to produce a linguistically adequate account of the phenomena described there. It may turn out that some of the operators presented are in fact better replaced with operators more specific to the constructions concerned.

The idea of an all-encompassing network of categorial logics, the possibility of adding further operators for linguistic purposes, and the programme of fitting existing categorial grammar formalisms into this network, all appear to be fruitful ways of attempting to tie together the diverse family of categorial grammars that have been used in linguistic description.

This thesis has attempted to demonstrate some of the ways in which categorial grammar can be adapted to deal with a wider range of linguistic phenomena, without losing mathematical precision. In my opinion this has served to demonstrate both a strength and a weakness of the formalism. The strength lies in the fact that since all the rules of CG are logical validities, it can be adapted to different linguistic requirements simply by changing the interpretation of types. The weakness lies in the fact that it can be difficult to accommodate linguistic phenomena which do not fit straightforwardly into the given interpretation, even when new operators are added. Whether or not the case has yet been proven for the use of CG as a general-purpose linguistic tool, it remains a rich area to explore for both the mathematician and the linguist.

References

Abrusci, V.M. (1989). Non-commutative classical linear propositional logic, I. Rapporto Scientifico N.8, Dipartmento di Scienze Filosofiche, University of Bari.

Ades, A.E. and Steedman, M.J. (1982). On the order of words. *Linguistics and Philosophy 4*, 517–558.

Ajdukiewicz, K. (1935). Die syntaktische Konnexität. *Studia Philosophica 1*, 1–27. Translated as 'Syntactic connexion' in S. McCall (Ed., 1967), *Polish Logic: 1920–1939*, Oxford University Press, Oxford, 207–231.

Anderson, A.R. and Belnap, N.D. (1975). *Entailment, Volume 1.* Princeton University Press, Princeton, New Jersey.

Bach, E. (1983). On the relationship between word-grammar and phrase-grammar. *Natural Language and Linguistic Theory 1*, 65–89.

Bach, E. (1984). Some generalizations of categorial grammars. In F. Landman and F. Veltman (eds), *Varieties of Formal Semantics*, Foris, Dordrecht, 1–23.

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

Bar-Hillel, Y., Gaifman, C. and Shamir, E. (1960). On Categorial and Phrase Structure Grammars. In *Language and Information.* Also appeared in The Bulletin of the Research Council of Israel, 9F.1.16.

Barry, G., Hepple, M., Leslie, N. and Morrill, G. (1991). Proof figures and structural operators for categorial grammar. In *Proceedings of the Fifth Conference of the European Chapter of the Association for Computational Linguistics*, 198–203.

Barry, G. and Pickering, M. (1990). Dependency and constituency in categorial grammar. In G. Barry and G. Morrill (eds), *Edinburgh Working Papers in Cognitive Science, Volume 5: Studies in Categorial Grammar*, Centre for Cognitive Science, University of Edinburgh, 23–45.

van Benthem, J. (1983). *The semantics of variety in categorial grammar.* Report 83–29, Department of Mathematics, Simon Fraser University. Also in W. Buszkowski, W. Marciszewski and J. van Benthem (eds), *Categorial Grammar*, Volume 25, Linguistic & Literary Studies in Eastern Europe, John Benjamins, Amsterdam/Philadelphia, 57–84.

van Benthem, J. (1986). Categorial grammar. In *Essays in Logical Semantics*, Volume 8, Studies in Linguistics and Philosophy, D. Reidel, Dordrecht, 123–150.

van Benthem, J. (1987). Categorial grammar and type theory. Prepublication Series 87–07, Institute for Language, Logic and Information, University of Amsterdam.

Bird, S. (1991). Focus and phrasing in Unification Categorial Grammar. In S. Bird (ed.), *Edinburgh Working Papers in Cognitive Science, Volume 7: Declarative Perspectives on Phonology*, Centre for Cognitive Science, University of Edinburgh, 139–165.

Bouma, G. (1988). Modifiers and specifiers in categorial unification grammar. *Linguistics 26*, 21–46.

Bouma, G. (1989). Efficient processing of flexible categorial grammar. In *Proceedings of the Fourth Conference of the European Chapter of the ACL*, UMIST, Manchester.

Buszkowski, W. (1983). Algebraic models of categorial grammars. In G. Dorn and P. Weingartner, *Foundations of Logic and Linguistics: Problems and their Solutions*, Plenum Press, New York.

Chomsky, N. (1965). *Aspects of the Theory of Syntax.* MIT Press, Cambridge, Massachusetts.

Corbett, G.G. (1979). The agreement hierarchy. *Journal of Linguistics 15*, 203–224.

Curry, H.B. and Feys, R. (1958). *Combinatory Logic, Volume I.* North Holland, Amsterdam.

Dosen, K. (1990). Modal logic as metalogic. In P. Petkov (ed.), *Proceedings of the "Kleene '90" Conference*, Springer-Verlag.

Dowty, D. (1988). Type raising, functional composition, and non-constituent conjunction. In R. Oehrle, E. Bach and D. Wheeler (eds), *Categorial Grammars and Natural Language Structures*, D. Reidel, Dordrecht, 153–197.

Fraser, N. (1989). Parsing and dependency grammar. In R. Carston (ed.), *UCL Working Papers in Linguistics*, Volume 1, 296–319.

Gaifman, H. (1965). Dependency systems and phrase-structure systems. *Information and Control 8*, 304–337.

Gazdar, G., Klein, E., Pullum, G. and Sag, I. (1985). *Generalized Phrase Structure Grammar.* Blackwell, London.

Gentzen, G. (1936). On the meanings of the logical constants. In Szabo (ed., 1969), *The Collected Papers of Gerhard Gentzen*, North Holland, Amsterdam.

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

Girard, J-Y. (1989). Towards a geometry of interaction. *Proceedings of the AMS Conference on Categories, Logic and Computer Science.*

Goodall, G. (1987). *Parallel Structures in Syntax: Coordination, Causatives and Restructuring.* Cambridge: Cambridge University Press.

Hepple, M. (1990). *The Grammar and Processing of Order and Dependency: a Categorial Approach.* PhD thesis, Centre for Cognitive Science, University of Edinburgh.

Hepple, M. (1991). Efficient incremental processing with categorial grammar. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, 79–86.

Hepple, M. and Morrill, G. (1989). Parsing and derivational equivalence. In *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, UMIST, Manchester.

Howard, W.A. (1969). The formulae-as-types notion of construction. In J.R. Hindley and J.P. Seldin (eds, 1980), *To H.B. Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press.

Hudson, R.A. (1984). *Word Grammar.* Basil Blackwell, Oxford.

Hudson, R.A. (1987). Zwicky on heads. *Journal of Linguistics 23*, 109–132.

Jackendoff, R. (1977). $\overline{X}$ *Syntax: a Study of Phrase Structure.* MIT Press, Cambridge, Mass.

Keenan 1974 — functional control

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

Lambek, J. (1961). On the calculus of syntactic types. In *Structure of Language and its Mathematical Aspects*, Providence, Rhode Island, 166–178.

van der Linden, E.-J. (1991). Accent placement and focus in categorial logic. In S. Bird (ed.), *Edinburgh Working Papers in Cognitive Science, Volume 7: Declarative Perspectives on Phonology*, Centre for Cognitive Science, University of Edinburgh, 197–217.

Matthews, P. H. (1981). *Syntax.* Cambridge University Press, Cambridge.

Montague, R. (1970). Universal grammar. *Theoria 36*, 373–398. Reprinted in R.H.Thomason (ed, 1974), *Formal Philosophy: Selected Papers of Richard Montague*, Yale University Press, New Haven, Conn, 222–246.

Moortgat, M. (1988). *Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus.* Foris, Dordrecht.

Moortgat, M. (1990a). Cut elimination and the elimination of spurious ambiguity. In *Proceedings of the Seventh Amsterdam Colloquium*, University of Amsterdam.

Moortgat, M. (1990b). The logic of discontinuous type constructors. In *Proceedings of the Symposium on Discontinuous Constituency*, Institute for Language Technology and Information, University of Tilburg.

Morrill, G. (1989). Intensionality and boundedness. *Linguistics and Philosophy 13*, 699–726.

Morrill, G., Leslie, N., Hepple, M. and Barry, G. (1990). Categorial deductions and structural operations. In G. Barry and G. Morrill (eds), *Edinburgh Working Papers in Cognitive Science, Volume 5: Studies in Categorial Grammar*, Centre for Cognitive Science, University of Edinburgh, 1–21.

Oehrle, R. (1987). Boolean properties in the analysis of gapping. In G. Huck and A. Ojeda (eds), *Syntax and Semantics 20: Discontinuous Constituents*, Academic Press, New York, 201–240.

Ono, H. (1988). Structural rules and a logical hierarchy. In *Proceedings of the Conference on Mathematical Logic and its Applications*, North-Holland, Amsterdam.

Palmer, F. (1971). *Grammar.* Penguin Books.

Pareschi, R. and Steedman, M. J. (1987). A lazy way to chart-parse with extended categorial grammars. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford University, Stanford, Ca.

Pickering, M. (1991). *Processing Dependencies.* PhD thesis, Centre for Cognitive Science, University of Edinburgh.

Pollard, C. and Sag, I. (1987). *An Information-Based Approach to Syntax and Semantics: Volume 1 Fundamentals.* CSLI Lecture Notes, Number 13, Stanford.

Prawitz, D. (1965). *Natural Deduction: a Proof Theoretical Study.* Almqvist and Wiksell, Uppsala.

Pullum, G.K. (1982). Free word order and phrase structure rules. In J. Pustejovsky and P. Sells (eds), *Proceedings of the Twelfth Annual Meeting of the North Eastern Linguistic Society*, 209–220.

Radford, A. (1988). *Transformational Grammar: a First Course.* Cambridge University Press, Cambridge.

Robinson, J.J. (1970). Dependency structures and transformational rules. *Language 46*, 259–285.

Sag, I., Gazdar, G., Wasow, T. and Weisler, S. (1985). Coordination and how to distinguish categories. *Natural Language and Linguistic Theory 3*, 117–171.

Steedman, M.J. (1987). Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory 5*, 403–439.

Steedman, M.J. (1990). Gapping as constituent coordination. *Linguistics and Philosophy 13*, 207–263.

Steedman, M.J. (1991). Type-raising and directionality in combinatory grammar. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics.*

Uszkoreit, H. (1987). *Word Order and Constituent Structure in German.* CSLI Lecture Notes.

Vennemann, T. and Harlow, R. (1977). Categorial grammar and consistent basic VX serialization. *Theoretical Linguistics 4*, 227–254.

Wansing, H. (1989). Relevant quasi-deductions, weak implicational logics, and operational semantics. Ms., Institut für Philosophie, Freie Universität Berlin.

Williams, E. (1978). Across-the-board rule application. *Linguistic Inquiry 9*, 31–43.

Wood, M. (1988). *A Categorial Syntax for Coordinate Constructions.* PhD thesis, University College London. Available as Technical Report UMCS–89–2–1, Department of Computer Science, University of Manchester.

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

Zwicky, A.M. (1985). Heads. *Journal of Linguistics 21*, 1–29.

Categorial Deductions and Structural Operations*

Glyn Morrill, Neil Leslie, Mark Hepple and Guy Barry

Categorial grammar is an approach to language description in which expressions are classified according to recursively defined categories. The combination of expressions is governed not by specific linguistic rules but by general logical inference mechanisms, so that all the information about the combining potential of a word resides in its lexical categorization. Various systems of inference for categorial grammar have been proposed, in particular the 'syntactic calculus' of Lambek (1958). However, use of the Lambek calculus for linguistic work has generally been rather limited. There appear to be two main reasons for this: the notations most commonly used can sometimes obscure the structure of proofs and fail to clearly convey linguistic structure, and the calculus as it stands is apparently not powerful enough to describe many phenomena encountered in natural language.

In this paper we suggest ways of dealing with both these deficiencies. Firstly, we reformulate the Lambek calculus using proof figures based on the 'natural deduction' notation commonly used for derivations in logic. Natural deduction is generally regarded as the most economical and comprehensible system for working on proofs by hand, and we suggest that the same advantages hold for a similar presentation of categorial derivations. Secondly, we introduce devices for the characterization of word order variation, iteration and optionality, features of natural language which the Lambek calculus does not capture with the desired sensitivity and generality. This is achieved by means of operators called *structural modalities*, derived from the usual *structural rules* found in logic.

1 Fregean Analysis

The point of departure for categorial grammar can be seen as Frege's position that there are certain 'complete expressions' which are the primary bearers of meaning, and that the meanings of 'incomplete expressions' (including words) are derivative, being their contribution to the meanings of the expressions in which they occur. Frege took proper names and sentences to be the complete expressions. Consider a language containing some proper names {"John", "Mary", ...} and some sentences {"John walks", "Mary walks", ..., "John likes John", ...}; such sets of linguistic objects will be called *categories*, and will be indexed by *types*, in this case NP and S respectively.[1] Now, the significance of "walks" is that it occurs with an NP to form an S, and it is attributed with a meaning which is a function from NP meanings into the meanings of the S's it forms with those NP's; the category of such objects can be indexed

*We would like to thank Martin Pickering and Pete Whitelock for comments and discussion relating to this work. The authors were respectively supported by SERC Postdoctoral Fellowship B/ITF/206; ESPRIT Project 393 and Cognitive Science/HCI Research Initiative 89/CS01 and 89/CS25; ESRC Research Studentship C00428722003; SERC Research Studentship 88306971.

[1]Thus a type is a symbol denoting a category. There are alternative terminologies, e.g. use of *category* to mean category symbol, or *type* to mean a domain, rather than a symbol indexing a domain.

by the type NP → S, and we say that (following the Fregean procedure) type NP → S is *assigned* to "walks". Similarly, "likes John", "likes Mary", ... are assigned type NP → S, and consequently "likes" is assigned type NP → (NP → S), etc.

Note that on this view compositionality is methodological rather than empirical: the meanings of parts are *defined* in terms of the meanings of wholes. Note also that the semantic functions we build up in this way fall within a simple type hierarchy; Russell's theory of types, prompted by the paradoxes, was a major influence on Leśniewski's 'semantic categories', the immediate precursor of Ajdukiewicz and categorial grammar.

The above system for semantic type-assignment can be extended to one which is simultaneously a syntactic type-assignment, i.e. one in which the types by which incomplete expressions are classified encode the manner in which they constitute subparts of other expressions. This can be achieved by the use of a *directional* system, in which the undirected implication $Y \rightarrow X$ is replaced by two directed implications X/Y ('X over Y') and $Y\backslash X$ ('Y under X'), representing respectively incomplete expressions that combine with a following or preceding expression of type Y to form an expression of type X.[2] Thus for example "walks" forms an S when preceded by an NP, so is assigned type NP\S, and "likes" forms an NP\S when followed by an NP, so is assigned type (NP\S)/NP.

Linguistic categories are not limited to finite inhabitation, e.g. in general languages contain an infinite number of sentences. However, the set of words from which the objects in those categories are constructed is usually presumed to be a finite set, known as the *vocabulary*. The task of a grammar is to finitely describe the membership of (possibly) infinite categories, and the above type-theoretic perspective suggests a procedure for doing this. By establishing an initial type assignment relation between the vocabulary and types (a *lexicon*), a full type assignment between expressions and types may be determined, according to a general system of type inference. In the next section we shall present such a system of inference for the directed types described above, the calculus **L** of Lambek (1958), and discuss properties and linguistic applications of that system.

2 The Lambek Calculus L

2.1 Preliminaries

Given a set of primitive types, the set of *bidirectional types* is defined as shown in (1).

(1) a. If X is a primitive type
then X is a type.

b. If X and Y are types
then X/Y and $Y\backslash X$ are types.

Let us assume as basic categories prepositional phrases, noun phrases, sentences, and common nouns, indexed by primitive types PP, NP, S, and N respectively. By the principles outlined above, we may assign types to words as follows:

[2] In the alternative notation used by, for example, Steedman, these types would be written X/Y, $X\backslash Y$ respectively.

(2)

for	:= PP/NP
John, Mary	:= NP
likes	:= (NP\S)/NP
man	:= N
the	:= NP/N
thinks	:= (NP\S)/S
votes	:= (NP\S)/PP
who	:= (N\N)/(S/NP)

We assume that words have (at least) two components, form (represented by italics) and meaning (represented by boldface). For example, the word "for" will be taken to have form denoted by *for* and meaning denoted by **for**.

2.2 Proof Figures

We shall present the rules of **L** by means of *proof figures*, based on Prawitz' (1965) systems of 'natural deduction'. Natural deduction was developed by Gentzen (1936) to reflect the natural process of mathematical reasoning in which one uses a number of *inference rules* to justify a single *conclusion* on the basis of a number of propositions, called *assumptions*. During a proof one may *temporarily* make a new assumption if one of the rules licenses the subsequent withdrawal of this assumption. The rule is said to *discharge* the assumption. The conclusion is said to *depend* on the undischarged assumptions, which are called the *hypotheses* of the proof.

A proof is usually represented as a tree with the assumptions as leaves and the conclusion at the root. Finding a proof is then seen as the task of filling this tree in, and the inference rules as operations on the partially completed tree. One can write the inference rules out as such operations, but as these are rather unwieldy it is more usual to present the rules in a more compact form as operations from a set of subproofs (the *premises*) to a conclusion, as follows:

(3)

$$\frac{\begin{array}{ccccccc} \vdots & & \vdots & [Y_1]^i & & [Y_n]^i \\ X_1 & \cdots & X_m & X_{m+1} & \cdots & X_{m+n} \end{array}}{Z}\,\text{Rule}^i$$

This states that a proof of Z can be obtained from proofs of $X_1, \ldots, X_{m+n}$ by discharging appropriate occurrences of assumptions $Y_1, \ldots, Y_n$. The use of square brackets around an assumption indicates its discharge. The index i is used to disambiguate proofs, where there may be an uncertainty as to which rule has discharged which assumption.

As propositions are represented by formulas in logic, so linguistic categories are represented by type formulas in the Lambek calculus. The left-to-right order of types indicates the order in which the forms of subexpressions are to be concatenated to give a composite expression derived by the proof. Thus we must take note of the order and place of occurrence of the premises of the rules in the proof figures for **L**. There is also a problem with the presentation of the rules in the compact notation as some of the rules will be written *as if* they had a number of conclusions, as follows:

(4) $$\frac{\begin{array}{ccc}\vdots & & \vdots \\ X_1 & \cdots & X_m\end{array}}{\begin{array}{ccc}Y_1 & \cdots & Y_n\end{array}}\text{Rule}$$

This rule should be seen as a shorthand for:

(5) $$\frac{\begin{array}{ccc}\vdots & & \vdots \\ X_1 & \cdots & X_m\end{array}}{\begin{array}{ccc}Y_1 & \cdots & Y_n\end{array}}\text{Rule} \quad \begin{array}{ccc}\vdots & & \vdots \\ \vdots & & \vdots\end{array}$$
$$\overline{\qquad\qquad Z \qquad\qquad}$$

If the rules are viewed in this way it will be seen that they do not violate the single conclusion nature of the figures.[3]

As with standard natural deduction, for each connective there is an *introduction* rule which states how a type containing that connective may be derived, and an *elimination* rule which states how a type containing that connective may be consumed. We shall follow the historical development of the calculus by giving the elimination rules before the introduction rules.

The elimination rule for / states that a proof of type X/Y followed by a proof of type Y yields a proof of type X. Similarly the elimination rule for \ states that a proof of type $Y\backslash X$ preceded by a proof of type Y yields a proof of type X. Using the notation above, we may write these rules as follows:

(6) a. $$\frac{\begin{array}{cc}\vdots & \vdots \\ X/Y & Y\end{array}}{X}/\text{E}$$ b. $$\frac{\begin{array}{cc}\vdots & \vdots \\ Y & Y\backslash X\end{array}}{X}\backslash\text{E}$$

The meaning of the composite expression is given by the functional application of the meaning of the *functor* expression (i.e. the one of type X/Y or $Y\backslash X$) to the meaning of the *argument* expression (i.e. the one of type Y). We represent function application by juxtaposition, so that **likes John** means **likes** applied to **John**.

The subsystem of **L** using only the rules /E and \E corresponds to the categorial calculus of Ajdukiewicz (1935) and Bar-Hillel (1953), and is sometimes known as the **AB** calculus. Within this calculus, we may derive "Mary likes John" as a sentence as follows:

(7) $$\frac{\text{Mary} \qquad \frac{\text{likes} \qquad \text{John}}{(\text{NP}\backslash\text{S})/\text{NP} \qquad \text{NP}}/\text{E}}{\text{NP} \qquad\qquad \text{NP}\backslash\text{S}}\backslash\text{E}$$
$$\text{S}$$

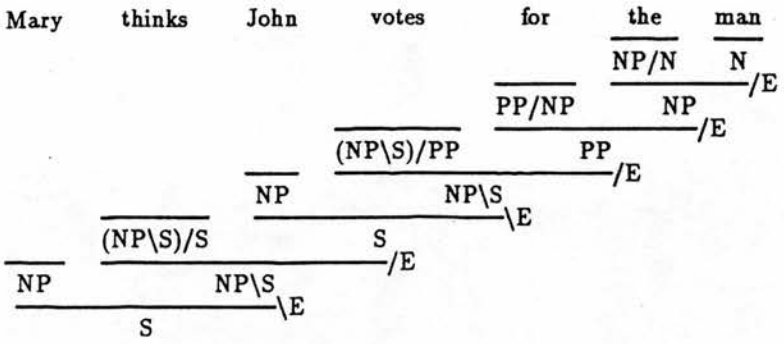
The meaning of the sentence is read off the proof by interpreting the /E and \E inferences as function application, giving the following:

(8) **(likes John) Mary**

Similarly, the sentence "Mary thinks John votes for the man" may be derived thus:

[3]Notice that these rules mean that the proof figures are not trees, but directed acyclic graphs.

(9)



From this we can read off the meaning representation:

(10) **(thinks ((votes (for (the man))) John)) Mary**

The introduction rule for / states that where the rightmost assumption in a proof of the type X is of type Y, that assumption may be discharged to give a proof of the type X/Y. Similarly, the introduction rule for \ states that where the leftmost assumption in a proof of the type X is of type Y, that assumption may be discharged to give a proof of the type $Y\backslash X$. Using the notation above, we may write these rules as follows:

(11) a. $\dfrac{\begin{array}{c}[Y]^n \\ \vdots \\ X\end{array}}{X/Y}/\mathrm{I}^n$ b. $\dfrac{\begin{array}{c}[Y]^n \\ \vdots \\ X\end{array}}{Y\backslash X}\backslash\mathrm{I}^n$

Note however that this notation does not embody the conditions that have been stated, namely that in /I Y is the rightmost undischarged assumption in the proof of X, and in \I Y is the leftmost undischarged assumption in the proof of X. In addition, the Lambek calculus carries the condition that in both /I and \I the sole assumption in a proof cannot be withdrawn, so that no types are assigned to the empty string.[4]

In the introduction rules, the meaning of the result is given by *functional abstraction* over the meaning of the discharged assumption, which can be represented by a variable of the appropriate type. For example, the functional abstraction of [**(likes** x**) Mary**] over the variable x, written λx[**(likes** x**) Mary**], is the function that gives **(likes John) Mary** when applied to **John**, **(likes (the man)) Mary** when applied to **the man**, and so on. This general relation between abstraction, application and substitution is expressed by the law of *β-equality*:

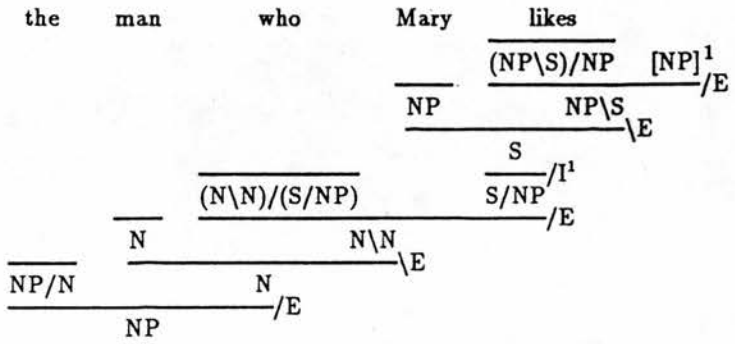
(12) $(\lambda y[\alpha])\beta = \alpha[\beta/y]$

This is read: '$(\lambda y[\alpha])\beta$ is equal to α with β substituted for y'. The rules in (11) are analogous to the usual natural deduction rule of *conditionalization*, except that the latter allows withdrawal of any number of assumptions, in any position. The semantic algebra defined by the above set of inference rules corresponds to the subset of the typed lambda-calculus in which each binder binds exactly one variable (van Benthem 1983). We shall refer to this as the 'single-bind' lambda-calculus. This correspondence between proofs in **L** and meaning representations in the single-bind lambda-calculus can be seen as a version of the Curry-Howard correspondence between intuitionistic implicational deduction and full lambda-calculus (Curry and Feys 1958; Howard 1969).

[4] However, this condition is not an essential feature of the logic (cf. subsection 3.1).

The /I and \I rules are commonly used in constructions that are standardly assumed to involve 'empty categories'. For example, the non-canonical constituent "Mary likes" can be derived as of type S/NP, and hence assignment of type (N\N)/(S/NP) to an object relative pronoun allows analysis of relativization as follows (cf. Ades and Steedman 1982):

(13)



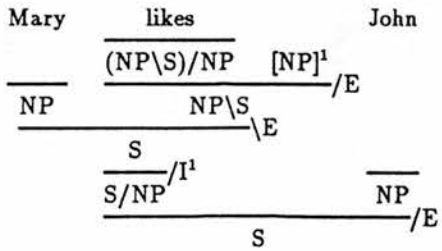
Again, the meaning of the string can be read off the above proof by interpreting /I and \I as functional abstraction, giving the following:

(14) **the (who (λx[(likes x) Mary]) man)**

Note that this mechanism is only powerful enough to allow constructions where the extraction site is clause-peripheral; for non-peripheral extraction, and multiple extraction such as with parasitic gaps, we appear to need an extended logic. Meeting such requirements is part of the concern of section 4 below.

The use of the /I and \I rules in conjunction with the /E and \E rules means that it is possible to give more than one proof for a single reading of a string. For example, compare the derivation of "Mary likes John" in (7), and the corresponding lambda-term in (8), with the derivation in (15) and the lambda-term in (16):

(15)



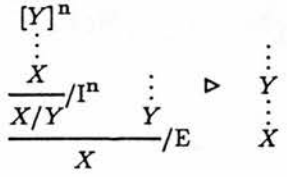
(16) **(λx[(likes x) Mary]) John**

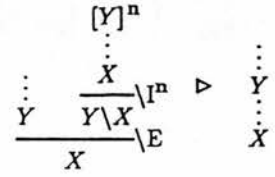
By the definition in (12), the terms in (8) and (16) are β-equal, and thus have the same meaning; the proofs in (7) and (15) are thus equivalent. This property of *derivational equivalence* (also termed 'spurious ambiguity') is common to many versions of categorial grammar, and can cause problems for the design of efficient parsing algorithms, where the usual aim is to find each reading of a string once and only once. Such problems can however be overcome by defining a notion of normal form for proofs (and their corresponding terms) in such a way that each equivalence class of proofs contains a unique normal form (e.g. Hepple and Morrill 1989). One advantage of the present formulation is that it offers a particularly simple way of doing this.

2.3 Normal Forms

In order to define *normal forms* for both L and the single-bind lambda-calculus, we will first define the notions of *contraction* and *reduction*. A contraction schema $R \rhd C$ consists of a particular pattern R within proofs or terms (the *redex*) and an equal and simpler pattern C (the *contractum*). A reduction consists of a series of contractions, each replacing an occurrence of a redex by its contractum. A normal form is a proof or term on which no contractions are possible.

We define the following contraction schemas: *weak contraction* for proofs, and *β-contraction* for the corresponding lambda-terms.

(17) a.


b.


(18) $(\lambda y[\alpha])\beta \rhd \alpha[\beta/y]$

From (12) we see that β-reduction preserves meaning according to the standard functional interpretation of typed lambda-calculus. Therefore the corresponding weak reduction preserves the semantic functional interpretation of the proof; in addition it preserves the syntactic string interpretation since the redex and contractum contain the same leaves in the same order.

For example, the proof in (15) weakly contracts to the proof in (7), and correspondingly the term in (16) β-contracts to the term in (8). The results of these contractions cannot be further contracted and so are the respective results of reduction to *weak normal form* and *β-normal form*.

Weak contraction and β-contraction strictly decrease the sizes of proofs and terms, e.g. the number of symbols in a contractum is always less than that in a redex.[5] Thus there is *strong normalization* with respect to these reductions: every proof (term) reduces to a weak normal form (β-normal form) in a finite number of steps. This has as a corollary (*normalization*) that every proof (term) has a normal form, so that normal forms are *fully* representative: every proof (term) is equal to one in normal form. Since reductions preserve interpretations, an interpretation of a normal form will always be the same as that of the original proof (term). Thus restricting the search to just such proofs addresses the problem of derivational equivalence, while preserving generality in that all interpretations are found.

The single-bind proofs (lambda-terms) appear to straightforwardly inherit the property of intuitionistic proofs (full lambda-terms) that if a proof (term) M reduces to two proofs (terms) N_1, N_2, then there is a proof (term) N_3 to which both N_1 and N_2 reduce. It follows that a proof (term) cannot reduce to more than one normal form, and so every proof (term) has a *unique* normal form. This is known as the *Church-Rosser* property.

If all normal forms were non-equal, restriction of search to normal forms would be optimal in that each equivalence class has a single representative within the narrowed search space. However, the standard extensional functional interpretation of typed lambda-calculus also satisfies the following law of *η-equality*:

[5] This follows immediately from the single-bind character of the languages; if there were multiple binding, contraction could involve duplication of subparts substituted and so could increase size.

(19) $\lambda y[\alpha(y)] = \alpha$
(y not free in α)

Accordingly there is a second form of contraction for each of **L** and the single-bind lambda-calculus, which we shall term *strong contraction* and *η-contraction* respectively. These are as follows:

(20) a.
$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ X/Y\end{array} \quad [Y]^n}{X}/E}{X/Y}/I^n \quad \triangleright \quad \begin{array}{c}\vdots\\ X/Y\end{array}$$

b.
$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ [Y]^n\end{array} \quad \begin{array}{c}\vdots\\ Y\backslash X\end{array}}{X}\backslash E}{Y\backslash X}\backslash I^n \quad \triangleright \quad \begin{array}{c}\vdots\\ Y\backslash X\end{array}$$

(21) $\lambda y[\alpha(y)] \triangleright \alpha$

A proof (term) is said to be in *strong normal form* (*$\beta\eta$-normal form*) if there are no weak or strong contractions (β- or η- contractions) possible on it. Since strong reductions ($\beta\eta$-reductions) also preserve interpretations, it is sufficient to find all and only the strong normal proofs for a given string and a given result type in order to find all possible readings in that type.

A notion of contraction creates a partitioning into normal and non-normal proofs. For the purposes of proof search a structural characterization of just the normal proofs can be exploited (see e.g. König 1989; Hepple 1990). Weak normal proofs can be specified as follows. Suppose we define a *branch* of a proof in **L** as starting with any hypothesis and tracing down until it hits either the conclusion type (a main branch) or the argument type of an elimination inference (a side branch). Then in a weak normal proof no branch can contain an introduction inference followed by an elimination inference. It follows that each branch must consist of a sequence of (zero or more) eliminations followed by a sequence of (zero or more) introductions. However, a similar structural characterization of strong normal proofs is harder to formulate.

2.4 Sequent Calculus

The above notation for proofs is convenient for their presentation on the page, but for the development of proof-search algorithms it is more convenient to use Gentzen's (1936) *sequent calculus* notation, where the objects reasoned about are *sequents* corresponding to entire natural deduction proofs. A sequent calculus consists of axioms and inference rules; axioms state that certain primitive sequents are valid, and inference rules allow new valid sequents to be derived from old ones. A sequent calculus proof may thus be seen as the instructions for the construction of a natural deduction proof. Lambek (1958), in his original presentation of the calculus, used a sequent calculus notation in order to establish its decidability.

We shall use $\Gamma, \Delta, \Phi \ldots$ to stand for sequences of types, and $X, Y, Z \ldots$ to stand for single occurrences of a type. A (single-conclusioned) sequent is a statement $\Gamma \Rightarrow X$; it asserts that there is a deduction of X depending on the hypotheses Γ. In Lambek calculus both the order of the types in Γ and the number of occurrences of each type are relevant, and there is a restriction that Γ must be non-empty.

We shall present the sequent rules in *refinement* notation to emphasize their application to proof search. A rule such as (22) states that Z follows from Γ if X follows from Δ and Y follows from Φ. It does not matter which order we solve the sub-problems in, so long as we solve them all.

(22) $\Gamma \Rightarrow Z$ [Rule]
$\Delta \Rightarrow X$
$\Phi \Rightarrow Y$

In standard sequent systems there is a single schematic axiom saying that every type yields itself, and for each connective there are two inference rules; a *left rule*, which allows us to derive a sequent where the connective newly appears on the left-hand side of the arrow, and a *right rule*, which allows us to derive a sequent where the connective newly appears on the right-hand side of the arrow. The Gentzen-style formulation of **L** follows this pattern:

(23) $X \Rightarrow X$ [Ax]

$\Gamma\ X/Y\ \Delta\ \Phi \Rightarrow Z$ [/L]
$\Delta \Rightarrow Y$
$\Gamma\ X\ \Phi \Rightarrow Z$

$\Gamma\ \Delta\ Y\backslash X\ \Phi \Rightarrow Z$ [\L]
$\Delta \Rightarrow Y$
$\Gamma\ X\ \Phi \Rightarrow Z$

$\Gamma \Rightarrow X/Y$ [/R]
$\Gamma\ Y \Rightarrow X$

$\Gamma \Rightarrow Y\backslash X$ [\R]
$Y\ \Gamma \Rightarrow X$

There is a close analogy between the use of left rules in sequent calculus and elimination rules in natural deduction, and similarly between right rules in sequent calculus and introduction rules in natural deduction. For example, compare the following sequent proof with the proof in (13):

(24)
NP/N N (N\N)/(S/NP) NP (NP\S)/NP ⇒ NP [/L]
N (N\N)/(S/NP) NP (NP\S)/NP ⇒ N [/L]
NP (NP\S)/NP ⇒ S/NP [/R]
NP (NP\S)/NP NP ⇒ S [/L]
NP ⇒ NP [Ax]
NP NP\S ⇒ S [\L]
NP ⇒ NP [Ax]
S⇒S [Ax]
N N\N ⇒ N [\L]
N ⇒ N [Ax]
N ⇒ N [Ax]
NP ⇒ NP [Ax]

L may be formulated with one additional rule, the *Cut rule*, which combines two proofs by identifying the conclusion of one with one of the premises of the other:

(25) $\Delta\ \Gamma\ \Phi \Rightarrow Y$ [Cut]
$\Gamma \Rightarrow X$
$\Delta\ X\ \Phi \Rightarrow Y$

Lambek shows that the sequent formulation of **L** has the property of *Cut-elimination*, i.e. every **L** proof can be transformed into an **L** proof with the same end-sequent, in which no Cuts occur. Cut-elimination in sequent calculus corresponds (approximately) to normalization in natural deduction (cf. the corresponding result for intuitionistic logic; see also Moortgat 1990a). The Cut-free formulation provides a decision procedure. A step in the search for a proof of a sequent can only be an axiom matching, or one of a finite number of possible

rule applications, these creating strictly simpler subgoals since each rule application removes a connective.

Moortgat (1988) shows how the types in sequents in **L** may be annotated with lambda-terms to indicate their semantic content; the resulting terms can be shown to be the same as those obtained in the natural deduction style formulation.

3 A Structural Hierarchy of Logics

The rest of this paper will be concerned with the development of mechanisms extending **L** to a calculus with greater linguistic applicability, in particular addressing the problems of word order variation, iteration and optionality. In this section we describe how **L** can be extended to yield a structural hierarchy of logics, and in the next section we refine these extensions in a manner suited to linguistic description.

From a logical perspective, **L** can be thought of as the weakest member of a hierarchy of implicational type-inference logics, differing in the amount of freedom allowed in the use of assumptions; **L** takes into account the order of assumptions and the number of occurrences of each, but we may choose to ignore these factors to a greater or lesser extent. This 'categorial hierarchy' is discussed in e.g. van Benthem (1987), Moortgat (1988), Ono (1988) and Wansing (1989). It consists of **L** followed by what are essentially the implicational fragments of linear, relevance and intuitionistic logic.

Sequent calculus differentiates between levels in the hierarchy by means of *structural rules* explicitly manipulating lists of assumptions. In natural deduction style formulations, on the other hand, structural operations are implicit in the treatment of occurrences of assumptions and the conditions on their discharge. In the presentations below we shall break with common practice by including explicit structural rules in proof figures; this will enable a closer comparison with the structural operators to be discussed in the next section.

3.1 L and Sequential Logic

L as it stands forms a 'sequential logic', which is a subset of *non-commutative linear logic* (see Girard 1989). The single-conclusioned, implicational fragment of non-commutative linear logic corresponds exactly to **L**, except that proofs from no premises are permissible in the former.

3.2 LP and Linear Logic

The 'Lambek-van Benthem' calculus **LP** (van Benthem 1983, 1986) is like **L** except that the order of premises is irrelevant. We may implement this by adding a structural rule of *permutation* to the natural deduction proof figures:

$$(26)\quad \begin{array}{cc} \vdots & \vdots \\ X & Y \\ \hline Y & X \end{array} \mathrm{P}$$

This is the first appearance of a rule that is not single-conclusioned. As noted earlier, such a rule may not form the last inference in a proof, since complete proofs must have a single conclusion type.

Addition of the permutation rule means that the directional implicational types X/Y, $Y\backslash X$ are equivalent, since they are mutually derivable:

(27)
$$\dfrac{\dfrac{[Y]^1 \quad X/Y}{X/Y \quad Y}\mathrm{P}}{\dfrac{X}{Y\backslash X}\backslash\mathrm{I}^1}/\mathrm{E} \qquad\qquad \dfrac{\dfrac{Y\backslash X \quad [Y]^1}{Y \quad Y\backslash X}\mathrm{P}}{\dfrac{X}{X/Y}/\mathrm{I}^1}\backslash\mathrm{E}$$

For the rest of this section, therefore, we need use only a single implicational type; we choose $Y\backslash X$, and notate it in the more conventional fashion as $Y\rightarrow X$. We repeat the elimination and introduction rules here:

(28)
$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ Y & Y\rightarrow X\end{array}}{X}\rightarrow\mathrm{E} \qquad\qquad \dfrac{\begin{array}{c}[Y]^n \\ \vdots \\ X\end{array}}{Y\rightarrow X}\rightarrow\mathrm{I}^n$$

Without loss of generality, rule applications can be kept subject to the previous ordering conditions for \. For example, we may derive the transition $X\rightarrow(Y\rightarrow Z) \Rightarrow Y\rightarrow(X\rightarrow Z)$ in **LP** as follows:

(29)
$$\dfrac{\dfrac{\dfrac{\dfrac{[X]^1 \quad [Y]^2}{Y \quad X}\mathrm{P} \quad X\rightarrow(Y\rightarrow Z)}{Y\rightarrow Z}\rightarrow\mathrm{E}}{Z}\rightarrow\mathrm{E}}{\dfrac{X\rightarrow Z}{Y\rightarrow(X\rightarrow Z)}\rightarrow\mathrm{I}^2}\rightarrow\mathrm{I}^1$$

However, since the order of premises is irrelevant, the logic is unaffected by allowing the argument type Y in the $\rightarrow$E rule to occur either to the left or the right of the functor type $Y\rightarrow X$, and Y to be *any* undischarged assumption above X in the $\rightarrow$I rule. This removes the need for an explicit permutation rule. In a sequent calculus presentation of **LP**, the structural rule of permutation runs as follows:

(30)
$$\dfrac{\Gamma\ X\ Y\ \Delta \Rightarrow Z}{\Gamma\ Y\ X\ \Delta \Rightarrow Z}\ [\mathrm{P}]$$

LP corresponds to the single-bind lambda-calculus, and is a subsystem of Girard's (1987) *linear logic*, in which permutation of assumptions preserves derivability; the implication in **LP** is like Girard's *linear implication*.

3.3 LPC and Relevance Logic

LPC is an extension of **LP** which allows assumptions to be used more than once. This may be achieved by adding a structural rule of *contraction* to the natural deduction formulation:

(31)
$$\dfrac{\begin{array}{c}\Gamma \\ \vdots \\ X\end{array}}{X \quad X}\mathrm{C}$$

The $\rightarrow$E and $\rightarrow$I rules are as before in this formulation. By way of example of derivation in **LPC**, the transition $X\rightarrow(X\rightarrow Y) \Rightarrow X\rightarrow Y$ is proved as follows:

(32)
$$\dfrac{\dfrac{\dfrac{[X]^1}{X}\mathrm{C} \quad \dfrac{X \quad X\rightarrow(X\rightarrow Y)}{X\rightarrow Y}\rightarrow\mathrm{E}}{Y}\rightarrow\mathrm{E}}{X\rightarrow Y}\rightarrow\mathrm{I}^1$$

LPC could be alternatively formulated in a natural deduction style by relaxing the directionality conditions on **L** and allowing withdrawal of one or more occurrences of a conditionalized assumption. This removes the need for explicit rules of permutation and contraction. The sequent calculus formulation of the contraction rule is as follows:

(33) $\Gamma\ X\ \Delta \Rightarrow Z$ [C]
$\Gamma\ X\ X\ \Delta \Rightarrow Z$

LPC corresponds to the multiple-bind lambda-calculus without vacuous abstraction, and it shares with *relevance logic* (Anderson and Belnap 1975) the preservation of derivability under the structural operations of permutation and contraction.

3.4 LPCW and Intuitionistic Logic

Finally we may extend **LPC** to **LPCW**, where it is permissible for assumptions not to be used at all. This may be achieved by adding a structural rule of *weakening* to the natural deduction formulation:

(34)
$$\dfrac{\vdots}{\dfrac{X}{}\mathrm{W}}$$

In fact, in order that a weakened assumption may be understood as a subpart of a proof, applications of weakening in a derivation will be represented as follows:

(35) a.
$$\dfrac{\begin{array}{cc}\vdots & \vdots\\ X & Y\end{array}}{Y}\mathrm{W_l}$$
b.
$$\dfrac{\begin{array}{cc}\vdots & \vdots\\ Y & X\end{array}}{Y}\mathrm{W_r}$$

For example, the transition $Y \Rightarrow X\rightarrow Y$ is valid in **LPCW**:

(36)
$$\dfrac{\dfrac{[X]^1 \quad Y}{Y}\mathrm{W_l}}{X\rightarrow Y}\rightarrow\mathrm{I}^1$$

This derivation satisfies the ordering conditions on conditionalization since X is indeed the leftmost assumption dominating Y. **LPCW** could be alternatively formulated in a natural deduction style by relaxing the directionality conditions on **L** and allowing withdrawal of zero or more occurrences of a conditionalized assumption. This removes the need for explicit rules of permutation, contraction and weakening. The sequent calculus formulation of the weakening rule is as follows:

(37) $\Gamma\ X\ \Delta \Rightarrow Z$ [W]
$\Gamma\ \Delta \Rightarrow Z$

LPCW corresponds to the full lambda-calculus allowing vacuous abstraction, and it shares with *intuitionistic logic* the preservation of derivability under the structural operations of permutation, contraction and weakening.

4 Structural Modalities

Freely applying structural rules are clearly not appropriate in categorial grammars for linguistic description. Permutation would imply that word order was irrelevant; contraction would imply that any word in a sentence could fulfil any number of roles; and weakening would imply that sentences could contain arbitrary words not contributing to their meaning. Nevertheless, commutable, iterable, and optional elements do arise in natural language: a relative pronoun may license a gap[6] anywhere in body of the relative clause, it may license multiple gaps (as in parasitic gap constructions), and a gap may be optionally required (perhaps as in resumptive pronoun constructions); similarly a head may allow some freedom in the location of a complement, it may allow multiple complements (as in iterated coordination), and it may optionally take a complement. This suggests that we should be able to indicate that structural operations are permissible on specific types, while still forbidding them as general inference rules.

There is a precedent for the above in (commutative) linear logic, which contains the so-called *exponentials* ! ('of course') and ? ('why not'), which bring back the structural rules of contraction and weakening in a controlled way. The behaviour of ! and ? bears a resemblance to that of the operators $\Box$ (universal modality) and $\Diamond$ (existential modality) in some modal logics. For our linguistic calculus, which has rather different applications, we shall suggest a system of operators called *structural modalities*. Corresponding to commutation, iteration and optionality, we define both universal and existential structural modalities; there are two directional versions for each of the commutation modalities.

(38)

	Universal	Existential
Commutation	$\triangleright\triangleleft$	$>\!<$
Iteration	!	+
Optionality	‖	?

Broadly speaking the universal modalities may be said to represent 'any' (any location, any number of occurrences greater than zero, any number of occurrences less than two), and the existential modalities to represent 'some' (some location, some number of occurrences greater than zero, some number of occurrences less than two). For each modality there is an 'operational rule' which governs its behaviour; the operational rules for universal modalities are analogous to the corresponding structural rules, and those for existential modalities are analogous to inverted versions of the corresponding structural rules. We shall examine these rules in detail for each operator below.

Universal modalities have the common property that any item of a universally modal type also has the corresponding non-modal type, and the existential modalities have the property that any item of a non-modal type also has the corresponding existentially modal type. This means there are straightforward elimination rules for the universal modalities and introduction rules for the existential modalities, but the complementary rules in each case are less straightforward. Since we are not fixing here a particular semantics with respect to which

[6] We use the term 'gap' purely descriptively.

we expect the logic to be complete, we shall be ignoring introduction rules for the universal modalities and elimination rules for the existential modalities.

In grammar, the universals and existentials will effect the structural operations with respect to gaps and realized elements respectively. When we look at lexical type-assignments involving these modalities, we will find that existential modalities are appropriate for describing the behaviour of lexical items that subcategorize for commutable, iterable or optional arguments; such items typically receive types of the form $\Diamond Y \rightarrow X$, where $\Diamond$ represents one of the existential modalities and $\rightarrow$ one of the implications.[7] Conversely, universal modalities are appropriate for describing the behaviour of lexical items that subcategorize for strings that are missing commutable, iterable or optional arguments; such items typically receive types of the form $(\Box Z \rightarrow Y) \rightarrow X$, where $\Box$ represents one of the universal modalities. As we shall see in the examples, it follows from these assignments that the missing introduction and elimination rules are rarely needed for linguistic purposes.

4.1 Commutation Modalities

We notate the two universal commutation modalities as $\triangleright$ (written as a prefix operator) and $\triangleleft$ (written as a postfix operator). The type $\triangleright X$ (resp. $X \triangleleft$) is assigned to an item of type X which may occur in its current position or anywhere to the right (resp. left) of its current position. This is achieved by means of the rules $\triangleright$Perm and $\triangleleft$Perm, which allow types bearing the appropriate modality to be moved to any position on the given side, and the rules $\triangleright$E and $\triangleleft$E, which allow the modality to be dropped:

(39) a. $\dfrac{\begin{array}{cc}\vdots & \vdots \\ \triangleright X & Y\end{array}}{\begin{array}{cc} Y & \triangleright X\end{array}}\triangleright\text{Perm}$ b. $\dfrac{\begin{array}{cc}\vdots & \vdots \\ Y & X \triangleleft\end{array}}{\begin{array}{cc} X \triangleleft & Y\end{array}}\triangleleft\text{Perm}$

(40) a. $\dfrac{\begin{array}{c}\vdots \\ \triangleright X\end{array}}{X}\triangleright\text{E}$ b. $\dfrac{\begin{array}{c}\vdots \\ X \triangleleft\end{array}}{X}\triangleleft\text{E}$

The above may also be expressed as sequent rules:

(41) $\dfrac{\Gamma\ \triangleright X\ Y\ \Delta \Rightarrow Z}{\Gamma\ Y\ \triangleright X\ \Delta \Rightarrow Z}\ [\triangleright\text{Perm}]$ $\qquad$ $\dfrac{\Gamma\ Y\ X \triangleleft\ \Delta \Rightarrow Z}{\Gamma\ X \triangleleft\ Y\ \Delta \Rightarrow Z}\ [\triangleleft\text{Perm}]$

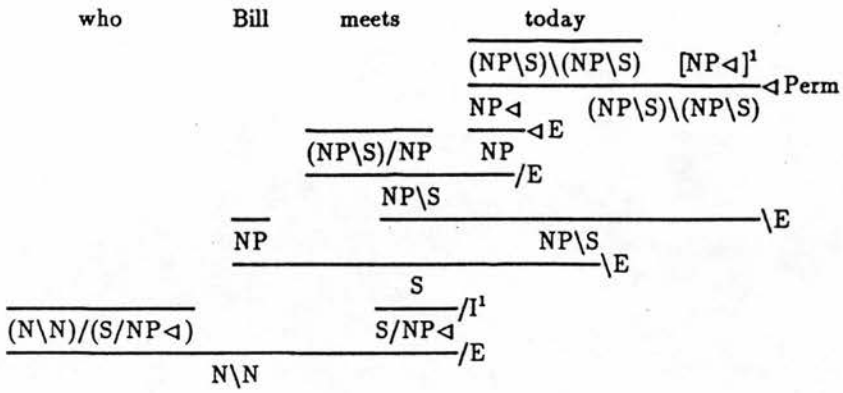
$\dfrac{\Gamma\ \triangleright X\ \Delta \Rightarrow Y}{\Gamma\ X\ \Delta \Rightarrow Y}\ [\triangleright\text{L}]$ $\qquad$ $\dfrac{\Gamma\ X \triangleleft\ \Delta \Rightarrow Y}{\Gamma\ X\ \Delta \Rightarrow Y}\ [\triangleleft\text{L}]$

Note that the structural operation performed by the $\triangleright$Perm and $\triangleleft$Perm rules (in either formulation) is the normal structural operation of permutation.

We may use these modalities in a treatment of non-peripheral extraction, as in "the man who Bill meets today". Moortgat (1988) deals with such constructions by means of the extraction operator $\uparrow$; the type $X \uparrow Y$ is assigned to any string of type X missing a subpart of type Y somewhere within it. So assigning the type $(N\backslash N)/(S\uparrow NP)$ to object relative pronouns licenses extraction from *any* position in the body of a relative clause. We may accomplish the same in the present framework by giving relative pronouns the type $(N\backslash N)/(S/NP \triangleleft)$:

[7] Here, and uniformly in what follows, unary operators bind tighter than binary ones.

(42)



Moortgat (1990b) suggests that the type $X \uparrow Y$ may in fact be definable in terms of modalities similar to the ones above.

We notate the two existential commutability modalities as $>$ (written postfix) and $<$ (written prefix). The type $X>$ (resp. $<X$) is assigned to an item of type X either in its current position or somewhere to the right (resp. left) of its current position. This is achieved by means of the rules >I and <I, which allow the appropriate modality to be added, and the operational rules >Exch and <Exch ('exchange'), which allow types bearing the modality to be moved to any position on the given side:

(43) a. $\dfrac{\vdots\ X}{X>}>\text{I}$ b. $\dfrac{\vdots\ X}{<X}<\text{I}$

(44) a. $\dfrac{\vdots\ X> \quad \vdots\ Y}{Y \quad X>}>\text{Exch}$ b. $\dfrac{\vdots\ Y \quad \vdots\ <X}{<X \quad Y}<\text{Exch}$

The corresponding sequent rules are as follows:

(45) $\dfrac{\Gamma \Rightarrow X>}{\Gamma \Rightarrow X}\ [>\text{R}]$ $\qquad$ $\dfrac{\Gamma \Rightarrow <X}{\Gamma \Rightarrow X}\ [<\text{R}]$

$\dfrac{\Gamma(X>\ Y) \Rightarrow Z}{\Gamma(Y\ X>) \Rightarrow Z}\ [>\text{Exch}]$ $\qquad$ $\dfrac{\Gamma(Y\ <X) \Rightarrow Z}{\Gamma(<X\ Y) \Rightarrow Z}\ [<\text{Exch}]$

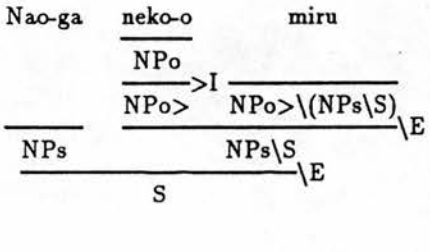
Note that the permutation operation is symmetrical and so the structural operations associated with the >Exch and <Exch rules are trivially inversions of the usual permutation operation.

By way of example a type $X/<Y$ represents elements which are functors over Ys *somewhere* to the right, but not necessarily immediately to the right. In this way we can define the 'long-distance' functors $X//Y$ and $Y\backslash\backslash X$ of Bach (1984). Typically such specifications would need to be made with reference also to the domains within which the word order variation is allowed, e.g. by using the modal treatment of boundedness introduced in Morrill (1989), but the essential idea is as illustrated by Bach's example of SOV and OSV alternation in Japanese:

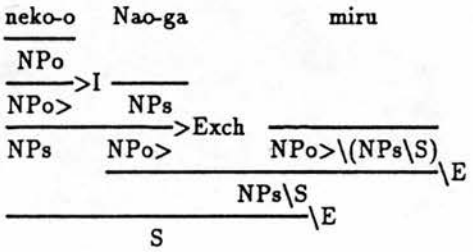
(46) a. Nao-ga neko-o miru
PN-subj cat-obj sees
'Nao sees a cat'

b. neko-o Nao-ga miru
'Nao sees a cat'

(47) a.



b.



4.2 Iterability Modalities

The universal iterability modality is such that the type $X^!$ is assigned to an item from which one or more occurrences of the type X may be derived. This is achieved by means of the rule $^!$Con, which copies any type bearing the operator, and the rule $^!$E, which removes the operator.

(48) a. $\dfrac{\vdots\ X^!}{X^!\quad X^!}\,{}^!\mathrm{Con}$ b. $\dfrac{\vdots\ X^!}{X}\,{}^!\mathrm{E}$

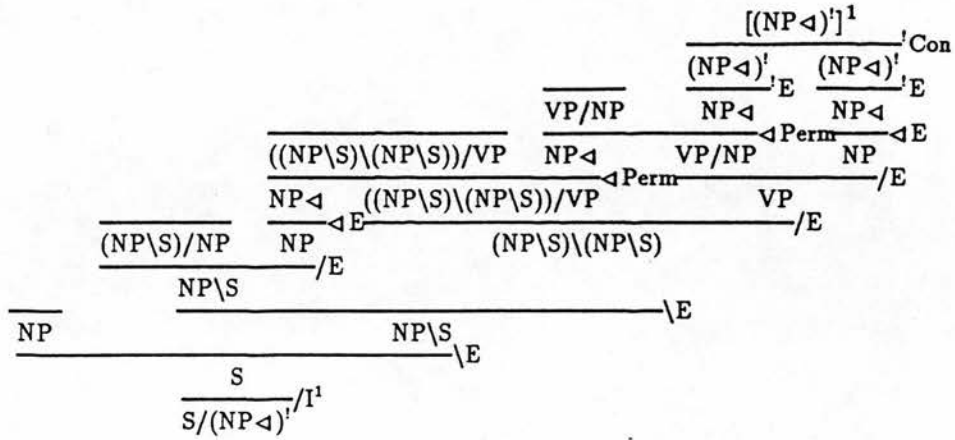
The corresponding sequent rules are:

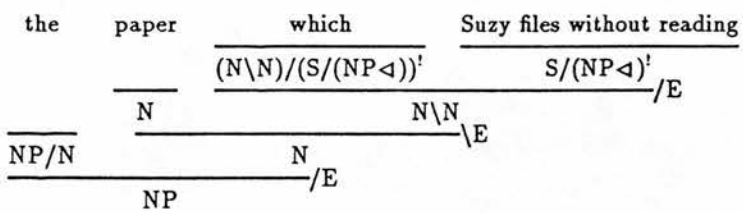
(49) $\dfrac{\Gamma\ X^!\ \Delta \Rightarrow Y}{\Gamma\ X^!\ X^!\ \Delta \Rightarrow Y}\ [{}^!\mathrm{Con}]$ $\qquad$ $\dfrac{\Gamma\ X^!\ \Delta \Rightarrow Y}{\Gamma\ X\ \Delta \Rightarrow Y}\ [{}^!\mathrm{L}]$

The structural operation performed by the $^!$Con rule is contraction. Note that the premise in the sequent formulation of this rule is more complex than the conclusion since it contains at least an extra $^!$, and so the sequent formulation no longer provides a straightforward decision procedure.

We now have the apparatus for a rudimentary treatment of parasitic gaps. Assignment of the type $(N\backslash N)/(S/(NP\triangleleft)^!)$ to a relative pronoun will allow it to fill *any number of* gaps in any positions. (This clearly overgenerates, but allows an illustration of the interaction of iteration and commutation operators.) "The paper which Suzy files without reading" can thus be derived as follows:

(50) Suzy files without reading





The existential iteration modality is such that X^+ represents some non-zero number of items of type X. This is achieved by means of the $^+$I rule, which introduces the operator, and the $^+$Exp ('expansion') rule, which combines identical types bearing it:

(51) a. $\dfrac{\vdots\ X}{X^+}{}^+\mathrm{I}$ b. $\dfrac{\vdots\ X^+ \quad \vdots\ X^+}{X^+}{}^+\mathrm{Exp}$

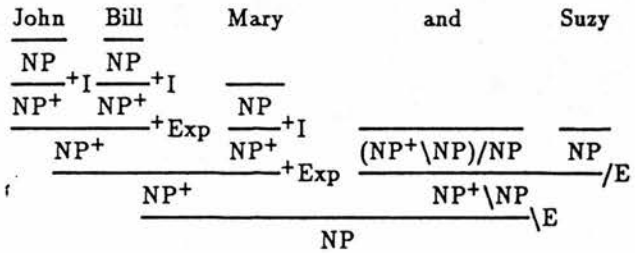
Correspondingly there are the following sequent rules:

(52) $\dfrac{\Gamma \Rightarrow X^+}{\Gamma \Rightarrow X}\ [^+\mathrm{R}]$ $\qquad$ $\dfrac{\Gamma\ X^+\ X^+\ \Delta \Rightarrow Y}{\Gamma\ X^+\ \Delta \Rightarrow Y}\ [^+\mathrm{Exp}]$

The operation associated with the $^+$Exp rule is the inverse of the operation of contraction.

Iterated coordination may be treated by assignment of coordinators to $(X^+\backslash X)/X$:

(53) John Bill Mary and Suzy



4.3 Optionality Modalities

The universal optionality modality is such that $X^{\|}$ is assigned to an item from which zero or one occurrences of the type X may be derived. This is achieved by means of the rules $^{\|}\text{Wkn}_l$ and $^{\|}\text{Wkn}_r$, which remove any type bearing the operator, and a standard universal modality elimination rule $^{\|}\text{E}$:

(54) a. $\dfrac{\vdots \quad \vdots}{\dfrac{X^{\|} \quad Y}{Y}\text{Wkn}_l}$ b. $\dfrac{\vdots \quad \vdots}{\dfrac{Y \quad X^{\|}}{Y}\text{Wkn}_r}$ c. $\dfrac{\vdots}{\dfrac{X^{\|}}{X}{}^{\|}\text{E}}$

The corresponding sequent rules are:

(55) $\dfrac{\Gamma\ X^{\|}\ \Delta \Rightarrow Y}{\Gamma\ \Delta \Rightarrow Y}\ [^{\|}\text{Wkn}]$ $\qquad$ $\dfrac{\Gamma\ X^{\|}\ \Delta \Rightarrow Y}{\Gamma\ X\ \Delta \Rightarrow Y}\ [^{\|}\text{L}]$

Again, the structural operation performed by $^{\|}$Wkn is that of weakening.

This operator would be used in the type of a lexical item whose argument optionally contained a gap (perhaps as in languages with resumptive pronouns or optional clitic doubling). Such constructions are rare in English, but may occur when an adjective is modified by "too": contrast "the lecture was too boring for me to follow" with "the lecture was too boring for me to stay awake". Let predicative adjectives have the type Pred, and let "for ... to" clauses have the type SP. Then we may capture the above data by giving "too" a type such that "too boring" receives the type $\text{Pred}/(\text{S}/\text{NP}^{\|})$:

(56)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\text{Pred}/(\text{SP}/\text{NP}^{\|})}\quad \dfrac{\dfrac{\dfrac{}{\text{SP}/\text{S}} \quad \dfrac{\dfrac{}{\text{NP}} \quad \dfrac{\overline{\text{to follow}}}{(\text{NP}\backslash\text{S})/\text{NP}}\backslash\text{E}}{\text{S}}/\text{E}}{\text{SP}}}{\text{SP}/\text{NP}^{\|}}/\text{I}^1}{\text{Pred}}/\text{E} \quad \dfrac{[\text{NP}^{\|}]^1}{\text{NP}}{}^{\|}\text{E}}{\text{NP}}/\text{E}$$

(57)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\text{Pred}/(\text{SP}/\text{NP}^{\|})}\quad \dfrac{\dfrac{\dfrac{}{\text{SP}/\text{S}} \quad \dfrac{\dfrac{}{\text{NP}} \quad \dfrac{\overline{\text{to stay awake}}}{\text{NP}\backslash\text{S}}\backslash\text{E}}{\text{S}}/\text{E}}{\text{SP}} \quad [\text{NP}^{\|}]^1}{\text{SP}}{}^{\|}\text{Wkn}_r}{\text{SP}/\text{NP}^{\|}}/\text{I}^1}{\text{Pred}}/\text{E}$$

The existential optionality modality in $X^{?}$ indicates zero or one items of type X. The $^{?}$I rule introduces the operator on a non-modal type as with other existential modalities; the $^{?}$Stn ('strengthening') rule creates a modal type from nothing:

(58) a. $\dfrac{\vdots\ X}{X^?}\,{}^?\mathrm{I}$ b. $\dfrac{\quad}{X^?}\,{}^?\mathrm{Stn}$

Expressed as sequent proof rules these are as in (59):

(59) $\dfrac{\Gamma \Rightarrow X^?}{\Gamma \Rightarrow X}\ [{}^?\mathrm{R}]$ $\qquad$ $\dfrac{\Gamma\ \Delta\ \Rightarrow Y}{\Gamma\ X^?\ \Delta\ \Rightarrow Y}\ [{}^?\mathrm{Stn}]$

Once again the ?Stn rule is associated with an operation which is the inverse of weakening. As with the !Con rule, the premise in the sequent version is more complex than the conclusion, which may cause problems for proof search.

By way of example, the optionality of the sentential complement of *belief* may be characterized by assignment to $\mathrm{N}/\mathrm{SP}^?$:

(60) $\dfrac{\dfrac{\text{the}}{\mathrm{NP/N}}\quad \dfrac{\dfrac{\text{belief}}{\mathrm{N/SP^?}}\quad \dfrac{\dfrac{\text{that John lies}}{\mathrm{SP}}}{\mathrm{SP^?}}\,{}^?\mathrm{I}}{\mathrm{N}}/\mathrm{E}}{\mathrm{NP}}/\mathrm{E}$

(61) $\dfrac{\dfrac{\text{the}}{\mathrm{NP/N}}\quad \dfrac{\dfrac{\text{belief}}{\mathrm{N/SP^?}}\quad \dfrac{\quad}{\mathrm{SP^?}}\,{}^?\mathrm{Stn}}{\mathrm{N}}/\mathrm{E}}{\mathrm{NP}}/\mathrm{E}$

5 Conclusion

This paper has introduced a scheme of natural deduction-like proof figures for the Lambek calculus, and has proposed structural modalities which are suitable for the capture of linguistic generalizations in categorial grammar. It remains to refine the semantics and proof theory, and the development of strategies for proof search offers a particular challenge. For the present, we hope that the proposals made can be seen as gaining linguistic practicality in the categorial description of natural language, without losing mathematical elegance.

Ades, A.E. and Steedman, M.J. (1982). On the order of words. *Linguistics and Philosophy 4*, 517–558.

Ajdukiewicz, K. (1935). Die syntaktische Konnexität. *Studia Philosophica 1*, 1–27. Translated as 'Syntactic connexion' in S. McCall (Ed., 1967), *Polish Logic: 1920–1939*, Oxford University Press, Oxford, 207–231.

Anderson, A.R. and Belnap, N.D. (1975). *Entailment, Volume 1.* Princeton University Press, Princeton.

Bach, E. (1984). Some generalizations of categorial grammars. In F. Landman and F. Veltman (Eds), *Varieties of Formal Semantics*, Foris, Dordrecht.

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

van Benthem, J. (1983). *The semantics of variety in categorial grammar.* Report 83–29, Department of Mathematics, Simon Fraser University. Also in In W. Buszkowski, W. Marciszewski and J. van Benthem (Eds), *Categorial Grammar*, Volume 25, Linguistic & Literary Studies in Eastern Europe, John Benjamins, Amsterdam/Philadelphia, 57–84.

van Benthem, J. (1986). Categorial grammar. In *Essays in Logical Semantics*, Volume 8, Studies in Linguistics and Philosophy, D. Reidel, Dordrecht, 123–150.

van Benthem, J. (1987). Categorial grammar and type theory. Prepublication Series 87–07, Institute for Language, Logic and Information, University of Amsterdam.

Curry, H.B. and Feys, R. (1958). *Combinatory Logic, Volume I.* North-Holland, Amsterdam.

Gentzen, G. (1936). On the meanings of the logical constants. In Szabo (Ed., 1969), *The Collected Papers of Gerhard Gentzen*, North Holland, Amsterdam.

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

Girard, J-Y. (1989). Towards a geometry of interaction. *Proceedings of the AMS Conference on Categories, Logic and Computer Science.*

Hepple, M. (1990). Normal form theorem proving for the Lambek calculus. To appear in *Proceedings of COLING 1990.*

Hepple, M. and Morrill, G. (1989). Parsing and derivational equivalence. In *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, UMIST, Manchester.

Howard, W.A. (1969). The formulae-as-types notion of construction. In J.R. Hindley and J.P. Seldin (Eds, 1980), *To H.B. Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press.

König, E. (1989). Parsing as natural deduction. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics.*

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

Moortgat, M. (1988). *Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus*, Foris, Dordrecht.

Moortgat, M. (1990a). Cut elimination and the elimination of spurious ambiguity. In *Proceedings of the Seventh Amsterdam Colloquium*, University of Amsterdam.

Moortgat, M. (1990b). The logic of discontinuous type constructors. To appear in *Proceedings of the Symposium on Discontinuous Constituency*, Institute for Language Technology and Information, University of Tilburg.

Morrill, G. (1989). Intensionality and boundedness. To appear in *Linguistics and Philosophy.*

Ono, H. (1988). Structural rules and a logical hierarchy. To appear in *Proceedings of the Conference on Mathematical Logic and its Applications*, North-Holland, Amsterdam.

Prawitz, D. (1965). *Natural Deduction: a Proof Theoretical Study.* Almqvist and Wiksell, Uppsala.

Wansing, H. (1989). Relevant quasi-deductions, weak implicational logics, and operational semantics. Ms., Institut für Philosophie, Freie Universität Berlin.

Dependency and Constituency in Categorial Grammar

Guy Barry and Martin Pickering

1 Introduction

It has been seen as an advantage of some systems of categorial grammar (Steedman 1987; Moortgat 1988) that they allow the definition of some notion of 'flexible constituency'. This has been argued to allow accounts of syntactic phenomena such as coordination and extraction, as well as other phenomena such as incremental interpretation and intonational structure.

In phrase-structure theories of grammar, category symbols are merely labels for constituents, and so by definition everything that can be given a category must be a constituent. In categorial grammars, on the other hand, the symbols used reflect more general combining properties of strings. To avoid confusion we shall use the word *type* rather than *category* to refer to these symbols. Nevertheless, there is still an implicit assumption that any string that can be given a type is a constituent. We shall refer to this assumption as the *type-constituent correspondence* hypothesis.

In the Lambek calculus, every string can be given a type, so type-constituent correspondence would predict that every string is a constituent. If the notion of constituency has any independent linguistic use, then we cannot maintain type-constituent correspondence within the Lambek calculus. This indicates that we must either abandon the Lambek calculus as a linguistic framework or abandon type-constituent correspondence.

The former approach suggests the use of a more restricted calculus such as Steedman's Combinatory Categorial Grammar (CCG). By means of a particular set of type-combining rules, Steedman attempts to assign types to all and only those strings that can be regarded as constituents. This approach is open to two criticisms. Firstly, the type-combining rules are proposed on the basis of specific linguistic phenomena rather than general principles, so that the set of constituents generated is in some sense arbitrary, and (as we shall argue) leads to some anomalies in linguistic description. This could perhaps however be rectified by a more principled choice of rules. Secondly, though, Steedman's approach has no way of characterizing strings that are not constituents, even though it appears (as we shall also argue) that the characterization of such strings is necessary for the description of certain linguistic phenomena. We shall go into more detail on both these points in section 5.

In this paper we shall investigate the alternative approach of retaining the general Lambek framework, but dropping type-constituent correspondence. Instead we shall argue from the standpoint of dependency, by proposing a definition of dependency within the Lambek calculus, and from it deriving a new notion of constituency, which we shall refer to as *dependency constituency*. We shall argue that the notion of a dependency constituent has applications in the syntactic description of coordinate constructions, as well as in the development of an incremental account of sentence processing. We believe that it has applications in other areas of linguistics such as the syntax of extraction, and intonational structure.

2 Dependencies and Dependency Constituents

2.1 Heads and Dependents

All notions of dependency rely on a primitive notion of *head*; roughly speaking, the head is the element on which other elements depend. For example, consider the following sentence:

(1) Bill talks to Mary.

Three approaches seem plausible. If we assume that heads and dependents are both phrases, then given standard phrase-structure assumptions **talks to Mary** is the head of the sentence and has the single dependent **Bill**, and **talks** is the head of **talks to Mary** and has the single dependent **to Mary**. Alternatively, if we assume that heads are words and dependents are phrases, then **talks** can be taken as the head of the sentence and as having the two dependents **to Mary** and **Bill**. Finally, if we assume that heads and dependents are both words, then **talks** can be taken as the head of the sentence and as having the two dependents **to** and **Bill**, and **to** can be taken as having the single dependent **Mary**.

There is no primitive notion of head in categorial grammar, but the notion of semantic/syntactic *functor* is often taken to be head-like. This raises two questions; should categorial grammar be thought of as having phrasal or lexical heads and dependents, and what is the relationship of functors to 'traditional' (linguistically motivated) heads?

To answer the first point, let us limit our attention for the moment to applicative categorial grammar (**AB**). **AB** can be thought of as having phrasal heads and phrasal dependents if we adopt the following definition (where the semantic type $Y \rightarrow X$ collapses the two syntactic types $X/Y, Y\backslash X$):

Definition 1 *When a string of a functor type $Y \rightarrow X$ combines with a string of an argument type Y to form a string of type X, the functor string is the* phrasal head *and the argument string the* (phrasal) dependent.

Thus in **Bill talks to Mary** of type S, **talks to Mary** of type NP\S is the phrasal head, and **Bill** of type NP is the dependent. Similarly, in **talks to Mary** of type NP\S, **talks** of type (NP\S)/PP is the phrasal head, and **to Mary** of type PP is the dependent.

But we may also regard **AB** as having lexical heads and phrasal dependents if we adopt the following alternative definition:

Definition 2 *When a word of a functor type $Y_1 \rightarrow (\ldots \rightarrow (Y_n \rightarrow X)\ldots)$ combines with strings of argument types $Y_1, \ldots, Y_n$ to form a string of type X, the functor word is the* lexical head *and the argument strings the* (phrasal) dependents.

Thus, in **Bill talks to Mary** of type S, **talks** of type (NP\S)/PP is the lexical head, and **to Mary** of type PP and **Bill** of type NP are the dependents.

In other words, we can derive the notion of a lexical head from that of a phrasal head by a process similar to 'uncurrying' of functions. But it seems more natural to view dependents as phrasal, since functions take expressions of phrasal rather than lexical types as arguments. (This view will be reinforced when we consider the full Lambek calculus below.)

The second point begs the question of what is the traditionally accepted notion of head, about which there is widespread disagreement. We shall not go into the issues here; see Zwicky (1985) and Hudson's (1987) reply to Zwicky for a comprehensive discussion. Hudson gives a great deal of evidence which suggests that, for constructions involving complementation, most syntactically motivated definitions of head in fact coincide with the notion of semantic

functor. However, the reverse appears to be true in modifier constructions, where according to Hudson's criteria for headship (and nearly all linguistic theories) the modified element appears to be the head, whereas on semantic grounds the modifier is standardly assumed to be the functor (as witnessed by the most common categorial type assignments N/N, N\N, (NP\S)\(NP\S) for adjectives, relative clauses and adverbs respectively).

There are three possible solutions to this problem, none of them entirely satisfactory. We could deny the equivalence of head and functor, but this would require us to define a separate theoretical primitive of 'head' independently of the categorial grammar formalism; we shall not pursue this further. Alternatively we could claim that modifiers are heads, although as we shall see in subsection 2.3 this approach leads to an unsatisfactory notion of constituency. Finally we could claim that modified elements are functors, which seems to be supported by the fact that the distinction between modifiers and optional arguments is not always clear.

We shall formalize the above description of modifiers by assuming that modifiers have atomic types, and that modifiable types are functions over zero or more premodifiers and zero or more postmodifiers (cf. HPSG (Pollard and Sag 1987)). We can achieve this by extending the categorial machinery with a 'Kleene star' structural operator, so that X^* means 'zero or more occurrences of X'.[1] We shall classify noun premodifiers as AdnPre, noun postmodifiers as AdnPost, verb premodifiers as AdvPre and verb postmodifiers as AdvPost. Thus lexical nouns will be categorized as

(AdnPre*\N)/AdnPost*,

from which the types N, AdnPre\N, AdnPre\(AdnPre\N), N/AdnPost etc. are derivable. Similarly intransitive verbs will be categorized as

(AdvPre*\(NP\S))/AdvPost*,

transitive verbs as

((AdvPre*\(NP\S))/AdvPost*)/NP,

and so on. Since it would be unwieldy (and uninformative) to write down these complex types in every derivation, we shall usually write only the derived type that is appropriate to the particular derivation. For example, man will be given the type N in the man walks, AdnPre\N in the old man walks, N/AdnPost in the man who I like walks and so on.

2.2 Dependency in the Lambek Calculus

When we try to generalize the ideas of head and dependent developed above for **AB** to the full Lambek calculus **L**, we run into a problem, since the roles of functor and argument can be reversed. Suppose we assume that head-dependent relations are based on the functor-argument relations implicit in lexical type assignments. Then, for instance, John walks consists of a function walks of type NP\S applying to an argument John of type NP, and so walks is the head. But if we assign the higher type S/(NP\S) to John, then John appears to be the head. If the basic meaning of John is represented as **john** in the first case, then its meaning in the second case will be $\lambda f.f$ **john**, where f is a variable of type NP$\rightarrow$S; when John is type-raised, the dependency relations implicit in the lexical assignments are lost. We shall refer to a meaning representation as *dependency-preserving* if it maintains the head-dependent relations derivable from lexical assignments. More formally:

Definition 3 *A lambda-calculus meaning representation is* dependency-preserving *iff it does not involve abstraction of a variable that occurs as a functor within it.*

As pointed out in Morrill, Leslie, Hepple and Barry (1990), there is a direct correspondence between such lambda-terms and derivations in the 'natural deduction' style formulation of **L**. Thus definition 3 is equivalent to the following:

Definition 4 *A derivation in* **L** *is* dependency-preserving *iff it does not discharge an assumption that forms the major premise of an Elimination inference.*

So for example consider the six derivations below, and their associated lambda-terms. The three in (2) are dependency-preserving, and the three in (3) are not:[2]

(2) a.
$$\dfrac{\dfrac{\textbf{the}}{\text{NP/N}} \quad \dfrac{\textbf{dog}}{\text{N}}}{\text{NP}}/\text{E}$$
the dog

b.
$$\dfrac{\dfrac{\dfrac{\textbf{John}}{\text{NP}} \quad \dfrac{\dfrac{\textbf{likes}}{\text{(NP\S)/NP}} \quad [\text{NP}]_1}{\text{NP\S}}/\text{E}}{\text{S}}\backslash\text{E}}{\text{S/NP}}/\text{I}_1$$
λx^{NP}**[likes** x **john]**

c.
$$\dfrac{\dfrac{\dfrac{\textbf{will}}{\text{(NP\S)/VP}} \quad \dfrac{\textbf{see}}{\text{VP/NP}} \quad [\text{NP}]_1}{\text{VP}}/\text{E}}{\text{NP\S}}/\text{E}}{\text{(NP\S)/NP}}/\text{I}_1$$
λx^{NP}**[will (see** x**)]**

(3) a.
$$\dfrac{[\text{NP/N}]_1 \quad \dfrac{\textbf{dog}}{\text{N}} \quad \dfrac{\textbf{runs}}{\text{NP\S}}}{\text{NP}}/\text{E}}{\text{S}}\backslash\text{E}}{\text{(NP/N)\S}}\backslash\text{I}_1$$
$\lambda f^{\text{N}\rightarrow\text{NP}}$**[runs (**$f$ **dog)]**

b.
$$\dfrac{\dfrac{\textbf{that}}{\text{SP/S}} \quad \dfrac{\dfrac{\textbf{Harry}}{\text{NP}} \quad [\text{NP\S}]_1}{\text{S}}\backslash\text{E}}{\text{SP}}/\text{E}}{\text{SP/(NP\S)}}/\text{I}_1$$
$\lambda f^{\text{NP}\rightarrow\text{S}}$**[that (**$f$ **harry)]**

c.
$$\dfrac{\dfrac{\dfrac{[((\text{NP\S})/\text{NP})/\text{NP}]_1 \quad \dfrac{\textbf{Mary}}{\text{NP}} \quad \dfrac{\textbf{John}}{\text{NP}}}{\text{(NP\S)/NP}}/\text{E}}{\text{NP\S}}/\text{E}}{\text{(((NP\S)/NP)/NP)\(NP\S)}}\backslash\text{I}_1$$
$\lambda f^{\text{NP}\rightarrow(\text{NP}\rightarrow(\text{NP}\rightarrow\text{S}))}$**[**$f$ **mary john]**

The term **the dog** is dependency-preserving because it does not involve abstraction, and the terms λx**[likes** x **john]** and λx**[will (see** x**)]** are dependency-preserving because in each case the abstracted variable does not occur as a functor. On the other hand, the terms λf**[runs (**f **dog)]**, λf**[that (**f **harry)]** and λf**[**f **mary john]** are not dependency-preserving, since in each case the abstracted variable is a functor over one or more arguments.

We shall refer to strings with dependency-preserving analyses as *dependency constituents*. More precisely:

Definition 5 *A* dependency constituent *is a string (under a particular reading) whose normal-form derivation in* **L** *is dependency-preserving (equivalently, for which* some *derivation in* **L** *is dependency-preserving).*

(The equivalence of the two definitions is immediate because the normalization process never introduces new functors.) Thus the underlined substrings in (4) below are analysable as dependency constituents, whereas those in (5) are not:

(4) a. The dog runs.

b. John likes Mary.

c. John will see Mary.

(5) a. The dog runs.

b. I think that Harry left.

c. I showed Mary John.

Intuitively, we may think of the underlined substrings in (5) as having 'missing heads'; two words cannot be related because the head of one or both of them is absent from the string.

If we take the traditional notion of constituent to correspond to strings with purely applicative derivations, then the notion of dependency constituent is seen to subsume the traditional notion. The advantage of this notion of constituency is that it allows a degree of flexibility in constituent structure while still requiring that constituents are (in some sense) semantically coherent units. For example, consider:

(6) John thinks that Harry likes Mary.

Here, the substrings that Harry, thinks that Harry and John thinks that Harry are not dependency constituents, but every other substring (including one-word strings and the entire sentence) is, e.g. John thinks, John thinks that, that Harry likes, thinks that Harry likes.

2.3 Consequences of Dependency Constituency

Clearly the choice of head in modifier constructions will make a difference to what strings are regarded as dependency constituents. Consider for example:

(7) the tall man

(8) the man who I saw

If modifiers are regarded as heads, the underlined substring in (7) will be a dependency constituent, but the one in (8) will not be. On the other hand, if modified elements are regarded as heads, the underlined substring in (7) will not be a dependency constituent, but the one in (8) will be. The relevant proofs and lambda-terms follow:

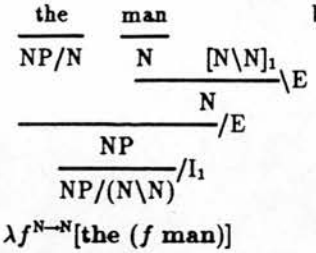
(9) a.
$$\dfrac{\dfrac{\dfrac{\text{the}}{\text{NP/N}} \quad \dfrac{\dfrac{\text{tall}}{\text{N/N}} \quad [\text{N}]_1}{\text{N}}/\text{E}}{\text{NP}}/\text{E}}{\dfrac{\text{NP}}{\text{NP/N}}/\text{I}_1}$$

$\lambda x^{\text{N}}[\text{the (tall } x)]$

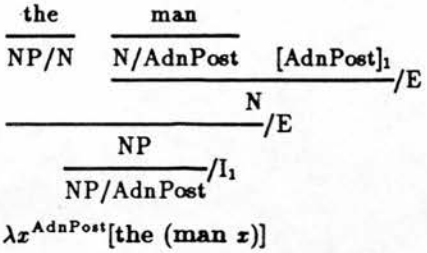
b.
$$\dfrac{\dfrac{\dfrac{\text{the}}{\text{NP/N}} \quad \dfrac{\dfrac{\text{tall}}{\text{AdnPre}} \quad [\text{AdnPre}\backslash\text{N}]_1}{\text{N}}\backslash\text{E}}{\text{NP}}/\text{E}}{\dfrac{\text{NP}}{\text{NP}/(\text{AdnPre}\backslash\text{N})}/\text{I}_1}$$

$\lambda f^{\text{AdnPre}\rightarrow\text{N}}[\text{the } (f \text{ tall})]$

(10) a.



b.

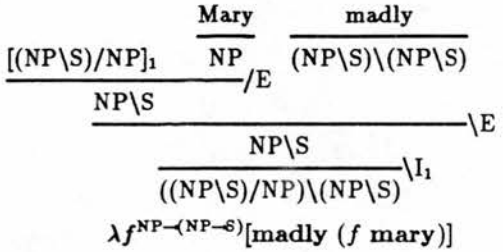


This would seem to support the choice of modified element as head, since there is a clear intuitive connection between **the** and **man**, but not between **the** and **tall**. We shall assume modified elements to be heads for the rest of the paper (though we shall sometimes give alternative analyses where modifiers are assumed to be heads).

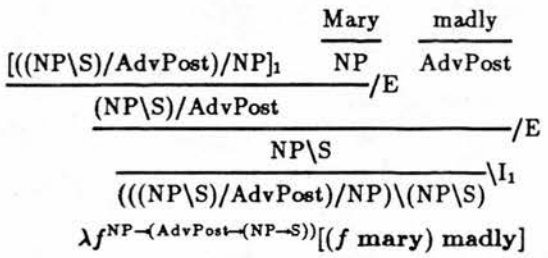
It is worth noting here that the string underlined in (11) cannot be a dependency constituent, whatever the choice of head in modifier constructions:

(11) John loves <u>Mary madly</u>.

(12) a.



b.

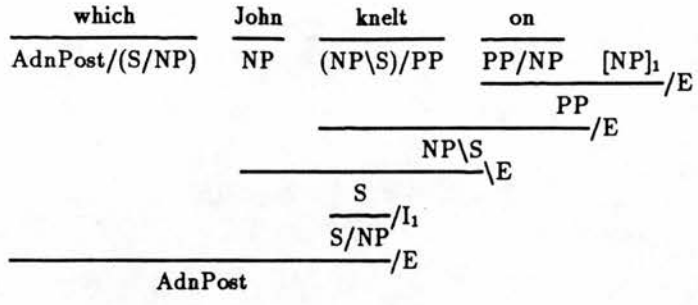


Some interesting consequences of this definition of dependency emerge when we consider constructions involving extraction. Firstly, as one might expect, any relative clause is a dependency constituent, e.g. the underlined fragments of the following two constructions:[3]

(13) the box <u>which John knelt on</u>

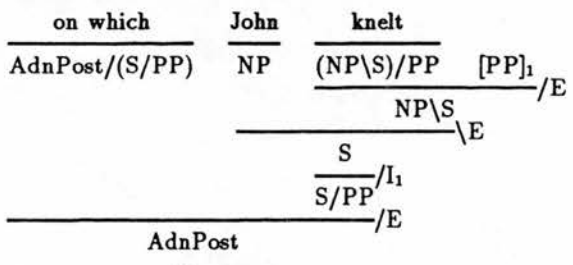
(14) the box <u>on which John knelt</u>

(15)



which (λx^{NP}[**knelt (on** x**) john**])

(16)

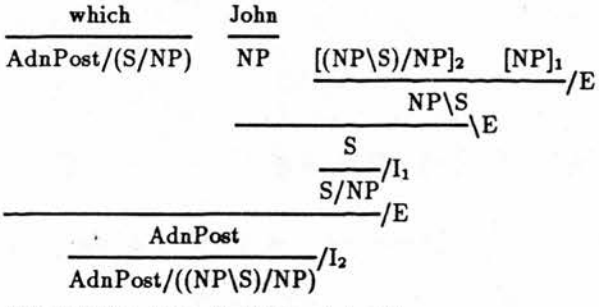


on-which (λx^{PP}[**knelt** x **john**])

On the other hand, some fragments of relative clauses are not dependency constituents, such as the two underlined below:

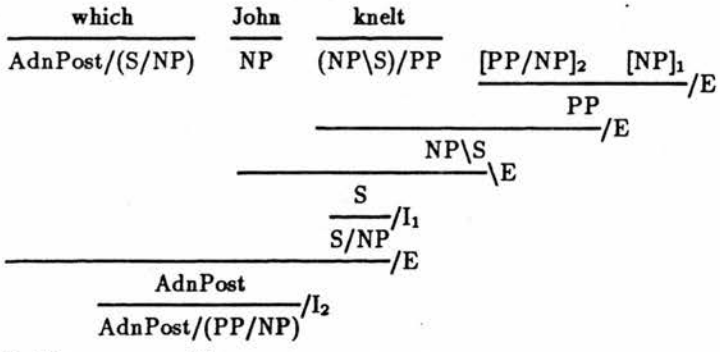
(17) the box <u>which John</u> knelt on
(18) the box <u>which John knelt</u> on

(19)



$\lambda f^{NP\rightarrow(NP\rightarrow S)}$[**which** ($\lambda x^{NP}$[$f$ x **john**])]

(20)



$\lambda g^{NP\rightarrow PP}$[**which** ($\lambda x^{NP}$[**knelt** ($g$ x) **john**])]

By our definition, neither derivation is dependency-preserving, since in each case an abstracted variable acts as a functor ($f^{NP\to(NP\to S)}$ in the first case, $g^{NP\to PP}$ in the second case). But each time the argument of the abstracted functor is not a constant but another abstracted variable, so that the 'headless' element is not a lexical item.

Thus we see that definition 3 enables us to make some subtle distinctions. For example, the underlined phrase in (21) will be analysed as a dependency constituent, whereas that in (22) will not:

(21) the tray <u>on which Mary placed</u> the cake

(22) the tray <u>which Mary placed</u> the cake on

(23)
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{on which}}{\text{AdnPost}/(\text{S}/\text{PP})}\quad\dfrac{\text{Mary}}{\text{NP}}\quad\dfrac{\dfrac{\text{placed}}{((\text{NP}\backslash\text{S})/\text{PP})/\text{NP}}\quad[\text{NP}]_2}{(\text{NP}\backslash\text{S})/\text{PP}}/\text{E}\quad[\text{PP}]_1}{\text{NP}\backslash\text{S}}/\text{E}}{\text{S}}\backslash\text{E}}{\text{S}/\text{PP}}/\text{I}_1}{\text{AdnPost}}/\text{E}}{\text{AdnPost}/\text{NP}}/\text{I}_2$$

λy^{NP}[on-which (λx^{PP}[placed y x mary])]

(24)
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{which}}{\text{AdnPost}/(\text{S}/\text{NP})}\quad\dfrac{\text{Mary}}{\text{NP}}\quad\dfrac{\dfrac{\text{placed}}{((\text{NP}\backslash\text{S})/\text{PP})/\text{NP}}\quad[\text{NP}]_3}{(\text{NP}\backslash\text{S})/\text{PP}}/\text{E}\quad\dfrac{[\text{PP}/\text{NP}]_2\quad[\text{NP}]_1}{\text{PP}}/\text{E}}{\text{NP}\backslash\text{S}}/\text{E}}{\text{S}}\backslash\text{E}}{\text{S}/\text{NP}}/\text{I}_1}{\text{AdnPost}}/\text{E}}{\text{AdnPost}/(\text{PP}/\text{NP})}/\text{I}_2}{(\text{AdnPost}/(\text{PP}/\text{NP}))/\text{NP}}/\text{I}_3$$

$\lambda y^{NP}\lambda g^{NP\to PP}$[which ($\lambda x^{NP}$[placed y (g x) mary])]

In the first case only complete arguments of placed are abstracted, but in the second the variable g is abstracted, which is itself a functor over the abstracted variable x. This seems to capture the intuition that in (21) the relative pronoun on which fills an argument role of the verb placed, whereas in (22) the relative pronoun which fills an argument role not of placed itself but of the prepositional argument of placed (which is outside the string).

To summarize this section, we have attempted to capture the intuitive notion of dependency within the Lambek calculus, and from it derived a flexible notion of constituency. We have thus abandoned the assumption that any string that can be assigned a type must be a constituent. Let us now see how this notion can be applied to problems of coordination and incremental processing.

3 Coordination

3.1 Coordination of Dependency Constituents

A claim often made in favour of flexible categorial grammars is their ability to deal with a large proportion of coordination phenomena by means of a single principle, usually assumed to be as follows:

Hypothesis 1 *Two (or more) strings may be coordinated to give a string of type X iff each has type X.*

For the purposes of this discussion we shall treat the two directions of implication in hypothesis 1 as two separate hypotheses. One direction may be phrased as follows:

Hypothesis 2 *If two (or more) strings can be given the same type X, then they can be coordinated to give a string of type X.*

We shall discuss the converse later (hypothesis 4).

The precise mechanism used to implement hypothesis 2 varies from formulation to formulation (e.g. polymorphic types for conjunctions, syncategorematic rule schemas), but the principle remains the same. In this section we are concerned not with the mechanism for coordination, but with the characterization of what strings can be coordinated. To save space in examples, instead of giving full derivations we shall usually merely bracket each conjunct, and specify the type that must be assigned to each conjunct (and to the coordinate structure) for the derivation to proceed.

If we are working within the full Lambek calculus, hypothesis 2 holds true in a large class of cases. For instance, we can clearly coordinate strings that form standard phrase structure constituents, like (25) below:

(25) John [sang some songs] and [played the piano].
Type of each conjunct: NP\S

We can also deal with cases like (26) and (27), both of which are usually classified under the heading of 'non-constituent' coordination:

(26) John [will buy] and [may eat] the beans.
Type of each conjunct: (NP\S)/NP

(27) John loves [Mary madly] and [Sue passionately].
Type of each conjunct: (((NP\S)/AdvPost*)/NP)\(NP\S)

In our terms, (25) and (26) both involve coordination of dependency constituents, while (27) involves coordination of non-dependency constituents. Although these three examples look very different, in each case the structure of each conjunct is the same, in the sense that each consists of a sequence of words of the same lexical types. Hypothesis 2 will always allow coordination in such cases, since **L** can always give the same type to two strings with the same structure (in this sense).

More interesting are cases where the conjuncts have different structures, such as (28) and (29):

(28) John [plays the piano] and [sings].
Type of each conjunct: NP\S

(29) John [bought] and [may eat] the beans.
Type of each conjunct: (NP\S)/NP

However, the unacceptability of (30) shows that hypothesis 2 overgenerates:[4]

(30) *John loves [Mary madly] and [Sue].
Type of each conjunct: (((NP\S)/AdvPost*)/NP)\(NP\S)

In (28) and (29) both conjuncts are again dependency constituents, but in (30) the first conjunct is not. Note that even if we took modifiers to be heads we could still give the conjuncts in (30) the shared type ((NP\S)/NP)\(NP\S).

In general, the restrictions on coordination of unlike structures seem to be stronger than those on coordination of like structures. For example, (31) appears to be acceptable, but (as noted by Steedman) (32) does not:

(31) [I believe that John] and [Harry thinks that Mary] is a genius.
Type of each conjunct: S/(NP\S)

(32) *[I believe that John] and [Mary] is a genius.
Type of each conjunct: S/(NP\S)

Note once more that both conjuncts in (31) and the first conjunct is (32) are not dependency constituents. Similarly, (33) seems acceptable, but not (34):

(33) [two small] and [three large] oranges
Type of each conjunct: NP/(AdnPre*\N)

(34) *[two small] and [three] oranges
Type of each conjunct: NP/(AdnPre*\N)

Here, the conjuncts in (33) and the first conjunct in (34) are not dependency constituents if modified elements are analysed as heads (although they would be if modifiers were analysed as heads).

This evidence suggests that there is some correlation between coordinability and dependency constituency. In the examples where both conjuncts are dependency constituents, namely (25), (26), (28) and (29), coordination is always possible. This suggests that hypothesis 2 might be replaced by the following weaker claim:

Hypothesis 3 *If two (or more) dependency constituents can be given the same type X, then they can be coordinated to give a string of type X.*[5]

But in the examples where at least one conjunct is not a dependency constituent, coordination appears to be restricted to cases like (27), (31) and (33) where the conjuncts are structurally similar; it is not possible for the other examples (30), (32) and (34). In order to discriminate between these cases we need a more precise notion of 'like structure', which we shall discuss in the next subsection.

So far we have not examined the converse claim to hypothesis 2:

Hypothesis 4 *If two (or more) strings can be coordinated to give a string of type X, then each can be given type X.*

In order to maintain this hypothesis we must have some mechanism for dealing with well-known cases of 'unlike category coordination' such as (35):

(35) John is [lucky] and [a rogue].

Whatever the type of **lucky and a rogue**, it must be assignable to both **lucky** and **a rogue**. If we say that these two items have only the lexical types AdnPre and NP respectively, and that **is** is lexically ambiguous between (NP\S)/AdnPre and (NP\S)/NP, then we are forced to conclude that no type is assignable to **lucky and a rogue**. But if we give **lucky** and **a rogue** an additional shared lexical type, PredP say, and give **is** the single lexical type (NP\S)/PredP, then assigning the type PredP to **lucky and a rogue** is consistent with hypothesis 4. Alternatively we might extend **L** with boolean operators, as suggested in Morrill (1990), to achieve a similar effect.

From hypotheses 3 and 4 it follows that two (or more) dependency constituents can be coordinated to give a string of type X iff each has type X.

3.2 Coordination of Non-Dependency Constituents

Hypothesis 3 puts no restrictions on the internal structure of the two dependency constituents that are coordinated, as we can see by comparing (25) and (28), or (26) and (29). But even when the conjuncts themselves are not dependency constituents, there appears to be no restriction on the internal structure of any substring that is a dependency constituent. For example, (36) and (37) are both as good as (27):

(36) John loves [Mary madly] and [the young woman passionately].

(37) John loves [Mary madly] and [Sue with great ardour].

This would appear to suggest a form of 'constituentwise' coordination, in which the conjuncts are matched dependency constituent for dependency constituent. We may formalize this as follows:

Definition 6 *Let α be any string. Suppose α can be divided into substrings $\langle \alpha_1, \ldots, \alpha_n \rangle$ (where $n \geq 1$) such that each α_i is a dependency constituent, but there is no i such that $\alpha_i \alpha_{i+1}$ is a dependency constituent. (It is easily shown that such a division must exist and be unique.) This is then called a division of α into* maximal dependency constituents (MDCs).

So for instance the division into MDCs of **John bought** is ⟨**John bought**⟩; the division into MDCs of **the young woman passionately** is ⟨**the young woman, passionately**⟩. We define further:

Definition 7 *A set of strings $\alpha, \beta, \ldots$, each with n MDCs $\langle \alpha_1, \ldots, \alpha_n \rangle$, $\langle \beta_1, \ldots, \beta_n \rangle$, ..., are* parallel *if for each i $\alpha_i, \beta_i, \ldots$ may be given the same type.*

A similar notion might be defined by including the product connective; for product-based treatments of coordination see e.g. Wood (1988), Bouma (1989).

Given this definition, we may make the following claim, which subsumes hypothesis 3:

Hypothesis 5 *Two (or more) parallel strings of type X may be coordinated to give a string of type X.*

Hypothesis 5 claims that any two (or more) strings of type X which each have n MDCs of types $\langle X_1, \ldots, X_n \rangle$ can be coordinated to give a string of type X. This covers a large class of examples of classical 'non-constituent coordination' not covered by hypothesis 3, including (27), (31), (33), (36) and (37) (repeated below as (38) to (42)):[6]

(38) John loves [Mary madly] and [Sue passionately].
Type of each conjunct: (((NP\S)/AdvPost*)/NP)\(NP\S)
Types of MDCs of each conjunct: ⟨NP, AdvPost⟩

(39) [I believe that John] and [Harry thinks that Mary] is a genius.
Type of each conjunct: S/(NP\S)
Types of MDCs of each conjunct: ⟨S/S, NP⟩

(40) [two small] and [three large] oranges
Type of each conjunct: NP/(AdnPre*\N)
Types of MDCs of each conjunct: ⟨NP/N, AdnPre⟩

(41) John loves [Mary madly] and [the young woman passionately].
Type of each conjunct: (((NP\S)/AdvPost*)/NP)\(NP\S)
Types of MDCs of each conjunct: ⟨NP, AdvPost⟩

(42) John loves [Mary madly] and [Sue with great ardour].
Type of each conjunct: (((NP\S)/AdvPost*)/NP)\(NP\S)
Types of MDCs of each conjunct: ⟨NP, AdvPost⟩

It also predicts, given some mechanism for dealing with constructions such as (35), that (43) will be grammatical:

(43) John is [pious in church] and [a rogue in the tavern].

The converse of hypothesis 5 (which subsumes hypothesis 4) also appears to be true:

Hypothesis 6 *If two (or more) strings can be coordinated to give a string of type X, then they are parallel strings of type X.*

For example, in none of (30), (32) and (34) (repeated below as (44) to (46)) are the intended conjuncts parallel, since in each case they contain different numbers of MDCs, and indeed coordination is not possible in any of them:

(44) *John loves [Mary madly] and [Sue].
Type of each conjunct: (((NP\S)/AdvPost*)/NP)\(NP\S)
Types of MDCs of first conjunct: ⟨NP, AdvPost⟩
Types of MDCs of second conjunct: ⟨NP⟩

(45) *[I believe that John] and [Mary] is a genius.
Type of each conjunct: S/(NP\S)
Types of MDCs of first conjunct: ⟨S/S, NP⟩
Types of MDCs of second conjunct: ⟨NP⟩

(46) *[two small] and [three] oranges
Type of each conjunct: NP/(AdnPre*\N)
Types of MDCs of first conjunct: ⟨NP/N, AdnPre⟩
Types of MDCs of second conjunct: ⟨NP/N⟩

Note that if we took modifiers to be heads both **two small** and **three** would be analysed as dependency constituents, and hence (46) would be incorrectly predicted to be grammatical.

The present account also rules out examples such as the following two:

(47) ?I talked [about cricket to Edna] and [to Eric about chess].

(48) ?*I gave [John a pencil] and [a pen to Mary].

It might be possible to generate (47) by some relaxation of the definition of parallelism, but there is no obvious general way to generate (48).

To summarize, we may combine hypotheses 5 and 6 into the following principle:

Hypothesis 7 *Two (or more) strings may be coordinated to give a string of type X iff each has type X and they are parallel.*

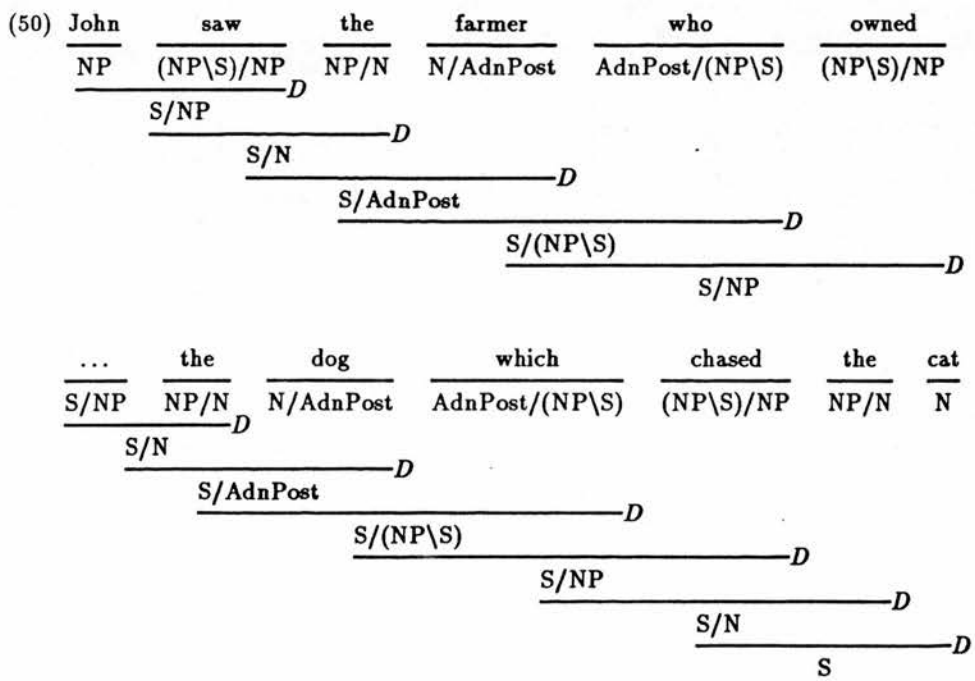
4 Incremental Interpretation

The above discussion of coordination shows that forming a dependency constituent is in some sense a straightforward operation. If this is so, it seems likely that the parser does not encounter any difficulties in forming these constituents, though it is of course not necessary that it does always act this way. This suggests that constructions which always form a single dependency constituent when processed *left-associatively*, that is, word by word from left to right, should not cause processing difficulties. As an example, consider (49) below:

(49) John saw the farmer who owned the dog which chased the cat.

For instance, **John**, **John saw**, **John saw the**, **John saw the farmer**, **John saw the farmer who** are all dependency constituents. Clearly we do not have any difficulty processing this sentence, and it is at least intuitively reasonable that we do process it word by word. This can be taken as an example of an 'incremental' parse (Steedman 1989). Note additionally that it has even intonation, which suggests that there are no points where we encounter momentary or prolonged processing difficulty.

Let us now look at the left-associative derivation for this sentence:



We have marked each combination with D, in order to indicate that they are combinations that form dependency constituents. At no point is there any need to abstract over a functor in the meaning representation. The simplicity of the operations seems to be reflected in the ease we have processing it.

Similar effects occur if we extend the sentence with further relative clauses:

(51) John saw the farmer who owned the dog which chased the cat which followed the mouse which nibbled the cheese.

The sentence does not become obviously harder to process when we extend it in this way. If we do a left-associative parse, we again see that the types generated do not become complex and we never have the need to make any non-dependency combinations. Hence we have support for the hypothesis that the formation of dependency constituents is easy for the parser.

We should note in contrast that a theory that does not allow the parser to form dependency constituents at will may encounter problems with such sentences as these. Standard phrase structure grammars would give (49) the following constituent structure:

(52) [John [saw [the [farmer [who [owned [the [dog [which [chased [the [cat]]]]]]]]]]].

A simple association between grammar and parser would assume that the parser could only form constituents when it reached the very end of this sentence. This is utterly out of keeping with the fact that the sentence, and its extensions, are straightforward to process. Hence presumably such an account has to make use of parsing principles that are not in direct correspondence with the grammar. A processing account based on dependency constituents, on the other hand, does lend itself to a parsimonious relationship between grammar and parser. See Steedman (1989) for related arguments.

So if the calculus allows a combination that forms a dependency constituent, the parser is able to make that combination. But what happens when the left-associative derivation does not solely form dependency constituents? Clearly we are still able to interpret many such sentences:

(53) The tall boy died.

There is no dependency relation between **the** and **tall**, so the left-associative parse does not always form dependency constituents. However we encounter no problem with this sentence. It seems that we can either make a non-dependency association between the first two words, or we wait until we reach **boy**, which is in a dependency relation with both words. In other words, is our method of processing entirely left-associative, or is it restricted to the formation of dependency constituents? We are not going to suggest an answer to this question, but simply discuss both alternatives.

Let us consider a nested construction as defined by Chomsky (1965):

(54) John saw the cat which the dog which the farmer owned chased.

Such constructions are uncontroversially hard to process. This is in sharp contrast to (49), even though they are 'grammatically' of similar complexity. Unlike (49), (54) does not have a left-associative derivation that only forms dependency constituents, and, unlike (53), this does not seem to be a 'local' effect. In (54), the initial substring **the cat which** forms a dependency constituent, but not **the cat which the**, so that **the dog which** must be parsed separately from **the cat which**. Similarly **the farmer** must be parsed separately from **the cat which** and **the dog which**. At the point before **owned**, then, we find that we have three unrelated dependency

constituents, **the cat which, the dog which** and **the farmer**. Intonationally as well, we can distinguish units **John saw the cat which, the dog which** and **the farmer**, which are the same as the three dependency constituents.

The point is even clearer when we contrast the effect found by recursing the construction in (51) above and in (55) below:

(55) John saw the girl who the rat which the cat which the dog which the farmer owned chased scratched bit.

(51) is no more complex to parse than (49), and remains parsable if we recurse the construction still further. The same cannot be said of (55), which is virtually incomprehensible. Similarly, at the point before **owned** in (55), we have four unrelated dependency constituents. Each extension of (54) increases the number of unrelated dependency constituents, in contrast to extensions of (49).

We could parse nested constructions like (54) purely left-associatively, in which case we are forced to allow the parser to make combinations that are not dependency-preserving (here notated by *N*):

(56)

John	saw	the	cat	which	the	dog
NP	(NP\S)/NP	NP/N	N/AdnPost	AdnPost/(S/NP)	NP/N	N/AdnPost
	S/NP *D*					
	S/N *D*					
	S/AdnPost *D*					
	S/(S/NP) *N*					
	S/((NP\S)/NP)/N *D*					
	S/((NP\S)/NP)/AdnPost					

...	which	the	farmer	owned	chased
S/((NP\S)/NP)/AdnPost	AdnPost/(S/NP) *D*	NP/N	N	(NP\S)/NP	(NP\S)/NP
S/((NP\S)/NP)/(S/NP) *N*					
S/((NP\S)/NP)/((NP\S)/NP)/N *D*					
S/((NP\S)/NP)/((NP\S)/NP) *D*					
S/((NP\S)/NP) *D*					
S *D*					

We can see the complexity of the syntactic types after these combinations; there is a strong contrast with the simple types found in the derivation of (49) above. The lambda-terms become similarly complex. For instance, the lambda-term for **John saw the cat which** is

$$\lambda f^{\mathrm{NP}\rightarrow\mathrm{S}}[\mathbf{saw}\ (\mathbf{the}\ (\mathbf{cat}\ (\mathbf{which}\ f)))\ \mathbf{john}],$$

but the term for **John saw the cat which the** is

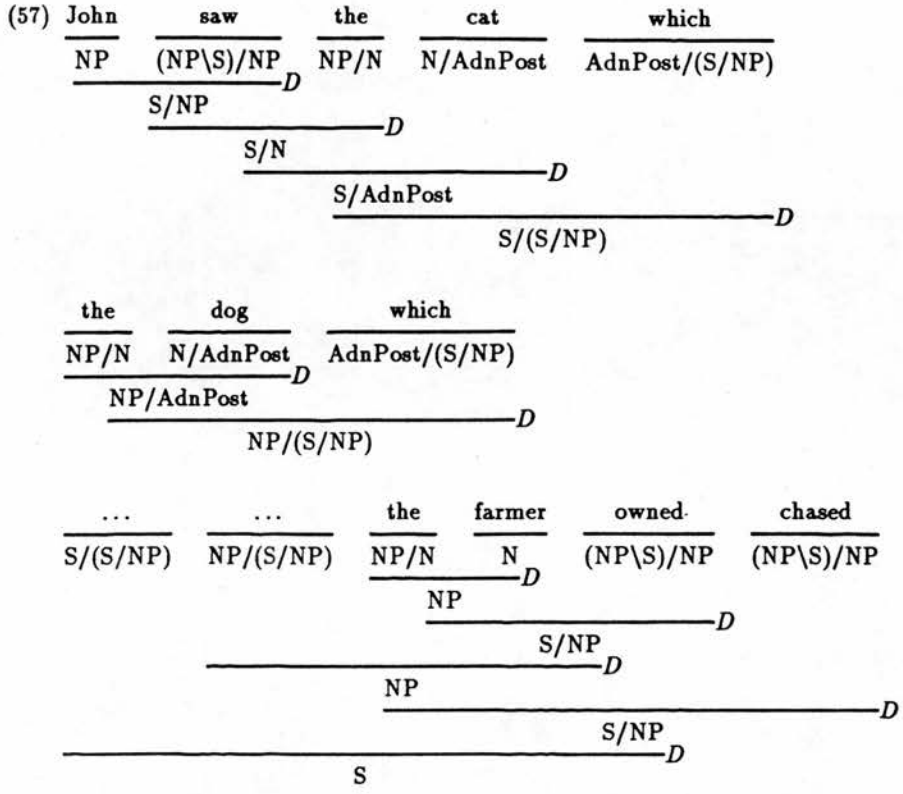
$$\lambda y^{\mathrm{N}}\lambda g^{\mathrm{NP}\rightarrow(\mathrm{NP}\rightarrow\mathrm{S})}[\mathbf{saw}\ (\mathbf{the}\ (\mathbf{cat}\ (\mathbf{which}\ (\lambda x^{\mathrm{NP}}[g\ x\ (\mathbf{the}\ y)])))))\ \mathbf{john}].$$

In the second lambda-term, the abstracted variable g acts as a functor, unlike f in the first lambda-term.

It is also possible that the parser does not make these non-dependency combinations at all, but instead uses some device like a stack to store dependency constituents (see Ades and

Steedman 1982). This means that our derivation is no longer fully left-associative. Here we represent the maximally incremental derivation containing only dependency combinations:

(57)



Another example of the contrast between nested and non-nested constructions can be seen in (21) and (22), repeated below as (58) and (59):

(58) the tray on which Mary placed the cake

(59) the tray which Mary placed the cake on

At the verb **put**, we have a single dependency constituent in (58) but not in (59). As pointed out in Pickering and Barry (1989, 1990), we can recurse each construction on the object noun phrase, for instance in the sentences below:

(60) John found the box in which I put the tray on which Mary placed the cake.

(61) John found the box which I put the tray which Mary placed the cake on in.

(61) is very hard to process, while (60) is easy to process. Our account predicts this directly, because (61) has three unrelated dependency constituents, **John found the box which, I put the tray which** and **Mary placed the cake**, at the point before the word **on**. This is not the case in (60), because each time we reach an embedded verb, we can form a single dependency constituent from all of the preceding material. In (61) we cannot form a single dependency constituent when we reach each embedded verb. Pickering and Barry contrast a dependency based account with an account making use of empty categories, and, given that the construction in (60) can be recursed, show that the latter account makes the wrong predictions for sentences like (60).

We have demonstrated that an account of sentence processing based on the construction of dependency constituents makes reasonable predictions about processing complexity. We

see this as an explanation of Chomsky's (1965) claim that nested constructions are hard to process, and hence why they are so rare in natural language use. It seems therefore that the notion of a dependency constituent is useful both in explaining linguistic data such as coordination, and explaining facts about sentence processing: combining elements to form a dependency constituent is easy, but combining elements to form a non-dependency constituent is hard and not automatic.

5 Comparisons and Conclusions

5.1 Comparisons with Combinatory Categorial Grammar

We have argued that the assumption of type-constituent correspondence within the Lambek calculus must be dropped if the notion of constituency is to retain any linguistic relevance, and dealt with this by defining the notion of dependency constituency. Let us contrast this notion of constituency with the CCG notion of constituency (Steedman 1987).

As mentioned in the Introduction, CCG uses a particular set of rules, known as *combinatory rules*, to assign types to all and only those strings that the theory regards as constituents. Steedman uses a combination of **L**-valid and **L**-invalid rules, but for the purposes of this discussion we shall restrict our attention to the former, since the **L**-invalid rules are highly restricted in their applicability and might reasonably be regarded as peripheral. Thus we are viewing CCG as a subset of **L**.

CCG puts two restrictions on possible type combinations. First, it restricts them to those that can be achieved by means of a particular set of binary and unary rules, including the following:

(62) Forward application (indexed $\rangle$)
$X/Y \;\; Y \Rightarrow X$

Backward application (indexed $\langle$)
$Y \;\; Y\backslash X \Rightarrow X$

Forward composition (indexed $\rangle\mathbf{B}$)
$X/Y \;\; Y/Z \Rightarrow X/Z$

Forward type-raising (indexed $\rangle\mathbf{T}$)
$X \Rightarrow Y/(X\backslash Y)$

However, CCG does not include other **L**-valid inferences such as forward division ($X/Y \Rightarrow (X/Z)/(Y/Z)$) or commutation ($(Y\backslash X)/Z \Rightarrow Y\backslash(X/Z)$). (The choice of rules is made on the basis of specific linguistic phenomena, so that there is no straightforward characterization of which rules are included and which are not.) Secondly, CCG puts type-specific restrictions on the use of the above rules. For instance, there is a stipulation that the use of the forward type-raising rule is restricted to subjects:

(63) NP $\Rightarrow$ S/(NP\S)

Although there is a considerable overlap between CCG constituents and dependency constituents, the existence of type-raising in CCG (which is not dependency-preserving by our definition) generates some CCG constituents that are not dependency constituents. For instance, let us return to our initial examples in section 2 (repeated here as (64) and (65)):

(64) a. The dog runs.

b. John likes Mary.

c. John will see Mary.

(65) a. The dog runs.

b. I think that Harry left.

c. I showed Mary John.

Recall that the underlined fragments in (64) were analysed as dependency constituents, but not those in (65). Steedman's analysis can give a type to the fragments in (64a–c) and (65b), but not to those in (65a) or (65c):

(66) a.
$$\dfrac{\dfrac{\text{the}}{\text{NP/N}}\quad\dfrac{\text{dog}}{\text{N}}}{\text{NP}}\rangle$$

b.
$$\dfrac{\dfrac{\dfrac{\text{John}}{\text{NP}}}{\text{S/(NP\S)}}\rangle\mathbf{T}\quad\dfrac{\text{likes}}{\text{(NP\S)/NP}}}{\text{S/NP}}\rangle\mathbf{B}$$

c.
$$\dfrac{\dfrac{\text{will}}{\text{(NP\S)/VP}}\quad\dfrac{\text{see}}{\text{VP/NP}}}{\text{(NP\S)/NP}}\rangle\mathbf{B}$$

(67) a.
$$\dfrac{\dfrac{\text{dog}}{\text{N}}\quad\dfrac{\text{runs}}{\text{NP\S}}}{}***$$

b.
$$\dfrac{\dfrac{\text{that}}{\text{SP/S}}\quad\dfrac{\dfrac{\text{Harry}}{\text{NP}}}{\text{S/(NP\S)}}\rangle\mathbf{T}}{\text{SP/(NP\S)}}\rangle\mathbf{B}$$

c.
$$\dfrac{\dfrac{\text{Mary}}{\text{NP}}\quad\dfrac{\text{John}}{\text{NP}}}{}***$$

The only point of disagreement here is **that Harry** in (65b), which is a CCG constituent but not a dependency constituent. It would however not be a CCG constituent if the non-dependency-preserving operation of type-raising were absent from the grammar. (Note that removing type-raising from the above set of rules would also stop **John likes** in (64b) from being a constituent, but this could be achieved by means of the dependency-preserving operation of commutation on **likes**.) The above account forces Steedman to analyse **I believe that John** as a constituent of type S/(NP\S), and hence to allow the forbidden coordination in (32) (repeated here as (68)):

(68) *[I believe that John] and [Mary] is a genius.

It is also possible for dependency constituents not to be CCG constituents. For example, Steedman treats modifiers as functors, but forbids modified elements to type-raise over them, in order to account for their status as islands, as in (69):

(69) *Beans, I met a man who likes.

A man **who likes** will be analysed as a dependency constituent irrespective of whether modifiers or modified elements are assumed to be heads. We therefore cannot expect dependency constituents to give us a complete account of island constraints. However, the absence of type-raising over modifiers and of commutation means that it cannot be analysed as a CCG constituent:

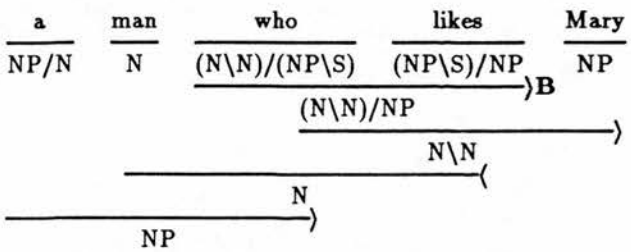
(70)
$$\dfrac{\dfrac{\text{a}}{\text{NP/N}}\quad\dfrac{\text{man}}{\text{N}}\quad\dfrac{\text{who}}{\text{(N\N)/(NP\S)}}\quad\dfrac{\text{likes}}{\text{(NP\S)/NP}}}{\text{(N\N)/NP}}\rangle\mathbf{B}$$

Steedman's attempt to capture island constraints merely by blocking derivations such as the above makes incorrect predictions about coordination and processing of such strings. For instance, it rules out coordinations such as the following:

(71) Most people like, but I know a man who hates, sonatas by Mozart.

This is allowed in our account because the conjuncts are both dependency constituents of type S/NP. More generally, Steedman's account always forbids a noun to combine with an incomplete relative clause. This has serious implications for processing, because it predicts that a noun cannot be combined with a following relative clause until the entire relative clause is processed. For instance, the maximally incremental derivation of **a man who likes Mary** in CCG is as follows:

(72)



Using arguments analogous to those in the previous section, this would incorrectly predict that multiple relative clause constructions such as (51) (repeated here as (73)) should *always* become difficult to process at some point, irrespective of the structure of the relative clauses themselves:

(73) I saw the farmer who owned the dog which chased the cat which followed the mouse which nibbled the cheese.

Steedman's account allows some combinations (e.g. **who owned the**), but the number of elements that have to be remembered independently still increases when the sentence is extended.

Even if the CCG rules were modified to generate all and only dependency constituents, it would still be impossible to account for phenomena that involved the formation of non-dependency constituents, such as the coordination in (27) (repeated here as (74)):

(74) John loves [Mary madly] and [Sue passionately].

The only way to generate such constructions under an assumption of type-constituent correspondence is to allow non-dependency-preserving rules. For example, Dowty (1988) deals with such constructions by using combinatory rules of backward composition and backward type-raising (he assumes the latter to be lexical):

(75) Backward composition (indexed $\langle \mathbf{B}$)
$Z\backslash Y \quad Y\backslash X \Rightarrow Z\backslash X$
Backward type-raising (indexed $\langle \mathbf{T}$)
$X \Rightarrow (Y/X)\backslash Y$

Although both rules are **L**-valid, only backward composition is dependency-preserving. Thus the conjuncts in (74) are constituents under Dowty's analysis, but not dependency constituents:[7]

[7] Here we follow Dowty in taking modifiers to be heads, but a similar argument can be constructed if modified elements are taken to be heads.

(76)
$$\dfrac{\dfrac{\dfrac{\textbf{Mary}}{\text{NP}}}{\dfrac{}{((\text{NP}\backslash\text{S})/\text{NP})\backslash(\text{NP}\backslash\text{S})}\langle\mathbf{T}} \quad \dfrac{\textbf{madly}}{(\text{NP}\backslash\text{S})\backslash(\text{NP}\backslash\text{S})}}{((\text{NP}\backslash\text{S})/\text{NP})\backslash(\text{NP}\backslash\text{S})}\langle\mathbf{B}$$

This means that Dowty is unable to prevent the forbidden coordination in (30) (repeated here as (77)), since both **Mary madly** and **Sue** can be analysed as constituents of type $((\text{NP}\backslash\text{S})/\text{NP})\backslash(\text{NP}\backslash\text{S})$:

(77) *John loves [Mary madly] and [Sue].

5.2 Further Applications

We would not be surprised if the notion of dependency constituency has other linguistic applications. Steedman has argued that flexible categorial grammars allow considerable generalizations between the constituents that are needed to deal with coordination, incremental interpretation, extraction and intonational structure. We agree with this view, except that we claim that the relevant notion is dependency constituency rather than CCG constituency. This paper has covered only coordination and incremental interpretation, but there are a number of possible applications to other linguistic phenomena. For example, the unacceptability of (78) suggests that extraction is subject to the restriction that the non-extracted part of the string must be a dependency constituent:[8]

(78) *How many boys do you think in ten like football?

However, as we have seen above, this condition is not sufficient in itself to capture all island constraints.

Dependency constituency also appears to be relevant to an account of intonational structure. Steedman (1990) shows that the range of possible intonations is greater than those that are most obviously allowable by traditional phrase structure, but that there are still many strings which do not form possible intonational units. Again we believe that the range of possible intonational units can be derived from the notion of dependency constituency.

In summary, we believe that the approach taken in this paper has the following two advantages. Firstly, the notion of constituency is defined in a principled fashion from the notion of dependency. Secondly, it is possible to give types to strings that are not constituents. This appears to be theoretically elegant and also to explain a considerable range of data.

Acknowledgements

We would like to thank Robin Cooper, Elisabet Engdahl, Mark Hepple, Janne Johannessen and Glyn Morrill for their comments and suggestions. We acknowledge the support of SERC Research Studentship 88306971 and ESRC Research Studentship C00428722002.

References

Ades, A.E. and Steedman, M.J. (1982). On the order of words. *Linguistics and Philosophy* *4*, 517–558.

[8] Apparent counterexamples such as Which boy do you know that plays football? are presumably examples of extraposition, which will have to be handled separately.

Bouma, G. (1989). Efficient processing of flexible categorial grammar. In *Proceedings of the Fourth Conference of the European Chapter of the ACL*, UMIST, Manchester.

Chomsky, N. (1965). *Aspects of the Theory of Syntax.* MIT Press, Cambridge, Massachusetts.

Dowty, D. (1988). Type raising, functional composition, and non-constituent conjunction. In R. Oehrle, E. Bach and D. Wheeler (Eds), *Categorial Grammars and Natural Language Structures*, D. Reidel, Dordrecht.

Hudson, R.A. (1987). Zwicky on heads. *Journal of Linguistics 23*, 109–132.

Moortgat, M. (1988). *Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus.* Foris, Dordrecht.

Morrill, G. (1990). Grammar and logical types. This volume.

Morrill, G., Leslie, N., Hepple, M. and Barry, G. (1990). Categorial deductions and structural operations. This volume.

Pickering, M. and Barry, G. (1989). Processing extractions without gaps. Research Paper EUCCS/RP–36, Centre for Cognitive Science, University of Edinburgh.

Pickering, M. and Barry, G. (1990). Sentence processing without empty categories. Ms., Centre for Cognitive Science, University of Edinburgh.

Pollard, C. and Sag, I. (1987). *An Information-Based Approach to Syntax and Semantics: Volume 1 Fundamentals.* CSLI Lecture Notes, Number 13, Stanford.

Steedman, M.J. (1987). Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory 5*, 403–439.

Steedman, M.J. (1989). Grammar, interpretation and processing from the lexicon. In W. Marslen-Wilson (Ed.), *Lexical Representation and Process*, MIT Press, Cambridge, Massachusetts.

Steedman, M.J. (1990). Syntax and intonational structure in a combinatory grammar. To appear in G.T.M. Altmann (Ed.), *Cognitive Models of Speech Processing: Psycholinguistic and Computational Perspectives*, MIT Press, Cambridge, Massachusetts.

Wood, M. (1988). *A Categorial Syntax for Coordinate Constructions.* PhD thesis, University College London. Available as Technical Report UMCS–89–2–1, Department of Computer Science, University of Manchester.

Zwicky, A.M. (1985). Heads. *Journal of Linguistics 21*, 1–29.

PROOF FIGURES AND STRUCTURAL OPERATORS FOR CATEGORIAL GRAMMAR*

Guy Barry, Mark Hepple†, Neil Leslie and Glyn Morrill‡
Centre for Cognitive Science, University of Edinburgh
2 Buccleuch Place, Edinburgh EH8 9LW, Scotland
guy@cogsci.ed.ac.uk, mrh@cl.cam.ac.uk,
neil@cogsci.ed.ac.uk, Glyn.Morrill@let.ruu.nl

ABSTRACT

Use of Lambek's (1958) categorial grammar for linguistic work has generally been rather limited. There appear to be two main reasons for this: the notations most commonly used can sometimes obscure the structure of proofs and fail to clearly convey linguistic structure, and the calculus as it stands is apparently not powerful enough to describe many phenomena encountered in natural language.

In this paper we suggest ways of dealing with both these deficiencies. Firstly, we reformulate Lambek's system using proof figures based on the 'natural deduction' notation commonly used for derivations in logic, and discuss some of the related proof-theory. Natural deduction is generally regarded as the most economical and comprehensible system for working on proofs by hand, and we suggest that the same advantages hold for a similar presentation of categorial derivations. Secondly, we introduce devices called *structural modalities*, based on the structural rules found in logic, for the characterization of commutation, iteration and optionality. This permits the description of linguistic phenomena which Lambek's system does not capture with the desired sensitivity and generality.

LAMBEK CATEGORIAL GRAMMAR

PRELIMINARIES

Categorial grammar is an approach to language description in which the combination of expressions is governed not by specific linguistic rules but by general logical inference mechanisms. The point of departure can be seen as Frege's position that there are certain 'complete expressions' which are the primary bearers of meaning, and that the meanings of 'incomplete expressions' (including words) are derivative, being their contribution to the meanings of the expressions in which they occur. We suppose that linguistic objects have (at least) two components, form (syntactic) and meaning (semantic). We refer to sets of such objects as *categories*, which are indexed by *types*, and stipulate that all complete expressions belong to categories indexed by *primitive types*. We then recursively classify incomplete expressions according to the means by which they combine (syntactically and semantically) with other expressions.

In the 'syntactic calculus' of Lambek (1958) (variously known as *Lambek categorial grammar*, *Lambek calculus*, or L), expressions are classified by means of a set of *bidirectional types* as defined in (1).

(1) a. If X is a primitive type then X is a type.
b. If X and Y are types then X/Y and $Y\backslash X$ are types.

X/Y (resp. $Y\backslash X$) is the type of incomplete expressions that syntactically combine with a following (resp. preceding) expression of type Y to form an expression of type X, and semantically are functions from meanings of type Y to meanings of type X.

Let us assume complete expressions to be sentences (indexed by the primitive type S), noun phrases (NP), common nouns (N), and non-finite verb phrases (VP). By the above definitions, we may assign types to words as follows:

(2)

John, Mary, Suzy	:= NP
man, paper	:= N
the	:= NP/N
likes, read	:= (NP\S)/NP
quickly	:= (NP\S)\(NP\S)
without	:= ((NP\S)\(NP\S))/VP
understanding	:= VP/NP

We represent the form of a word by printing it in italics, and its meaning by the same word in boldface. For instance, the form of the word "man" will be represented as *man* and its meaning as **man**.

PROOF FIGURES

We shall present the rules of L by means of *proof figures*, based on Prawitz' (1965) systems of 'natural deduction'. Natural deduction was developed by Gentzen (1936) to reflect the natural process of mathematical reasoning in which one uses a number of *inference rules* to justify a single proposition, the *conclusion*, on the basis of having justifications of a number of propositions, called *assumptions*. During

*We would like to thank Robin Cooper, Martin Pickering and Pete Whitelock for comments and discussion relating to this work. The authors were respectively supported by SERC Research Studentship 88306971; ESRC Research Studentship C00428722003; ESPRIT Project 393 and Cognitive Science/HCI Research Initiative 89/CS01 and 89/CS25; SERC Postdoctoral Fellowship B/ITF/206.

†Now at University of Cambridge Computer Laboratory, New Museums Site, Pembroke Street, Cambridge CB2 3QG, England.

‡Now at OTS, Trans 10, 3512 JK Utrecht, Netherlands.

a proof one may *temporarily* make a new assumption if one of the rules licenses the subsequent withdrawal of this assumption. The rule is said to *discharge* the assumption. The conclusion is said to *depend* on the undischarged assumptions, which are called the *hypotheses* of the proof.

A proof is usually represented as a tree with the assumptions as leaves and the conclusion at the root. Finding a proof is then seen as the task of filling this tree in, and the inference rules as operations on the partially completed tree. One can write the inference rules out as such operations, but as these are rather unwieldy it is more usual to present the rules in a more compact form as operations from a set of subproofs (the *premises*) to a conclusion, as follows (where $m \geq 1$ and $n \geq 0$):

(3)
$$\frac{\begin{array}{ccccccc} \vdots & & & & [Y_1]^i & & [Y_n]^i \\ X_1 & \cdots & X_{m-n} & & X_{m-n+1} & \cdots & X_m \end{array}}{Z} R^i$$

This states that a proof of Z can be obtained from proofs of $X_1, \ldots, X_m$ by discharging appropriate occurrences of assumptions $Y_1, \ldots, Y_n$. The use of square brackets around an assumption indicates its discharge. R is the name of the rule, and the index i is included to disambiguate proofs, since there may be an uncertainty as to which rule has discharged which assumption.

As propositions are represented by formulas in logic, so linguistic categories are represented by type formulas in **L**. The left-to-right order of types indicates the order in which the forms of subexpressions are to be concatenated to give a composite expression derived by the proof. Thus we must take note of the order and place of occurrence of the premises of the rules in the proof figures for **L**. There is also a problem with the presentation of the rules in the compact notation as some of the rules will be written *as if* they had a number of conclusions, as follows:

(4)
$$\frac{\begin{array}{ccc} \vdots & & \vdots \\ X_1 & \cdots & X_m \end{array}}{\begin{array}{ccc} Y_1 & \cdots & Y_n \end{array}} R$$

This rule should be seen as a shorthand for:

(5)
$$\frac{\begin{array}{ccc} \vdots & & \vdots \\ X_1 & \cdots & X_m \end{array}}{\begin{array}{ccc} Y_1 & \cdots & Y_n \end{array}} R \qquad \begin{array}{ccc} \vdots & & \vdots \end{array}$$
$$\frac{}{Z}$$

If the rules are viewed in this way it will be seen that they do not violate the single conclusion nature of the figures.

As with standard natural deduction, for each connective there is an *elimination* rule which states how a type containing that connective may be consumed, and an *introduction* rule which states how a type containing that connective may be derived. The elimination rule for / states that a proof of type X/Y followed by a proof of type Y yields a proof of type X. Similarly the elimination rule for \ states that a proof of type $Y\backslash X$ preceded by a proof of type Y yields a proof of type X. Using the notation above, we may write these rules as follows:

(6) a.
$$\frac{\begin{array}{cc} \vdots & \vdots \\ X/Y & Y \end{array}}{X} /\mathrm{E}$$
b.
$$\frac{\begin{array}{cc} \vdots & \vdots \\ Y & Y\backslash X \end{array}}{X} \backslash\mathrm{E}$$

We shall give a semantics for this calculus in the same style as the traditional functional semantics for intuitionistic logic (Troelstra 1969; Howard 1980). In the two rules above, the meaning of the composite expression (of type X) is given by the functional application of the meaning of the *functor* expression (i.e. the one of type X/Y or $Y\backslash X$) to the meaning of the *argument* expression (i.e. the one of type Y). We represent function application by juxtaposition, so that **likes John** means likes applied to **John**.

Using the rules /E and \E, we may derive "Mary likes John" as a sentence as follows:

(7)
$$\dfrac{\dfrac{\text{Mary}}{\text{NP}} \quad \dfrac{\dfrac{\text{likes}}{\text{(NP\S)/NP}} \quad \dfrac{\text{John}}{\text{NP}}}{\text{NP\S}}\,/\mathrm{E}}{\text{S}}\,\backslash\mathrm{E}$$

The meaning of the sentence is read off the proof by interpreting the /E and \E inferences as function application, giving the following:

(8) **(likes John) Mary**

The introduction rule for / states that where the rightmost assumption in a proof of the type X is of type Y, that assumption may be discharged to give a proof of the type X/Y. Similarly, the introduction rule for \ states that where the leftmost assumption in a proof of the type X is of type Y, that assumption may be discharged to give a proof of the type $Y\backslash X$. Using the notation above, we may write these rules as follows:

(9) a.
$$\frac{\begin{array}{c} [Y]^i \\ \vdots \\ X \end{array}}{X/Y} /\mathrm{I}^i$$
b.
$$\frac{\begin{array}{c} [Y]^i \\ \vdots \\ X \end{array}}{Y\backslash X} \backslash\mathrm{I}^i$$

Note however that this notation does not embody the conditions that have been stated, namely that in /I Y is the rightmost undischarged assumption in the proof of X, and in \I Y is the leftmost undischarged assumption in the proof of X. In addition, **L** carries the condition that in both /I and \I the sole assumption in a proof cannot be withdrawn, so that no types are assigned to the empty string.

In the introduction rules, the meaning of the result is given by lambda-abstraction over the meaning of the discharged assumption, which can be represented by a variable of the appropriate type. The relationship between lambda-abstraction and function application is given by the law of *β-equality* in (10),

where $\alpha[\beta|y]$ means 'α with β substituted for y'. (See Hindley and Seldin 1986 for a full exposition of the typed lambda-calculus.)

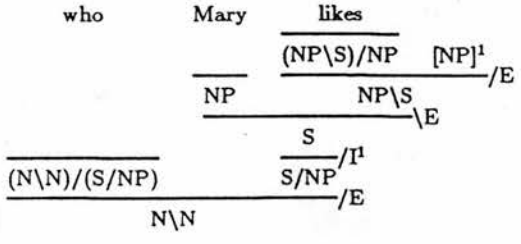
(10) $(\lambda y[\alpha])\beta = \alpha[\beta|y]$

Since exactly one assumption must be withdrawn, the resulting lambda-terms have the property that each binder binds exactly one variable occurrence; we refer to this as the 'single-bind' property (van Benthem 1983). The rules in (9) are analogous to the usual natural deduction rule of *conditionalization*, except that the latter allows withdrawal of any number of assumptions, in any position.

The /I and \I rules are commonly used in constructions that are assumed in other theories to involve 'empty categories', such as (11):

(11) (John is the man) who Mary likes.

We assume that the relative clause modifies the noun "man" and hence should receive the type N\N. The string "Mary likes" can be derived as of type S/NP, and so assignment of the type (N\N)/(S/NP) to the object relative pronoun "who" allows the analysis in (12) (cf. Ades and Steedman 1982):

(12)


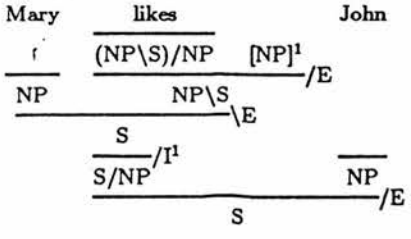
The meaning of the string can be read off the proof by interpreting /I and \I as lambda-abstraction, giving the term in (13):

(13) who $(\lambda x[(\text{likes } x)\ \text{Mary}])$

Note that this mechanism is only powerful enough to allow constructions where the extraction site is clause-peripheral; for non-peripheral extraction (and multiple extraction) we appear to need an extended logic, as described later.

DERIVATIONAL EQUIVALENCE AND NORMAL FORMS

In the above system it is possible to give more than one proof for a single reading of a string. For example, compare the derivation of "Mary likes John" in (7), and the corresponding lambda-term in (8), with the derivation in (14) and the lambda-term in (15):

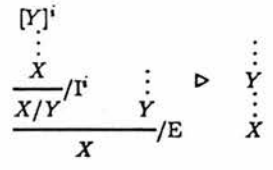
(14)


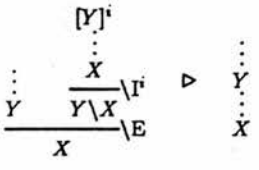
(15) $(\lambda x[(\text{likes } x)\ \text{Mary}])\ \text{John}$

By the definition in (10), the terms in (8) and (15) are β-equal, and thus have the same meaning; the proofs in (7) and (14) are said to exhibit *derivational equivalence*. The relation of derivational equivalence clearly divides the set of proofs into equivalence classes. We shall define a notion of *normal form* for proofs (and their corresponding terms) in such a way that each equivalence class of proofs contains a unique normal form (cf. Hepple and Morrill 1989).

We first define the notions of *contraction* and *reduction*. A contraction schema $R \rhd C$ consists of a particular pattern R within proofs or terms (the *redex*) and an equal and simpler pattern C (the *contractum*). A reduction consists of a series of contractions, each replacing an occurrence of a redex by its contractum. A normal form is then a proof or term on which no contractions are possible.

We define the following contraction schemas: *weak contraction* in (16) for proofs, and *β-contraction* in (17) for the corresponding lambda-terms.

(16) a.


b.


(17) $(\lambda y[\alpha])\beta \rhd \alpha[\beta|y]$

From (10) we see that β-contraction preserves meaning according to the standard functional interpretation of typed lambda-calculus. Therefore the corresponding weak contraction preserves the semantic functional interpretation of the proof; in addition it preserves the syntactic string interpretation since the redex and contractum contain the same leaves in the same order. For example, the proof in (14) weakly contracts to the proof in (7), and correspondingly the term in (15) β-contracts to the term in (8). The results of these contractions cannot be further contracted and so are the respective results of reduction to *weak normal form* and *β-normal form*.

Weak contraction in L strictly decreases the size of proofs (e.g. the number of symbols in a contractum is always less than that in a redex), and β-contraction in the single-bind lambda-calculus strictly decreases the size of terms. Thus there is *strong normalization* with respect to these reductions: every proof (term) reduces to a weak normal form (β-normal form) in a finite number of steps. This has as a corollary (*normalization*) that every proof (term) has a normal form, so that normal forms are *fully* representative: every proof (term) is equal to one in normal form. Since reductions preserve interpretations, an interpretation of a normal form will always be the

same as that of the original proof (term). Thus restricting the search to just such proofs addresses the problem of derivational equivalence, while preserving generality in that all interpretations are found.

Proofs in L and single-bind lambda-terms (like the more general cases of intuitionistic proofs and full lambda-terms) exhibit a property called the *Church-Rosser* property,[1] from which it follows that normal forms are unique.[2]

For formulations of L that are oriented to parsing, defining normal forms for proofs provides a basis for handling the so-called 'spurious ambiguity' problem, by providing for parsing methods which return all and only normal form proofs. See König (1989) and Hepple (1990).

STRUCTURAL MODALITIES

From a logical perspective, L can be seen as the weakest of a hierarchy of implicational sequent logics which differ in the amount of freedom allowed in the use of assumptions. The highest of these is (the implicational fragment of) the logistic calculus LJ introduced in Gentzen (1936). Gentzen formulated this calculus in terms of sequences of propositions, and then provided explicit *structural rules* to show the permitted ways to manipulate these sequences. The structural rules are *permutation*, which allows the order of the assumptions to be changed; *contraction*, which allows an assumption to be used more than once; and *weakening*, which allows an assumption to be ignored. For a discussion of the logics generated by dropping some or all of these structural rules see e.g. van Benthem (1987).

Although freely applying structural rules are clearly not appropriate in categorial grammars for linguistic description, commutable, iterable and optional elements do occur in natural language. This suggests that we should have a way to indicate that structural operations are permissible on specific types, while still forbidding their general application. To achieve this we propose to follow the precedent of the *exponential* operators of Girard's (1987) linear sequent logic, which lacks the rules of contraction and weakening, by suggesting a similar system of operators called *structural modalities*. Here we shall describe a system of *universal* modalities, which allow us to deal with the logic of commutable, iterable and optional extractions.[3]

For each universal modality we shall present an elimination rule, and one or more 'operational rules', which are essentially controlled versions of structural rules. (Introduction rules can also be defined, but we omit these here for brevity and because they are not required for the linguistic applications we discuss.) Note that these operators are strictly formal devices and not geared towards specific linguistic phenomena. Their use for the applications described, which are suggested purely for illustration, may lead to overgeneration in some cases.[4]

COMMUTATION

The type ΔX is assigned to an item of type X which may be freely permuted. Δ has the following inference rules:

(18)
$$\frac{\Delta X}{X}\,\Delta\mathrm{E} \qquad \frac{\Delta X \quad Y}{Y \quad \Delta X}\,\Delta\mathrm{Prm} \qquad \frac{Y \quad \Delta X}{\Delta X \quad Y}\,\mathrm{Prm}\Delta$$

From these rules we see that an occurrence of an item of type X in any position may be derived from an item of type ΔX.

We may use this operator in a treatment of relativization that will allow not only peripheral extraction as in (19a), but also non-peripheral extraction as in (19b):

(19) a. (Here is the paper) which Suzy read.

b. (Here is the paper) which Suzy read quickly.

We shall generate these examples by assuming that "which" licenses extraction from *any* position in the body of the relative clause. We may accomplish this by giving "which" the type (N\N)/(S/ΔNP) (cf. the extraction operator $\uparrow$ of Moortgat (1988)). This allows the derivations in (20a–b) (see Figure 1), which correspond to the lambda-terms in (21a–b) respectively:

(21) a. **which** (λx[(**read** x) **Suzy**])

b. **which** (λx[(**quickly** (**read** x)) **Suzy**])

ITERATION

The type $X^!$ is assigned to an item of type X which may be freely permuted and iterated. $^!$ has the following inference rules:

(22)
$$\frac{X^!}{X}\,{}^!\mathrm{E} \qquad \frac{X^! \quad Y}{Y \quad X^!}\,{}^!\mathrm{Prm} \qquad \frac{Y \quad X^!}{X^! \quad Y}\,\mathrm{Prm}^!$$

$$\frac{X^!}{X^! \quad X^!}\,{}^!\mathrm{Con}$$

[1] This is the property that if a proof (term) M reduces to two proofs (terms) N_1, N_2, then there is a proof (term) to which both N_1 and N_2 reduce.

[2] The above remarks also extend to a second form of reduction, *strong reduction/η-reduction*, which we have not space to describe here. See Morrill *et al.* (1990).

[3] The name is chosen because the elimination and introduction rules appropriate to each operator turn out to be those for the universal modality in the modal logic S4. See Dosen (1990).

[4] In Morrill *et al.* (1990) we give a system of modalities that differs from the present proposal in several respects. There are two unidirectional commutation modalities rather than the single bidirectional modality given here, and a *single* operational rule is associated with each of the universal modalities. We also suggest a (more tentative) system of *existential* modalities for dealing with elements that are themselves commutable, iterable or optional.

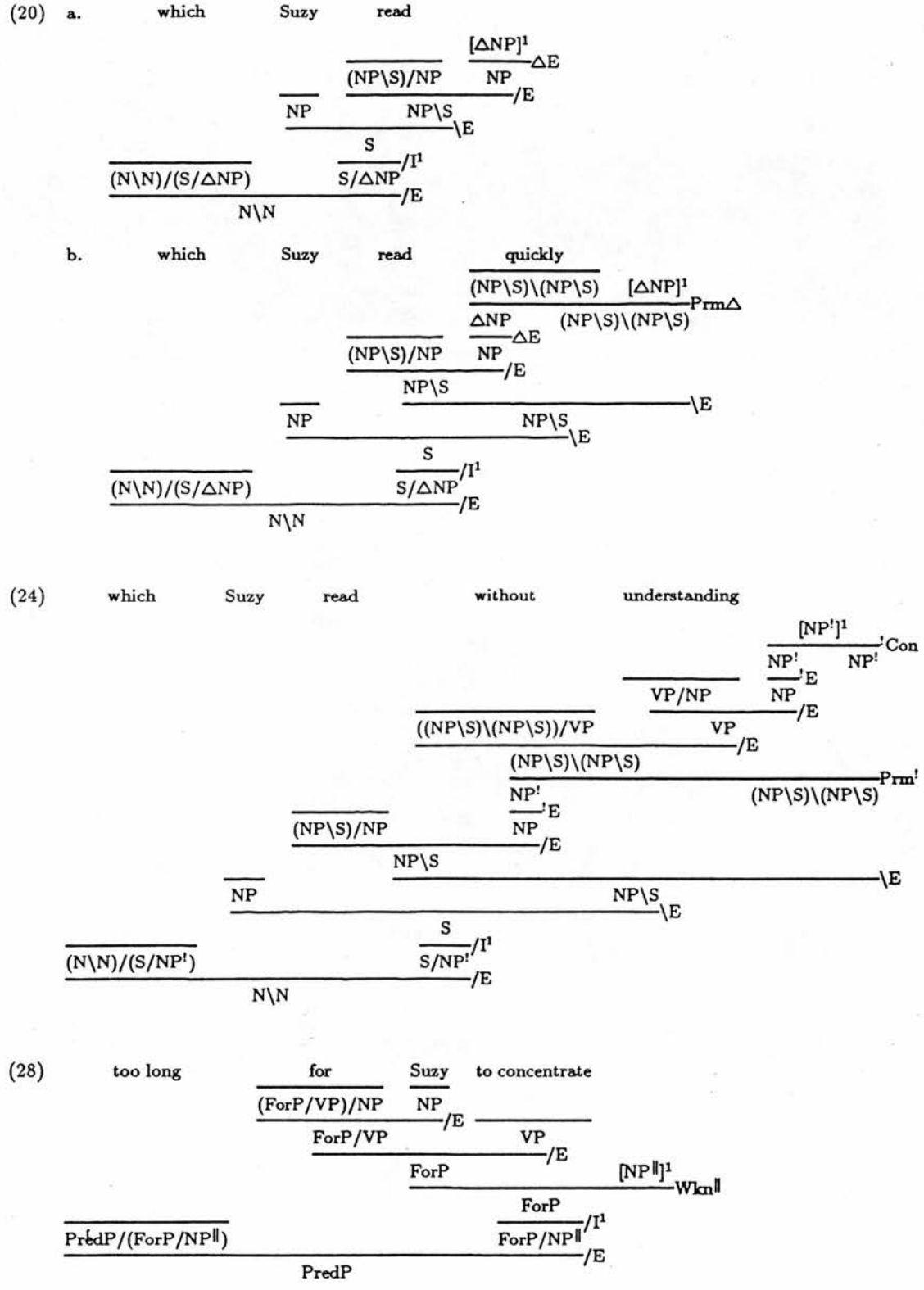


Figure 1. Derivations illustrating use of structural modalities

One or more occurrences of items of type X in any position may be derived from an item of type $X^!$.

We may use this modality in a treatment of multiple extraction. Consider the parasitic gap construction in (23):

(23) (Here is the paper) which Suzy read without understanding.

In order to generate both this example and the ones in (19), we shall now assume that "which" licenses extraction not just from any position in the body of a relative clause, but from *any number of* positions greater than or equal to one. We may do this by altering the type of "which" to $(N\backslash N)/(S/NP^!)$. Since $^!$ has all the inference rules of Δ, the derivations in (20) will still go through with the new type. In addition, the $^!$Con inference rule allows the derivation of (23) given in (24) (see Figure 1), and the corresponding term in (25):

(25) which (λx[((without (understanding x)) (read x)) Suzy])

OPTIONALITY

The type $X^{\|}$ is assigned to an item of type X which may be freely permuted, iterated and omitted. $^{\|}$ has the following inference rules:

(26)
$$\frac{\vdots\ X^{\|}}{X}\,{}^{\|}\mathrm{E} \qquad \frac{\vdots\ X^{\|}\quad \vdots\ Y}{Y\quad X^{\|}}\,{}^{\|}\mathrm{Prm} \qquad \frac{\vdots\ Y\quad \vdots\ X^{\|}}{X^{\|}\quad Y}\,\mathrm{Prm}^{\|}$$

$$\frac{\vdots\ X^{\|}}{X^{\|}\quad X^{\|}}\,{}^{\|}\mathrm{Con} \qquad \frac{\vdots\ X^{\|}\quad \vdots\ Y}{Y}\,{}^{\|}\mathrm{Wkn} \qquad \frac{\vdots\ Y\quad \vdots\ X^{\|}}{Y}\,\mathrm{Wkn}^{\|}$$

Zero or more occurrences of items of type X in any position may be derived from an item of type $X^{\|}$.

We may use this modality in a treatment of optional extraction, as illustrated by (27):

(27) a. (The paper was) too long for Suzy to read.

b. (The paper was) too long for Suzy to read quickly.

c. (The paper was) too long for Suzy to read without understanding.

d. (The paper was) too long for Suzy to concentrate.

We shall assume for simplicity that "to"-infinitives are single lexical items of type VP, that "for-to" clauses have a special atomic type ForP (so that "for" has the type (ForP/VP)/NP), and that predicate phrases have a special atomic type PredP. Given these assignments, the type $\mathrm{PredP}/(\mathrm{ForP}/\mathrm{NP}^!)$ for "too long" would allow (27a–c), but not (27d). In order to generate all four examples, we shall assume that "too long" licenses extraction from any number of positions in the embedded clause greater than or equal to zero, and thus give it the type $\mathrm{PredP}/(\mathrm{ForP}/\mathrm{NP}^{\|})$. Again, $^{\|}$ has all the inference rules of $^!$, generating (27a–c), and the $\mathrm{Wkn}^{\|}$ rule allows (27d) to be derived as in (28) (see Figure 1), giving the term in (29):

(29) too-long (λx[for (to-concentrate Suzy)])

CONCLUSIONS

We have introduced a scheme of proof figures for Lambek categorial grammar in the style of natural deduction, and proposed structural modalities which we suggest are suitable for the capture of linguistic generalizations. It remains to extend the semantic treatment of the structural modalities, to refine the proof theory, and hence to develop more efficient parsing algorithms. For the present, we hope that the proposals made can be seen as gaining linguistic practicality in the categorial description of natural language, without losing mathematical elegance.

REFERENCES

Ades, A.E. and Steedman, M.J. (1982). On the order of words. *Linguistics and Philosophy 4*, 517–558.

van Benthem, J. (1983). *The semantics of variety in categorial grammar.* Report 83–29, Department of Mathematics, Simon Fraser University. Also in W. Buszkowski, W. Marciszewski and J. van Benthem (eds), *Categorial Grammar*, Volume 25, Linguistic & Literary Studies in Eastern Europe, John Benjamins, Amsterdam/Philadelphia, 57–84.

van Benthem, J. (1987). Categorial grammar and type theory. Prepublication Series 87–07, Institute for Language, Logic and Information, University of Amsterdam.

Dosen, K. (1990). Modal logic as metalogic. To appear in P. Petkov (ed.), *Proceedings of the "Kleene '90" Conference*, Springer-Verlag.

Gentzen, G. (1936). On the meanings of the logical constants. In Szabo (ed., 1969), *The Collected Papers of Gerhard Gentzen*, North Holland, Amsterdam.

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

Hepple, M. (1990). Normal form theorem proving for the Lambek calculus. In *Proceedings of COLING 1990.*

Hepple, M. and Morrill, G. (1989). Parsing and derivational equivalence. In *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, UMIST, Manchester.

Hindley, J.R. and Seldin, J.P. (1986). *Introdution to Combinators and Lambda-Calculus.* Cambridge University Press, Cambridge.

Howard, W. (1980). The formulae-as-types notion of construction. In J.R. Hindley and J.P. Seldin (eds), *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, Academic Press, New York and London, 479–490.

Kōnig, E. (1989). Parsing as natural deduction. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics.*

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

Moortgat, M. (1988). *Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus.* Foris, Dordrecht.

Morrill, G., Leslie, N., Hepple, M. and Barry, G. (1990). Categorial deductions and structural operations. In G. Barry and G. Morrill (eds), *Edinburgh Working Papers in Cognitive Science, Volume 5: Studies in Categorial Grammar*, Centre for Cognitive Science, University of Edinburgh.

Prawitz, D. (1965). *Natural Deduction: a Proof Theoretical Study.* Almqvist and Wiksell, Uppsala.

Troelstra, A.S. (1969). *Principles of Intuitionism: Lecture Notes in Mathematics Vol. 95.* Springer-Verlag.