A TYPE SYSTEM FOR COMBINATORY CATEGORIAL GRAMMAR

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

GÜNEŞ ERKAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF COMPUTER ENGINEERING

AUGUST 2003

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Cem Bozşahin
Supervisor

Examining Committee Members

Assoc. Prof. Dr. Cem Bozşahin

Assist. Prof. Dr. Halit Oğuztüzün

Dr. Onur Tolga Şehitoğlu

Dr. Meltem Turhan Yöndem

Prof. Dr. Deniz Zeyrek

ABSTRACT

A TYPE SYSTEM FOR COMBINATORY CATEGORIAL GRAMMAR

Erkan, Güneş

M.S., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Cem Bozşahin

August 2003, 43 pages

This thesis investigates the internal structure and the computational representation of the lexical entries in Combinatory Categorial Grammar (CCG). A restricted form of typed feature structures is proposed for representing CCG categories. This proposal is combined with a constraint-based modality system for basic categories of CCG. We present some linguistic evidence to explain why both a unification-based feature system and a constraint-based modality system are needed for a lexicalist framework. An implementation of our system is also presented.

Keywords: Combinatory Categorial Grammar, type hierarchy, unary modality

ÖZ

ULAMSAL DİLBİLGİSİ İÇİN BİR TÜR SİSTEMİ

Erkan, Güneş

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Cem Bozşahin

Ağustos 2003, 43 sayfa

Bu tez, CCG (Ulamsal Dilbilgisi)'nin sözlük birimlerinin iç yapısını ve berimsel gösterimini incelemektedir. CCG kategorilerinin gösterimi için tür temelli özellik yapılarının sınırlandırılmış bir formu önerilmektedir. Bu öneri, CCG'nin temel kategorileri için tasarlanmış bir kip sistemiyle birleştirilmiştir. Sözlüksel formalizasyonların niçin hem eşleme-temelli hem de kısıt-temelli sistemlere gereksinim duyduğunu açıklamak için bazı dilbilimsel kanıtlar gösterilmiştir. Sistemimizin bir uygulaması da sunulmaktadır.

Anahtar Kelimeler: Ulamsal Dilbilgisi, tür hiyerarşisi, tekli kip

To my father, Hasan Hüseyin Erkan

ACKNOWLEDGMENTS

I would like to thank Jason Baldridge and Michael White, the other developers of OpenCCG, for their encouragement and suggestions in my effort to integrate my work into the OpenCCG project.

Special thanks goes to my supervisor, Cem Bozşahin, for his guidance and support in all stages of this study.

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Modern natural language theories try to give an explanation of the structure of a language construct in terms of its subparts or *constituents*. These formalisms differ in the constraints, rules and principles of how to define and combine these subparts to reach an analysis of the whole. Some of them, which are called *lexicalist* formalisms, assume that words (or *lexeme*s) are the most important atomic information unit in the language, where nearly all of the information needed for the grammar mechanisms is kept, or in Chomsky's (1981) terms, syntax is projected from the lexicon.

In this thesis, we primarily deal with Combinatory Categorial Grammar (CCG), one of the most promising lexicalist natural language formalisms. Our study investigates the internal structure and the computational representation of *categories*, the sole information units of CCG. Example (1) shows a sentence with some well-known categories assigned to its subparts. For instance, the category of *çocuk* and *kitabı* is np, and the category of *okudu* is s\np\np, which is a complex category composed of the atomic categories s and np.

(1) Çocuk kitabı okudu.
child book-ACC read-PAST

np np s\np\np

'The child read the book'.

In this study we look at the question "what exactly is a CCG category", that is, how categories can be represented and processed, and what are the consequences of different representations. We view CCG categories as typed feature structures, which are one of the widely used data structures in natural grammar formalisms (e.g. HPSG). We show that some linguistic phenomena like morphosyntactic parsing cannot be explained by feature structure mechanisms alone, but rather by unary modalities and constraints defined on them. Therefore, we combine our feature structure proposal with the unary modality mechanisms to reach our final representation for CCG atomic categories. In this sense, this thesis proposes a combination of the unification-based and the modality-based approaches to CCG basic categories.

Although we make use of the mechanisms of other formalisms, we keep the essential machinery and the generative power of CCG. We try to build a more elegant and complete theory by addressing some unanswered issues of CCG. In this sense, our work should be viewed as foundational, not as a critique or extension of CCG.

In Chapter 2, we make a brief introduction to CCG, its roots and syntactic mechanisms. We finish this chapter by making the assumption that CCG atomic categories are feature structures. Chapter 3 describes our proposal of defining a hierarchy for feature types of atomic categories. A formal definition of our proposal is given and the difference between CCG categories and typed feature structures in general sense is emphasized. Chapter 3 also presents some case studies where our type hierarchy produces elegant results to some unsolved or inefficiently handled problems in CCG. In Chapter 4, we view the type hierarchy framework from a completely different perspec-

tive where we use it as a tool to simulate morphosyntactic CCG, a recent development in CCG. We conclude this chapter with a discussion of the results we have gained from the morphosyntactic CCG case study. This discussion leads us to our final proposal on the structure of the atomic categories. Finally, in Chapter 5, we explain our implementation of the concepts we introduce in this thesis, and compare it with other similar systems.

CHAPTER 2

COMBINATORY CATEGORIAL GRAMMAR

Combinatory Categorial Grammar (CCG) (Steedman, 2000) is a natural language formalism which favors an extreme lexicalism ever since its inception into linguistics by Bar-Hillel (1953). CCG aims to store all of the syntactic and semantic information in the lexical entries and define a universal set of rules operating on them. The basic distinction from Chomskyan lexicalism is that the surface grammar of a language is lexicalized as well. In other words, for CCG, languages differ only in the lexicon. All of the mechanism functioning over the lexicon is universal. This mechanism also provides a transparent interface of syntax and semantics, which means that syntactic and semantic structures are built together inside the same mechanism and at the same time. For these reasons, CCG is one of the most promising formalisms towards a universal grammar, which is defined by Chomsky (1975) as:

(2) *. . . the system of principles, conditions, and rules that are elements or properties of all human languages . . . the essence of human language.*

This chapter first introduces pure categorial grammar (CG), and then CCG, which is an extension of CG. We will conclude with a discussion of the internal structure of

the atomic categories in CCG, on which our work in subsequent chapters will be built.

2.1 Pure Categorial Grammar

Pure categorial grammar (CG), often called AB calculus,[1] is the starting point for all categorial grammar formalisms. It is presented here because CCG is a direct extension of it. For more information on other extensions, see (Wood, 1993).

2.1.1 Categories

Crucial information units in CG are categories. A category may be atomic, or derived from other categories as a function of them.

Definition 2.1 (Syntactic Categories)

- The set of atomic categories, $\mathcal{A}$.
- The set of complex categories, $\mathcal{C}$, is the smallest set such that:
  - $\mathcal{A} \subseteq \mathcal{C}$
  - If $X, Y \in \mathcal{C}$, then $(X\backslash Y)$,$(X/Y) \in \mathcal{C}$

Lexical entries are represented as *word* := category pairs as in (3).

(3) a. *cat* := n

b. *John* := np

c. *loves* := (s\np)/np

[1] Named after Ajdukiewicz and Bar-Hillel. Ajdukiewicz's calculus (Ajdukiewicz, 1935), originally proposed for formal languages, was adapted to natural languages by Bar-Hillel (1953).

2.1.2 Rules

Complex categories can be viewed as curried functions with slashes specifying the direction of the arguments they seek for. X/Y denotes an argument Y to the right, while X\Y denotes an argument Y to the left.[2] The result is X. Categories are combined via two application rules:

(4) Forward Application (>): $X/Y \quad Y \Rightarrow X$

Backward Application (<): $Y \quad X\backslash Y \Rightarrow X$

A derivation in CG is incremental, that is, when an application rule is applied to two categories, they are not available for further steps of the derivation. Only the result category can be used. An example derivation is shown in (5).

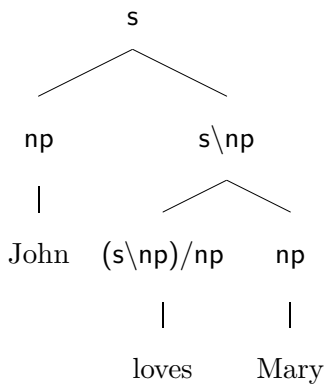
(5)

John	loves	Mary.
np	(s\np)/np	np
	s\np (>)	
s (<)		

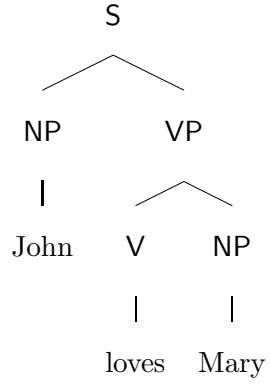
When we redraw this derivation upside-down as a parse tree like in (6a), we see that the result is nothing but the classical constituent analysis of the sentence in (6b) with derived names for nodes. Application rules are isomorphic to context-free grammar (CFG) productions, except for the difference that CFG parse trees are generated top-down while CG trees are generated bottom-up. Indeed, pure CG (AB calculus) is proved to be context-free (Bar-Hillel, Gaifman, and Shamir, 1964).

[2] This is Steedman notation. An alternative notation is due to Lambek, in which results are written on top; eg., (s\np)/np in Steedman notation is (np\s)/np in Lambek notation.

(6) a.



b.



2.1.3 Semantics

One of the most important features of CG is its syntax-semantics interface. All syntactic categories are paired with their semantic interpretation by the $\sigma : \mu$ notation, where σ denotes the syntactic category and μ denotes the semantic category. Semantics is represented with λ-calculus terms. For example, lexical entries in (3) look like (7) with their semantic interpretation.

(7) a. $cat := \mathsf{n} : \mathbf{cat}$

b. $John := \mathsf{np} : \mathbf{John}$

c. $loves := (\mathsf{s}\backslash\mathsf{np})/\mathsf{np} : \lambda x.\lambda y.\mathbf{loves}(y, x)$

Every syntactic rule has its semantic counterpart. Application rules are repeated in (8) with their semantic interpretation. This means that syntactic and semantic derivations are always performed in parallel. (9) shows both the syntactic and semantic analysis of (5).

(8) Forward Application (>): $\mathsf{X}/\mathsf{Y} : f \quad \mathsf{Y} : a \Rightarrow \mathsf{X} : fa$

Backward Application (<): $\mathsf{Y} : a \quad \mathsf{X}\backslash\mathsf{Y} : f \Rightarrow \mathsf{X} : fa$

(9)

John	loves	Mary.
np : **John**	(s\np)/np : $\lambda x.\lambda y.$**loves**(y, x)	np : **Mary**
	s\np : $\lambda y.$**loves**(y,**Mary**) (>)	
s : **loves**(**John**,**Mary**) (<)		

From now on, we will not include semantic expressions in our examples in this thesis since our work mainly deals with syntax. However, the important point here is to note that every syntactic discussion will have an implicit semantic counterpart going in parallel with it because of the transparent syntax-semantics interface of CG sketched above.

2.2 Combinatory Categorial Grammar

2.2.1 Rules

There is strong evidence that context-free power is not enough for analyzing natural languages (Shieber, 1985). Combinatory Categorial Grammar(CCG) (Steedman, 2000) is an extension of pure CG with three more rule classes: composition (10a), type raising (10b) and substitution (10c).

(10) a. Forward Composition (> **B**): $X/Y \quad Y/Z \quad \Rightarrow_{\mathbf{B}} \quad X/Z$

Backward Composition (< **B**): $Y\backslash Z \quad X\backslash Y \quad \Rightarrow_{\mathbf{B}} \quad X\backslash Z$

b. Forward Type Raising (> **T**): $X \quad \Rightarrow_{\mathbf{T}} \quad Y/(Y\backslash X)$

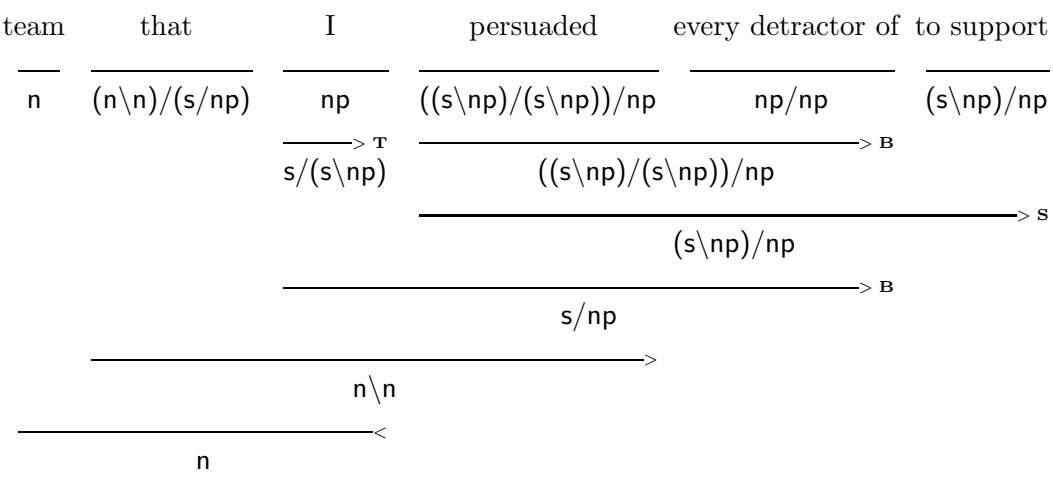
Backward Type Raising (< **T**): $X \quad \Rightarrow_{\mathbf{T}} \quad Y\backslash(Y/X)$

c. Forward Substitution (> **S**): $(X/Y)/Z \qquad Y/Z \quad \Rightarrow_{\mathbf{S}} \quad X/Z$

Backward Substitution (< **S**): $Y\backslash Z \qquad (X\backslash Y)\backslash Z \quad \Rightarrow_{\mathbf{S}} \quad X\backslash Z$

Derivation (11), excerpt from (Baldridge, 2002), exemplifies all four classes of rules in CCG. This derivation is not possible in the absence of the rules in 10.

(11)



These rules bring CCG to the *mildly context-sensitive* generative power (Vijay-Shanker and Weir, 1994), which is stronger than context-freeness but weaker than context-sensitivity.

2.2.2 Atomic Categories As Feature Structures

Although atomic categories are the sole information units, most of the CCG literature focuses on higher level mechanisms rather than the internal structure of the categories. This causes a potential problem for people in search for a computational realization of the theory since the design of the atomic categories is the crucial part of such a system.

We have already seen some atomic categories like n, np and s in above examples. However, we have ignored many important linguistic phenomena (case marking, agreement, person, etc.) in those examples. When such information is needed, it is often a convenient way to use subscripts as in (12). Example (12a) specifies that *cat* is a singular noun. In (12b), *John* is a third person singular noun phrase. In (12c), *loves* is a transitive verb seeking an accusative object and a nominative subject.

(12) a. $cat := \mathsf{n}_{\mathrm{sing}}$

b. *John* := $\mathsf{np}_{\mathrm{3rd}}$

c. *loves* := $(\mathsf{s}\backslash\mathsf{np}_{\mathrm{nom}})/\mathsf{np}_{\mathrm{acc}}$

This suggests that atomic categories may be represented by simple feature structures and be handled by unification. Indeed, this is an implicit assumption in almost all of the work in CCG. Any computational natural language framework should deal with unification to some extent; and when we have unification, we have to have features to unify. For CCG, it suffices to have a set of features for each atomic category. These feature structures are nothing but a simple set of properties with atomic values or variables. (13) shows CCG atomic categories for the words *cat* and *John*, respectively. It is clear from these examples that what we call np in the above examples is actually an n feature structure with the SPEC feature set to *+*, that is, a specified noun.[3] From a computational point of view, complex categories are then ordered and optionally nested list of these feature structures (or atomic categories) delimited by slashes.

(13) a. cat := $\begin{bmatrix} \mathsf{n} & \\ \mathsf{SPEC} & - \\ \mathsf{NUM} & \mathit{sing} \end{bmatrix}$ b. John := $\begin{bmatrix} \mathsf{n} & \\ \mathsf{SPEC} & + \\ \mathsf{NUM} & \mathit{sing} \\ \mathsf{PERS} & \mathit{3rd} \end{bmatrix}$

The feature structures of CCG should not be compared to those of Head-driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994). In HPSG, feature structures play such an important role that all of the syntactic and semantic mechanisms function *inside* them. Unification in HPSG is so powerful that there is no need for

[3] This is an assumption used in some CCG studies like (Baldridge, 2002). Alternatively, one may leave out the SPEC feature completely and make np a separate atomic category type.

another mechanism in parsing. Feature structures of HPSG are put in a global hierarchy so that they are nested according to this hierarchy. All combinatoric information provided by complex categories and universal rules in CCG is contained in *nested* and *coindexed* feature structures of HPSG.

CHAPTER 3

TYPE HIERARCHY IN CCG LEXICON

Unification is used extensively in natural language formalisms. Most common version of unification is the simple matching of unordered and possibly nested values. Prolog has become the dominant programming language in natural language grammar realizations because of its built-in unification mechanism. In this chapter, we list some theoretical and practical disadvantages of simple value matching and emphasize the need for a unification that respects a language-dependent hierarchy of values. We discuss the integration of this kind of unification into the CCG framework and show its significance by looking at some case studies.

3.1 Motivation

There are cases which call for a controlled degree of flexibility for feature values, where one or more features of some lexical entry can have more than one value. For example, in English, unlike most other nouns, NUMBER feature for *fish* can have both *singular* or *plural* value. A common technique in lexicon design, called *underspecification*, is to leave out the corresponding feature from such categories. For example, in (14b),

NUMBER (NUM) feature for *fish* is underspecified, compared to the noun in (14a). Since unification of a nonexistent feature always succeeds, NUMBER feature of (14b) gets whatever value it is unified with.

(14) a. cat := $\begin{bmatrix} \mathsf{n} & \\ \mathsf{SPEC} & \text{-} \\ \mathsf{NUM} & \textit{sing} \end{bmatrix}$ b. fish := $\begin{bmatrix} \mathsf{n} & \\ \mathsf{SPEC} & \text{-} \end{bmatrix}$

One problem about underspecified features is that they are subject to have any value as a result of a unification operation. Sometimes this is not a desired effect since we may want the underspecified feature to select a value from a subset of possible values. For example, bare forms of Turkish nouns can behave both as nominative and accusative but never as dative. Therefore they may appear as subjects or objects of verbs that seek for nominative or accusative nouns, but never as indirect objects of verbs that seek dative nouns. In (15a) and (15b), categories for the verbs *gördü* and *bindi* are given.[1] When a noun is lexically underspecified for its CASE feature (15c), the lexicon licenses the grammatical sentence (16a), but overgenerates some ungrammatical constructs such as (16b).

[1] Since the canonical word order for Turkish is Subject-Object-Verb, the category of a transitive verb would be $\mathsf{s}\backslash\mathsf{n}\backslash\mathsf{n}$.

(15) a. gördü := $\mathsf{s}\backslash\mathsf{n}_{\mathrm{nom}}\backslash\mathsf{n}_{\mathrm{acc}}$

b. bindi := $\mathsf{s}\backslash\mathsf{n}_{\mathrm{nom}}\backslash\mathsf{n}_{\mathrm{dat}}$

c. uçak := $\begin{bmatrix} \mathsf{n} & \\ \mathsf{NUM} & \mathit{sing} \end{bmatrix}$

d. $\text{uçak}_{(1)}$:= $\begin{bmatrix} \mathsf{n} & \\ \mathsf{NUM} & \mathit{sing} \\ \mathsf{CASE} & \mathit{nom} \end{bmatrix}$, $\text{uçak}_{(2)}$:= $\begin{bmatrix} \mathsf{n} & \\ \mathsf{NUM} & \mathit{sing} \\ \mathsf{CASE} & \mathit{acc} \end{bmatrix}$

(16) a. Adam uçak gördü.

$\underline{\mathsf{n}_{\mathrm{nom}}} \quad \underline{\mathsf{n}} \quad \underline{\mathsf{s}\backslash\mathsf{n}_{\mathrm{nom}}\backslash\mathsf{n}_{\mathrm{acc}}}$

b. *Adam uçak bindi.

$\underline{\mathsf{n}_{\mathrm{nom}}} \quad \underline{\mathsf{n}} \quad \underline{\mathsf{s}\backslash\mathsf{n}_{\mathrm{nom}}\backslash\mathsf{n}_{\mathrm{dat}}}$

c. Adam uçak gördü.

$\underline{\mathsf{n}_{\mathrm{nom}}} \quad \underline{\mathsf{n}_{\mathrm{acc}}} \quad \underline{\mathsf{s}\backslash\mathsf{n}_{\mathrm{nom}}\backslash\mathsf{n}_{\mathrm{acc}}}$

A common technique, used not only in CCG but also in many lexicalist formalisms, is to have distinct lexical entries for each possible feature value, as in (15d). We now have (16c) instead of (16a). Although this technique works, it enlarges the lexicon. Having a large lexicon with more than one category for some words creates theoretically and practically serious problems. First of all, CCG attempts to minimize the size of the lexicon by adopting the following principle defined by Steedman (2000):[2]

(17) *The Principle of Head Categorial Uniqueness (HCU):*
A single nondisjunctive lexical category for the head of a given construction specifies both the bounded dependencies that arise when its complements are in canonical position and the unbounded dependencies that arise when those complements are displaced under relativization, coordination, and the like.

[2] Minimizing the lexicon has important value in some other linguistic theories like GB (Chomsky, 1981) and LFG (Bresnan, 2001). For a further discussion of the HCU principle, as well as its significance in language acquisition, see (Steedman, 2000).

HCU states that it is best to have a *single* category for each word, at least for each canonical role. Lexical categories in (15d) are clearly a violation of this principle.

From a computational point of view, this technique leads to inefficient parsing since backtracking would be needed for a wrong choice among the possible categories for a word. Lexicon design would also be a tedious work. It is infeasible to enumerate every possible category for each word considering the above example, where enumerating only the case values for nouns enlarged the lexicon almost by a factor of two.

To remedy this problem, we propose to incorporate a type system for feature values, and an inheritance specification for types, as originally envisaged by HPSG (Pollard and Sag, 1987).

3.2 Type Hierarchy

3.2.1 Formal Definition of Feature Structures and Unification

HPSG has a built-in mechanism, which is called a *type hierarchy*, for the unification problem described in Section 3.1. In HPSG, all feature values are typed, and unification of these values are checked according to a predefined hierarchy. Following Carpenter (1992), we formally define (typed) feature structures of HPSG as follows:

Definition 3.1 (Feature Structure): Given a finite set of features **F** and a finite set of types **T**, a feature structure over **F** and **T** is a tuple $F =< Q, q_r, \theta, \delta >$, where:

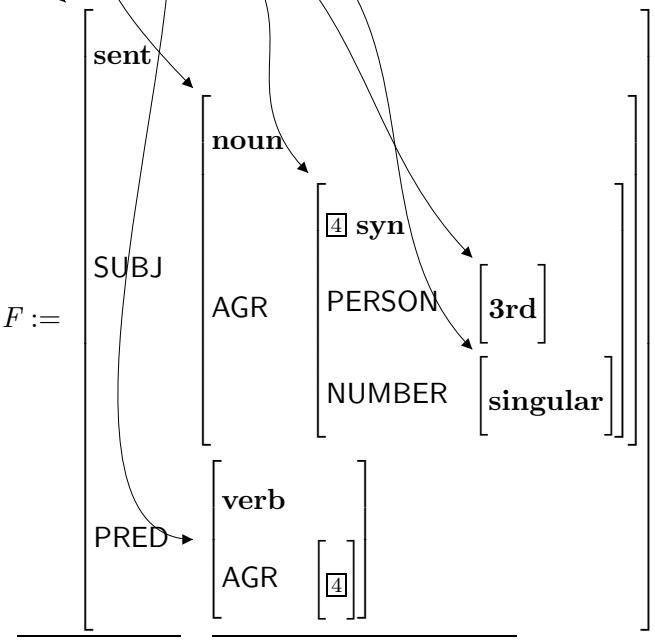
- Q : a finite set of *nodes*
- $q_r \in Q$: the root node
- $\theta : Q \rightarrow \mathbf{T}$: total *typing* function
- $\delta : \mathbf{F} \times Q \rightarrow Q$: partial *feature value* function

For example, the feature structure in (18) has six nodes. Nodes are nested in each other by being values to features. The root node has two features, SUBJ and PRED. Value of SUBJ feature is another node, whose AGR feature value is another node. Typing and feature value functions are given in the example for clarification.

(18) $F =< Q, q_1, \theta, \delta >$

$Q = \{q_1, q_2, q_3, q_4, q_5, q_6\}$

$F :=$



q	$\theta(q)$
q_1	sent
q_2	noun
q_3	verb
q_4	syn
q_5	person
q_6	singular

q	$feat$	$\delta(feat, q)$
q_1	SUBJ	q_2
q_1	PRED	q_3
q_2	AGR	q_4
q_3	AGR	q_4
q_4	PERSON	q_5
q_4	NUMBER	q_6

Type hierarchy is a partial order defined on types and specifies which types are unifiable with each other. Two types are unifiable only if they have a common subtype. For the example type hierarchy in Figure 3.1 **b** and **c** are unifiable while **b** and **d** are not. The resultant type of the unification of two types is the highest common subtype

of them in the hierarchy. Referring to the same figure, **a**⊔**b**=**b** and **b**⊔**c**=**e**, where ⊔ stands for the unification operator. Finally, two feature structures are unifiable only if their types are unifiable. How type hierarchy resolves the unification problem in Section 3.1 will be clear in Section 3.2.2, where we adapt the type hierarchy to the atomic categories of CCG.

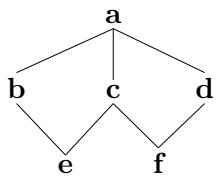


Figure 3.1: An example type hierarchy.

3.2.2 Type Hierarchy In CCG

In Section 2.2.2, we made an informal introduction to the internal structure of the atomic categories of CCG and said that they are simple feature structures. Following Definition 3.1, we may now formally define CCG atomic categories:

Definition 3.2 (CCG Atomic Category): A CCG atomic category is a feature structure in which no node other than the root node can have features.

Definition 3.2 implies that the depth of nesting the feature structures in atomic categories can be maximum one. This restriction brings the power of unification to a level that is envisaged by CCG. It contrasts with more powerful unification in HPSG, where feature structures can be nested indefinitely. Figure 3.2, excerpt from (Bozsahin, 2002), shows the HPSG analysis of a Turkish relative clause, where the extracted element (**NP**) is stored in the SLASH feature of other feature structures until it is found at a higher level in the parse tree; the propagation of this knowledge

is carried out by re-entrant unification, i.e. the common index $\boxed{1}$. On the other hand, the whole idea of CCG is to get rid of such a powerful unification mechanism and handle all syntactic phenomena by the use of complex categories and combinatory rules, in effect minimizes the linguistic work done by unification.

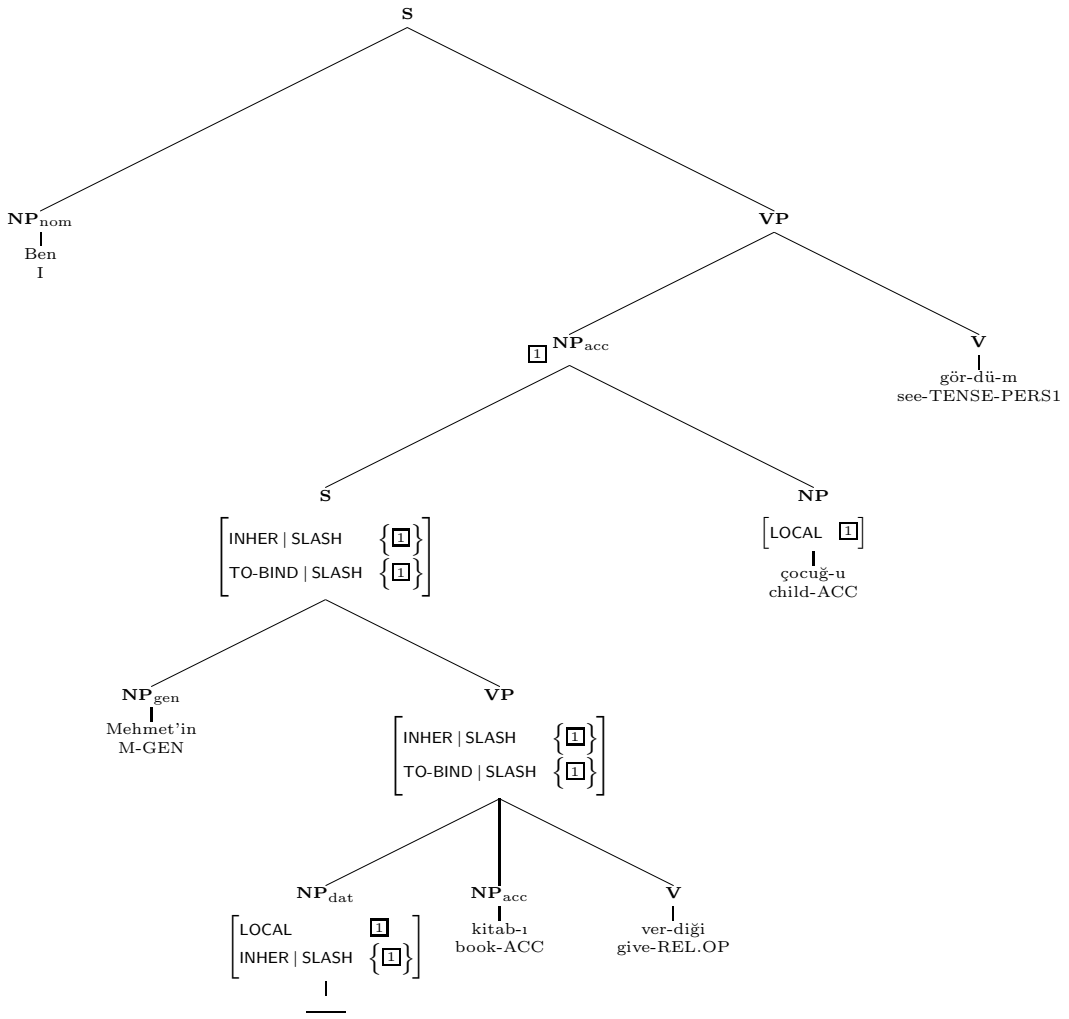


Figure 3.2: HPSG analysis of a Turkish relative clause (from (Bozsahin, 2002)).

By the above formalizations, it is now clear that what we have called *atomic feature values* previously are indeed typed feature structures with no features. For instance, in (15c), the type of the whole feature structure is n, and the value of the NUM feature is an empty feature structure with type *sing*.[3] When we talk about the category $\mathsf{n}_{\mathsf{sing}}$,

[3] It is a common convention to represent empty feature structures with their types in *italic* and

we refer to a feature structure of type n with a feature value of type *sing*.

Since CCG atomic categories are a restricted form of feature structures, all mechanisms of feature structures, including type hierarchy, can be applied to them. Recalling the unification problem in Section 3.1, we may want to create a new case type for Turkish bare nouns, combining nominative and accusative cases:

(19) uçak := $\begin{bmatrix} \mathsf{n} & \\ \mathsf{NUM} & \mathit{sing} \\ \mathsf{CASE} & \mathit{nom\_or\_acc} \end{bmatrix}$

The unification of this category with verbs is established via the simple type hierarchy in Figure 3.3. Since **nom_or_acc**⊔**acc**=**acc**, sentence (20a) is successfully parsed. The problem of overgenerating (16b), repeated here as (20b), is solved since **dat** and **nom_or_acc** cannot unify with each other according to the type hierarchy.

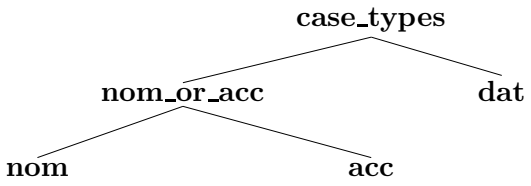


Figure 3.3: An instance of Turkish case hierarchy.

(20) a. Adam uçak gördü.

$\mathsf{n}_{\mathrm{nom\_or\_acc}}$ $\mathsf{n}_{\mathrm{nom\_or\_acc}}$ $\mathsf{s}\backslash\mathsf{n}_{\mathrm{nom}}\backslash\mathsf{n}_{\mathrm{acc}}$

$\mathsf{s}\backslash\mathsf{n}_{\mathrm{nom}}$ <

s <

b. *Adam uçak bindi.

$\mathsf{n}_{\mathrm{nom\_or\_acc}}$ $\mathsf{n}_{\mathrm{nom\_or\_acc}}$ $\mathsf{s}\backslash\mathsf{n}_{\mathrm{nom}}\backslash\mathsf{n}_{\mathrm{dat}}$

< ***cannot unify...

without the square brackets [. . .].

Another attempt to integrate type hierarchy into CCG is the typed-inheritance CCG (TCCG) of Beavers (2003), which has been developed recently. However, TCCG makes use of nested typed feature structures extensively. Basic idea of TCCG is to represent all categories (atomic or complex) as feature structures. All the rest is handled by unification as in HPSG. In other words, TCCG can be viewed as an HPSG system with a different feature structure design that makes use of CCG categories. Indeed, TCCG has been developed using LKB (Copestake, 1992), which is a widely-used system to implement HPSG grammars. Moreover, there is no use of type hierarchy in low-level features such as case, person, or morphosyntax, as we present in Chapter 4. Hence, TCCG is the opposite of our system in the sense that it uses the type hierarchy where we avoid it, and we use the type hierarchy where TCCG does not.

3.3 Case Studies

In this section, we present some case studies where our type hierarchy proposal for CCG gives linguistically and/or computationally fruitful results. We show more examples of the use of controlled unification in lexicon minimization such as the one in example (20). Although our examples will be only from English and Turkish, similar arguments can be made for other languages as well.

3.3.1 Case Marking

We have already seen an example use of the type hierarchy in Figure 3.3 on page 19, where we marked Turkish bare nouns as both accusative and nominative. Another case in Turkish where we need a controlled unification of case values is relativization. There are two types of relativization in Turkish: subject relativization realized by the affixes *-(y)An, -(y)AcAk,* and *-mIş*, and object relativization realized by the affixes *-dIk-* and

-(y)AcAk-. The general expression for the category of these affixes is $(n/n)\backslash(s\backslash np)$, where np in the category unifies with the extracted element (subject or object) in the relativization (Bozsahin, 2002).[4] For instance, subject relativizers should look for a nominative np since the subject will be nominative no matter what the verb is. On the other hand, object relativizers should look for an object category np, which has a case feature value depending on the verb's category (21). This case controlling problem can be solved by defining the hierarchy in Figure 3.4 and assigning the object relativizer the category $(n/n)\backslash(s\backslash np_{non\_nom})$ so that *non_nom* type unifies with *acc* in (21a) and with *dat* in (21a).

(21) a. adam ın gör -düğü

$$\dfrac{\dfrac{\dfrac{\dfrac{n \quad (s_{agr}/(s_{agr}\backslash np_{nom}))\backslash n}{s_{agr}/(s_{agr}\backslash np_{nom})}<} \quad s\backslash np_{nom}\backslash np_{acc}}{s_{agr}\backslash np_{acc}}>\mathbf{B}_{\times} \quad (n/n)\backslash(s_{agr}\backslash np_{acc})}{n/n}<$$

b. adam ın bin -diği

$$\dfrac{\dfrac{\dfrac{\dfrac{n \quad (s_{agr}/(s_{agr}\backslash np_{nom}))\backslash n}{s_{agr}/(s_{agr}\backslash np_{nom})}<} \quad s\backslash np_{nom}\backslash np_{dat}}{s_{agr}\backslash np_{dat}}>\mathbf{B}_{\times} \quad (n/n)\backslash(s_{agr}\backslash np_{dat})}{n/n}<$$

Type hierarchy can also be used in many cases for lexicon economy. For example, *you* pronoun can be both accusative and nominative in English. Instead of having two distinct lexical entry, we may have a *nom_or_acc* type and assign it to the case feature of *you*. This technique may increase efficiency in some parsing situations.

[4] Assigning categories to affixes to enable morpheme-based parsing has been recently integrated into the CCG framework by Bozsahin's (2002) morphosyntactic CCG, which we will describe in Section 4.1.

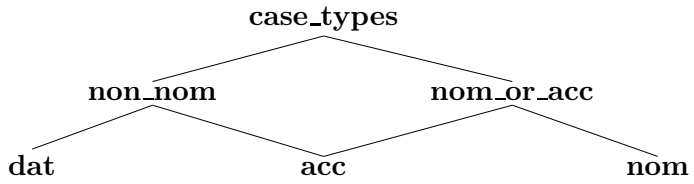


Figure 3.4: Turkish case hierarchy for handling relative clauses (extended version of Figure 3.3).

3.3.2 Person

It is a common engineering technique to assign a binary value to the PERSON feature of English noun phrases indicating whether it is third person singular or not. However, this is an ad hoc approach that makes use of the special case in subject-verb agreement of English where verb forms are all the same except for the third person. This is a potential problem for other languages that have more complicated verbal morphology. Even in English, there are cases where we need a more detailed person discrimination (e.g. the auxiliary verb *be*). The problem gets more serious for a semanticist who looks for the answer to the question "*who* did what".

Figure 3.5 shows a simple hierarchy for the PERSON feature of the English noun phrases.[5] In (22), we list the feature structures for some English words assuming this hierarchy. The feature structures for some atomic categories are not shown since they are irrelevant to the discussion.

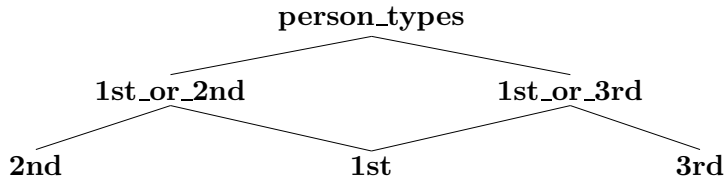


Figure 3.5: Person hierarchy for English.

[5] We treat number and person features separately for the simplicity of the hierarchy. However, in most implementations, number and person agreements are handled in one hierarchy for efficiency.

(22) a. I := $\begin{bmatrix} \mathsf{np} & \\ \mathsf{NUM} & \mathit{sing} \\ \mathsf{CASE} & \mathit{nom} \\ \mathsf{PERSON} & \mathit{1st} \end{bmatrix}$

b. you := $\begin{bmatrix} \mathsf{np} & \\ \mathsf{CASE} & \mathit{nom\_or\_acc} \\ \mathsf{PERSON} & \mathit{2nd} \end{bmatrix}$

c. wins := $\mathsf{s} \setminus \begin{bmatrix} \mathsf{np} & \\ \mathsf{NUM} & \mathit{sing} \\ \mathsf{CASE} & \mathit{nom} \\ \mathsf{PERSON} & \mathit{3rd} \end{bmatrix}$

d. won := $\mathsf{s} \setminus \begin{bmatrix} \mathsf{np} & \\ \mathsf{CASE} & \mathit{nom} \\ \mathsf{PERSON} & \mathit{person\_types} \end{bmatrix}$

e. is := $(\mathsf{s} \setminus \begin{bmatrix} \mathsf{np} & \\ \mathsf{NUM} & \mathit{sing} \\ \mathsf{CASE} & \mathit{nom} \\ \mathsf{PERSON} & \mathit{3rd} \end{bmatrix}) \,/\mathsf{adj}$

f. was := $(\mathsf{s} \setminus \begin{bmatrix} \mathsf{np} & \\ \mathsf{NUM} & \mathit{sing} \\ \mathsf{CASE} & \mathit{nom} \\ \mathsf{PERSON} & \mathit{1st\_or\_3rd} \end{bmatrix}) \,/\mathsf{adj}$

CHAPTER 4

UNARY MODALITIES IN CCG LEXICON

In this chapter, we look at the type hierarchy from a different perspective where we propose it as a tool in implementing morphosyntactic CCG (MCCG), which is a recent development in CCG framework. We finish this chapter in Section 4.2 with a discussion of the use of the typed feature structures and/or the unary modalities of MCCG in the design of atomic categories.

4.1 Morphosyntactic CCG

Morphosyntactic extension to CCG (Bozsahin, 2002), described in Section 4.1.1, brings us the solutions to some unsolved problems in CCG formalism such as semantic bracketing as well as presenting a more elegant and transparent interface of morphology, syntax and semantics. However, it complicates the theory by adding new mechanisms and changing the syntactic calculus. This creates the practical problem of adapting the widely used CCG realizations to these new mechanisms, which is a potential barrier to making morphosyntactic CCG a standard in the CCG community. In this section, we present the type hierarchy as a complete mechanism to implement

morphosyntax so that no other extensions to CCG are needed.

4.1.1 Background

Morphosyntactic CCG (MCCG) is a recent extension of CCG in which bound morphemes appear as separate lexical items.[1] The key idea of MCCG is that the inflectional morphemes can take part in the construction of linguistic expressions just like words, and that the inflectional paradigm of a language can be represented as a partial ordering. For example, in Turkish, number marking should always occur before case marking. Therefore, *kedi-ler-e* (cat-NUM-CASE) is grammatical, but *kedi-ye-ler* (cat-CASE-NUM) is not. These kinds of orderings give us a hierarchy in the form of a lattice (Figure 4.1).[2]

In MCCG, atomic categories are represented as $\overset{\alpha}{\square} A$, where α is a diacritic indicating the place of the category in the hierarchy and $\square$ is a morphosyntactic unary modality. There are two kinds of modalities: $\lhd$ ('up to and equals') and $\bowtie$ ('equals'). $\overset{n}{\bowtie}\mathsf{n}$ represents the set of nouns marked on number while $\overset{n}{\lhd}\mathsf{n}$ represents the set of nouns marked on number or any other diacritic that is lower than number in the hierarchy. These modalities provide different granularity of control over the basic categories.[3] For instance, if $\upsilon(\tau)$ denotes the set of strings that have type τ, then $\upsilon(\overset{n}{\lhd}\mathsf{n}) \subseteq \upsilon(\overset{c}{\lhd}\mathsf{n})$, but $\upsilon(\overset{n}{\bowtie}\mathsf{N}) \not\subseteq \upsilon(\overset{c}{\bowtie}\mathsf{N})$ according to Figure 4.1.

The extension of the combinatory rules of CCG to handle morphosyntactic modalities is handled via the conditions shown in (23). Other rules are extended similarly.

---

[1] For a further discussion of MCCG and its linguistic significance, refer to the original paper (Bozsahin, 2002). All examples and figures in this section are excerpt from Bozsahin's work.

[2] Note that every language has its own hierarchy with the expectation that morphologically rich languages have deeper and more complex hierarchies.

[3] These modalities differ from multi-modal CCG (Baldridge, 2002), which uses grammatical modalities as a way to model lexical constraints on the *combinatory* system, not as constraints on basic categories.

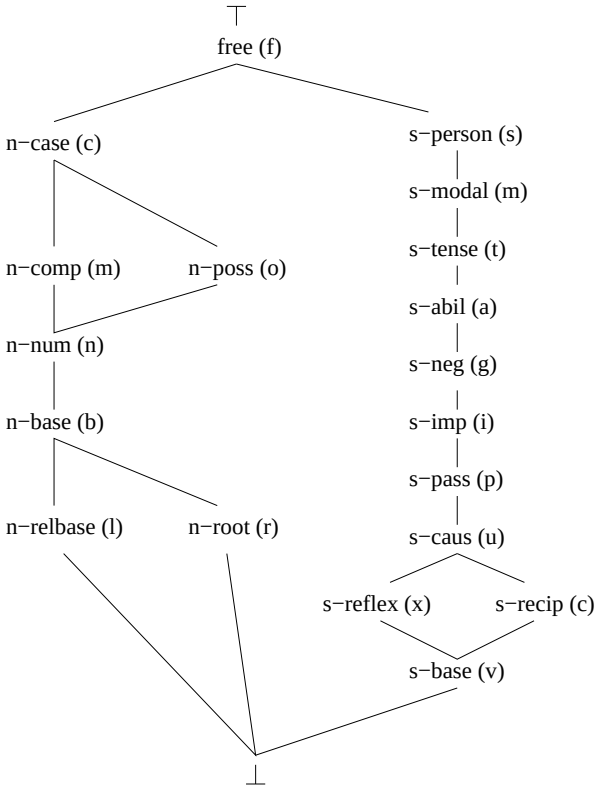


Figure 4.1: The lattice of diacritics for Turkish.

Example (24) shows an example derivation with morphosyntactic parsing.

(23) Forward Application (>):

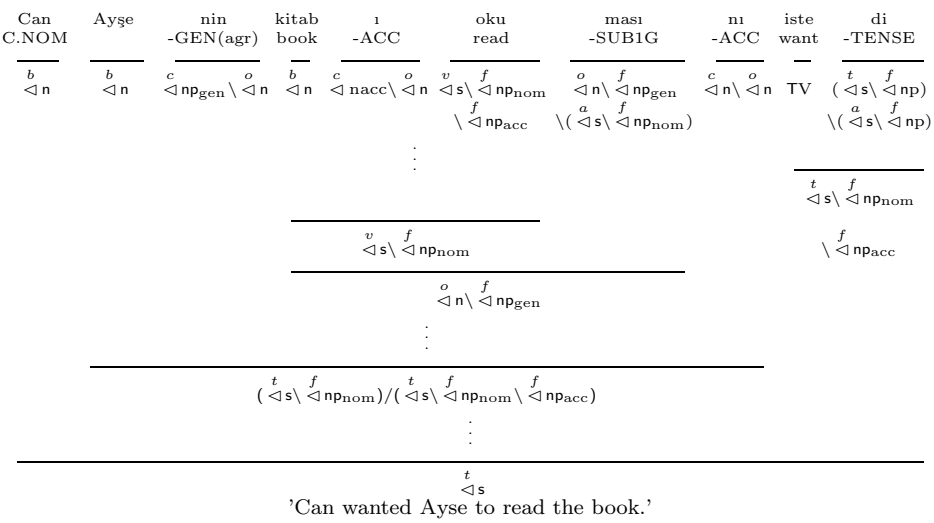
$$\frac{X/\,{}^{\alpha_1}_{\square_1}\,Y : f \qquad {}^{\alpha_2}_{\square_2}\,Y : a}{X : fa}\,{>}$$

if $\alpha_2 \square_1 \alpha_1$ in lattice L, for: $\square_1, \square_2 \in \{\bowtie, \lhd\}$, $\alpha_1, \alpha_2 \in \mathcal{D}$ in L,

Forward Composition (> **B**):

$$\frac{X/\,{}^{\alpha_1}_{\square_1}\,Y : f \qquad {}^{\alpha_2}_{\square_2}\,Y/Z : g}{X/Z : \lambda x.f(gx)}\,{>}\mathbf{B}$$

if $\alpha_2 \square_1 \alpha_1$ in lattice L, for: $\square_1, \square_2 \in \{\bowtie, \lhd\}$, $\alpha_1, \alpha_2 \in \mathcal{D}$ in L,



4.1.2 Embedding MCCG in Feature Structures

The constraints in (23), which check the applicability of the rules of MCCG, are similar to the unifiability check in our type hierarchy framework. In order to handle morphosyntactic parsing with type hierarchy, we should treat morphosyntactic types as feature values and find a mapping from the lattice of diacritics to some type hierarchy. A direct way to do this is to put the morphosyntactic types with $\lhd$ modality in a type hierarchy with the topology of the lattice of diacritics. Figure 4.2 is the type hierarchy version of Figure 4.1, where $<a$ is the featural type corresponding to $\overset{a}{\lhd}$. Unification check in this hierarchy is tantamount to the applicability check of the rules in (23), that is, if $a \lhd b$ in the morphosyntax lattice, then $<a$ can unify with $<b$ according to the corresponding type hierarchy.[4]

The remaining issue is to map the morphosyntactic types with $\bowtie$ modality to the correct places in the feature types hierarchy. This is achieved by creating a feature type $=a$ for each morphosyntactic type $\overset{a}{\bowtie}$, and making it the immediate subtype of $<a$ and the immediate supertype of the bottom type (Figure 4.3). We can now

[4] But note that $b \lhd a$ has the same unification condition as $a \lhd b$, but it is not the same constraint as $a \lhd b$, hence there is a need for separation of features and modalities. We say more on this later.

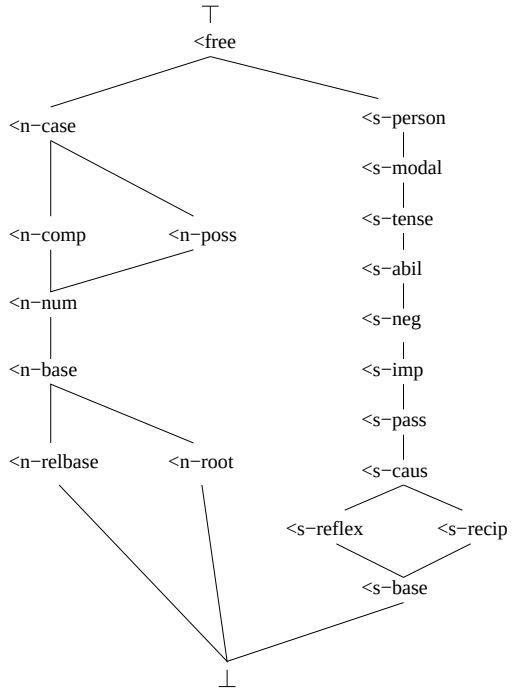


Figure 4.2: Morphosyntactic feature types hierarchy for Turkish (without ⋈ modalities).

create a new feature in the feature structure of each atomic category, specifying the morphosyntactic type of that category. (25) shows some example atomic feature structures for Turkish. (26a) is an example derivation with these categories that corresponds to the MCCG version in (26b).

(25) a. defter := $\begin{bmatrix} \mathsf{n} & \\ \mathsf{NUM} & \mathit{sing} \\ \mathsf{CASE} & \mathit{nom\_or\_acc} \\ \mathsf{MORPH\text{-}SYN} & <\mathit{n\text{-}base} \end{bmatrix}$

b. ler := $\begin{bmatrix} \mathsf{n} & \\ \mathsf{NUM} & \mathit{plu} \\ \mathsf{MORPH\text{-}SYN} & <\mathit{n\text{-}num} \end{bmatrix} \backslash \begin{bmatrix} \mathsf{n} & \\ \mathsf{NUM} & \mathit{sing} \\ \mathsf{MORPH\text{-}SYN} & <\mathit{n\text{-}base} \end{bmatrix}$

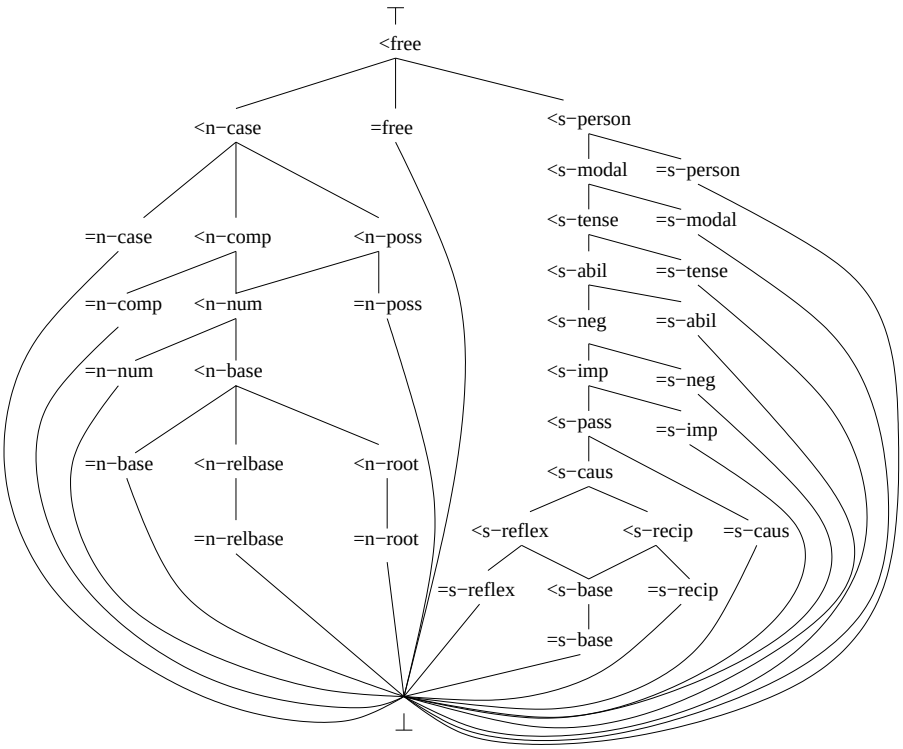


Figure 4.3: Morphosyntactic feature types hierarchy for Turkish (complete).

$$\text{c. i} := \begin{bmatrix} \mathsf{n} & \\ \mathsf{CASE} & \mathit{acc} \\ \mathsf{MORPH\text{-}SYN} & <\mathit{n\text{-}case} \end{bmatrix} \backslash \begin{bmatrix} \mathsf{n} & \\ \mathsf{MORPH\text{-}SYN} & <\mathit{n\text{-}poss} \end{bmatrix}$$

(26) a. defter ler i

$$\dfrac{\dfrac{\mathsf{n}_{<\mathrm{n-base}} \quad \mathsf{n}_{<\mathrm{n-num}} \backslash \mathsf{n}_{<\mathrm{n-base}}}{\mathsf{n}_{<\mathrm{n-num}}}< \quad \mathsf{n}_{<\mathrm{n-case,acc}} \backslash \mathsf{n}_{<\mathrm{n-poss}}}{\mathsf{n}_{<\mathrm{n-case,acc}}}<$$

b. defter ler i

$$\dfrac{\dfrac{\overset{b}{\triangleleft}\mathsf{n} \quad \overset{n}{\triangleleft}\mathsf{n} \backslash \overset{b}{\triangleleft}\mathsf{n}}{\overset{n}{\triangleleft}\mathsf{n}}< \quad \overset{c}{\triangleleft}\mathsf{np}_{\mathrm{acc}} \backslash \overset{o}{\triangleleft}\mathsf{n}}{\overset{c}{\triangleleft}\mathsf{np}_{\mathrm{acc}}}<$$

4.2 Unary Modalities and Feature Structures

We have presented a complete algorithm in Section 4.1.2 to convert a morphosyntactic lattice of diacritics into a feature types hierarchy. This conversion enables us to simulate MCCG modalities only by featural unification without the use of extra mechanisms in rules. However, there is a significant difference between MCCG and its featural simulation. Unification is a commutative operation while the $\alpha_2\Box_1\alpha_1$ constraint in (23) is not. In morphosyntactic parsing, the main functor's argument specification (i.e. $\Box_1$ of $\overset{\alpha_1}{\Box_1}\ Y$ in (23)) determines the applicability of the rules. In other words, when we write $X/\Box Y$ for some modality $\Box$, the functor X puts a basic category constraint on its argument Y via the modality, hence in a surface configuration such as $X/\Box_1 Y$ — $\Box_2 Y$, where (>) rule can be used, the functor's control over the argument is indicated by $\Box_1$, since $\Box_2 Y$ does not "know" that it is an argument of the function X/Y.[5] The functor/argument role of a category is dynamic since it may change depending on the context or as a result of a rule application. Since there is no way of featurally specifying whether an atomic category is an argument of a complex category or not, we cannot embed such an information in the feature structures. Therefore, when we embed the unary modalities in the feature structures, $X/\Box_1 Y$ — $\Box_2 Y$ configuration becomes X/Y — Y, where the functor/argument discrimination of two Y's is lost. So, the syntactic coverage of two systems are different if $\alpha_1\Box_1\alpha_2$ holds, but $\alpha_2\Box_1\alpha_1$ does not. Although no rule application should occur in this case, the unification of featural versions of these two types would succeed. Hence, our featural version of MCCG overgenerates in some cases (27a) where MCCG correctly

[5] This discussion presumes the existence of argument-taking entities, i.e. functions, in the lexicon. Verbs are typical functions. For example, the category $\mathsf{s}\backslash\mathsf{np}_{\mathrm{nom}}\backslash\mathsf{np}_{\mathrm{acc}}$ for the transitive verb imposes a constraint on its first argument that it be an accusative np. But the $\mathsf{np}_{\mathrm{acc}}$ category, for example *adamı* (man-ACC), does not impose a constraint since it is not a function, but simply describes the fact that the noun is marked accusative.

eliminates (27b). Featural mapping of a MCCG lexicon generates a superset of the language generated by MCCG.

(27) a. *defter i ler

$$\dfrac{\dfrac{\mathsf{n}_{<\mathrm{n-base}} \quad \mathsf{n}_{<\mathrm{n-case,acc}} \backslash \mathsf{n}_{<\mathrm{n-poss}}}{\mathsf{n}_{<\mathrm{n-case,acc}}}< \quad \mathsf{n}_{<\mathrm{n-num}} \backslash \mathsf{n}_{<\mathrm{n-base}}}{\mathsf{n}_{<\mathrm{n-num}}}<$$

since $<$*n-case* can unify with $<$*n-base*.

b. *defter i ler

$$\dfrac{\dfrac{{}^{b}\!\lhd \mathsf{n} \quad {}^{c}\!\lhd \mathsf{np}_{\mathrm{acc}} \backslash {}^{o}\!\lhd \mathsf{n}}{{}^{c}\!\lhd \mathsf{np}_{\mathrm{acc}}}< \quad {}^{n}\!\lhd \mathsf{n} \backslash {}^{b}\!\lhd \mathsf{n}}{}<$$

***since $c \lhd b$ is not satisfied.**

This is a significant linguistic result saying that inflectional morphology can be captured better by constraint-based mechanisms (MCCG), rather than by unification-based features alone (featural type hierarchy). The functor/argument distinction, which seems crucial for inflectional morphology, cannot be specified by the classical feature structure framework.

There are two options for someone who likes to get rid of either the unary modalities or the feature types hierarchy. First option is to define unary modalities as features like we have shown in Section 4.1.2, and simulate the constraints defined on the modalities with type hierarchy. This choice has the obvious flaw of overgeneration. Second option could be to define the features in the form of modalities somehow. However, such a constraint-based approach to features brings an unnecessary power and complicates the specification and the unification of featural values such as case and person, which have always been treated as simple valued (typed) features in all linguistic frameworks.[6]

[6] Heylen (1999) proposes multi-dimensional modalities to represent the features of a category. In this system, categories are decorated with multiple unary modalities, e.g. $\Box_3\Box_{sg}\Box_{nom}\mathsf{np}$. Handling

This discussion brings us to our initial question of "what exactly is a CCG atomic category." Some linguistic phenomena show that both unary modalities and typed feature structures are necessary in formalizing atomic categories.[7] For a complete theory of natural languages, combinatory lexicons should combine both mechanisms in defining the atomic categories.

these modalities requires extra mechanisms such as reordering via structural rules, and modifications to the combinatory rules. We avoid such unnecessary mechanisms and modifications to the core CCG framework in this thesis. Besides, although every simple feature can be viewed as a unary modality, not every unary modality is a simple feature in the sense of unification-based frameworks.

[7] Our aim here is to reach a complete theory of a transparent interface of inflectional morphology, syntax and semantics. If one chooses to ignore morphology or view it as a separate mechanism at a lower level than syntax, he could get rid of the unary modalities. However, this approach creates some problems in semantic bracketing and lexicon design. See (Bozsahin, 2002) for further discussion.

CHAPTER 5

DESIGN AND IMPLEMENTATION OF TYPES AND MODALITIES

5.1 OpenCCG Project

OpenCCG is an open source natural language project that aims to build a realistic universal CCG parser.[1] It is designed not only for CCG researchers and students but also to compete with other natural language tools in efficiency and accuracy.

OpenCCG is a Java-based system that uses Cocke-Kasami-Younger (CKY) chart parsing algorithm and builds both the semantic and syntactic structure of a given sentence. It is not designed for a particular language. The user designs the lexicon of the desired language and activates the universal rules of CCG required for parsing that language's sentences. The rest is handled by the system automatically when the user inputs her sentences.

OpenCCG adopts the idea of representing atomic categories as feature structures, which is the critical assumption for the work in this thesis. All of the input files

[1] For the details of OpenCCG project and a downloadable copy, see project's homepage: http://openccg.sourceforge.net/.

(lexicon, rules, etc.) in OpenCCG are in XML format. Figure 5.1 shows an example lexicon entry where the atomic category n is specified. This corresponds to the feature structure (28a), ignoring the semantic part. A feature value beginning with a capital letter indicates a variable, which means that the corresponding feature is underspecified. Lexical entries are written in the *morph.xml* file with references to the atomic categories in the lexicon file. Entries in the lexicon file are only skeletons of feature structures to be filled by macros in the morphs file. Figure 5.2 shows an example lexical entry for the English word *ball*. It includes the *sg* macro, which assigns the NUM feature the value *sg*. As a result, the information in Figure 5.1 and Figure 5.2 together gives us the feature structure (28b) for the word *ball*.

```
<family pos="N" name="Noun">
    <entry name="Primary">
      <atomcat type="n">
        <fs id="2">
          <feat attr="num">
            <featvar name="NUM"/>
          </feat>
          <feat attr="index">
            <lf>
              <nomvar name="X"/>
            </lf>
          </feat>
        </fs>
        <lf>
          <satop nomvar="X">
            <prop name="[*DEFAULT*]"/>
          </satop>
        </lf>
      </atomcat>
    </entry>
  </family>
```

Figure 5.1: A fragment of an OpenCCG lexicon file (lexicon.xml).

```
<entry word="ball" pos="N" macros="@sg"/>

<macro name="@sg">
    <fs id="2" attr="num" val="sg"/>
</macro>
```

Figure 5.2: A fragment of an OpenCCG morphs file (morph.xml).

(28) a. $\begin{bmatrix} \mathsf{n} \\ \mathsf{NUM} \quad \mathit{NUM} \end{bmatrix}$ b. $\begin{bmatrix} \mathsf{n} \\ \mathsf{NUM} \quad \mathit{sg} \end{bmatrix}$

An example command line parsing session is shown in Figure 5.3, where *tccg* is the name of the top class that invokes the OpenCCG system. We keep the example short here, but the system can be set to show the semantic expressions and feature values of each syntactic category.

5.2 Implementation of the Type Hierarchy

Before the implementation of the type hierarchy, all of the syntactic feature values in OpenCCG were strings, which were unified with simple string matching. To incorporate the type hierarchy into the system, first we added a new file, *types.xml*, where the hierarchy is specified. Figure 5.4 shows an example type hierarchy file and the graphical notation of the hierarchy it specifies. Each line contains the name and the immediate supertype(s) of a type.[2] Types can be entered in any order in the file. *top* is a built-in type which is the supertype of all types. All type hierarchies are connected to *top* type so that we have a big final hierarchy for all of the features of the language.

Types are indexed in breadth-first order as in Figure 5.4. We follow the greatest lower bound (GLB) algorithm described in (Aït-Kaci et al., 1989), which uses bit-

[2] This file format convention is similar to the LKB system of Copestake (1992), a knowledge based system primarily designed for unification based formalisms such as HPSG.

```
$ tccg
Loading grammar from URL: file:/home/gunes/openccg/grammars/turkish/grammar.xml

Enter strings to parse.
Type ':r' to realize selected reading of previous parse.
Type ':h' for help on display options and ':q' to quit.

tccg> cocuk kitab i okudu
1 parse found.

Parse 1: s
------------------------------
(lex) cocuk :- n
(>T) cocuk :- s$1/i(s$1\in)
(lex) kitab :- n
(lex) i :- n\*n
(<) kitab i :- n
(>T) kitab i :- s$1/i(s$1\in)
(lex) okudu :- s{\.n\.n}
(>) kitab i okudu :- s\.n
(>) cocuk kitab i okudu :- s

tccg>
```

Figure 5.3: An example command line parsing session in OpenCCG.

vectors in lattice operations, to find the highest common subtype of two types in unification. While initializing the system, we take the reflexive and transitive closure of the parent-child relation to find the all subtypes of a type. Subtypes information of a type is stored in a bit-vector in which the corresponding bits are set (Table 5.2). When unifying two types, bit-vectors are ANDed with each other. The index of the resultant type is equal to the index of the leftmost set bit of the resultant bit-vector. For instance, when we unify the types **e** and **f** in the example hierarchy, we AND the bit-vectors 001000101 and 000100101, and find the resultant vector 000000101. The index of the leftmost set bit of this vector is 6, which means the result of the unification is the type **g** (cf. Table 5.2).

A singleton object is created for each type, containing the name, index and the

```
<?xml version="1.0" encoding="UTF-8"?>
<types>
<type name="a"/>
<type name="b" parents="a"/>
<type name="c" parents="a"/>
<type name="d" parents="b c"/>
<type name="e"/>
<type name="f"/>
<type name="g" parents="e f"/>
<type name="h" parents="g"/>
</types>
```

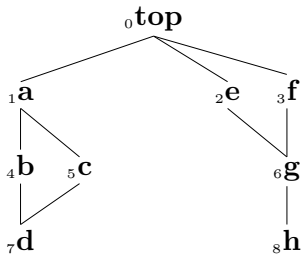


Figure 5.4: An example type hierarchy and its specification in OpenCCG.

subtypes bit-vector. If an unknown type (a type that does not have an entry in the *types.xml* file) is encountered in the lexicon, a new object with a new index is created for it. The only subtype of an unknown type is itself, that is, it can only unify with itself. In other words, unification for unknown types is just simple name matching as in the previous versions of OpenCCG. This makes our system backward compatible in the sense that the lexicons written for the previous versions of OpenCCG can also function in our new version even if the user does not provide a type hierarchy file.

To change the feature values from strings to type object references, we had to make some changes in the core OpenCCG source code as well as adding new classes. Since type objects are singletons, two features in separate feature structures point to the same object if they have the same type. Unification with bit-vectors is as efficient as string matching so that there is no observable difference in parsing times of our system and the previous version of OpenCCG.

Table 5.1: Subtypes bit-vectors for the types in Figure 5.4.

index	type	subtypes
0	top	111111111
1	a	010011010
2	e	001000101
3	f	000100101
4	b	000010010
5	c	000001010
6	g	000000101
7	d	000000010
8	h	000000001

5.3 Implementation of MCCG

We have also integrated the unary modalities of MCCG into OpenCCG. The lattice of diacritics is specified in an XML file whose format is similar to the format of the *types.xml* file in Figure 5.4. Since morphosyntactic modalities are not features, we have created a new item in the atomic category format of the *lexicon.xml* file, that specifies the morphosyntactic type for the corresponding atomic category (compare Figure 5.1 with Figure 5.5).

We have parsed the same test cases for the Prolog implementation of MCCG described in (Bozsahin, 2002) with our Java implementation. A sentence with 10 lexical items is parsed in less than half a second. An example with 21 morphemes is parsed in around 2 seconds, and the longest example with 37 morphemes is parsed in 15 seconds, which is two or three times faster than the Prolog version. The current version of the implementation stores the hierarchy of diacritics in simple lists. The performance is expected to get better when we use bit-vectors instead, like we have done in the implementation of the featural type hierarchy.

```
<family pos="N" name="Noun">
    <entry name="Primary">
      <atomcat type="n">
        <morph-syn m-mod="uptoandequals" diacritic="n-base"/>
        <fs id="2">
          <feat attr="num">
            <featvar name="NUM"/>
          </feat>
          <feat attr="index">
            <lf>
              <nomvar name="X"/>
            </lf>
          </feat>
        </fs>
        <lf>
          <satop nomvar="X">
            <prop name="[*DEFAULT*]"/>
          </satop>
        </lf>
      </atomcat>
    </entry>
  </family>
```

Figure 5.5: An OpenCCG lexicon file entry with morphosyntactic modality.

CHAPTER 6

CONCLUSION

The very internal structure of CCG categories are often ignored in the literature although the mechanisms functioning over them are deeply investigated. We have proposed a restricted form of typed feature structures to represent CCG atomic categories.

We make use of the type hierarchy mechanism of feature structures for a better control over the unification of lexicalized features. We have shown some cases, such as person and case marking features, where this mechanism produces efficient solutions and minimizes the lexicon.

We have tried to model morphosyntactic CCG (MCCG), which presents a transparent interface of inflectional morphology and syntax, in our type hierarchy system. Although we have reached a simulation of MCCG, we are unable to construct a one-to-one mapping from MCCG to our system. The result has turned out to be an overgenerating system. This supports the fact that inflectional morphology is basically a constraint-based system and not necessarily suitable to unification-based formalisms. Therefore, morpheme-based parsing still needs a mechanism other than

feature structure unification (such as MCCG). As a result, we reach an atomic category representation structure that is a combination of three mechanisms, all of which are necessary in explaining linguistic phenomena: the feature structure view described in (Baldridge, 2002), the featural type hierarchy framework proposed in this thesis, and the unary modalities proposed in the morphosyntactic framework of Bozsahin (2002). We avoid leaving out any of these mechanisms since this will either require modifications in the core machinery of CCG or lead to a less refined theory of natural languages.

The implementation of our system is integrated into the open source CCG project, OpenCCG, rather than being a standalone system written just to serve for this thesis. Since OpenCCG is well-known and rapidly spreading in the CCG community, this decision coincides with our initial purpose of making the type hierarchy and MCCG apparatus available in CCG as part of its representational basis. The result is an easily maintained parsing software that is far more efficient than Prolog based systems and can compete with any other successful natural language parsing systems.

REFERENCES

Aït-Kaci, Hassan, Robert Boyer, Patrick Lincoln, and Roger Nasr. 1989. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems*, 11(1):115–146, January.

Ajdukiewicz, Kazimierz. 1935. Die syntaktische Konnexitat. *Studia Philosophica*, 1:1–27. English translation in Storrs McCall (ed), *Polish Logic 1920-1939*, Oxford University Press, pp. 207-231.

Baldridge, Jason. 2002. *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh.

Bar-Hillel, Y., C. Gaifman, and E. Shamir. 1964. On categorial and phrase structure grammars. In Y. Bar-Hillel, editor, *Language and Information. Selected Essays on their Theory and Application*. Addison-Wesley, Reading, MA, pages 99–115.

Bar-Hillel, Yehoshua. 1953. A quasi-arithmetical notation for syntactic description. *Language*, 29.

Beavers, John, 2003. *Documentation: A CCG Implementation for LKB*. April Draft.

Bozsahin, Cem. 2002. The combinatory morphemic lexicon. *Computational Linguistics*, 28:145–186.

Bresnan, Joan. 2001. *Lexical-Functional Syntax*. Oxford: Blackwell.

Carpenter, Bob. 1992. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science 32. Cambridge University Press.

Chomsky, Noam. 1975. *Reflections on Language*. New York: Pantheon Books.

Chomsky, Noam. 1981. *Lectures in government and binding*. Studies in generative grammar 9. Dordrecht: Foris.

Copestake, Ann. 1992. The ACQUILEX LKB: representation issues in semi-automatic acquisition of large lexicons. In *Proceedings of the Third Conference on Applied Natural Language Processing*, pages 88–95, Trento, Italy. Association for Computational Linguistics. 31 March - 3 April.

Heylen, Dirk. 1999. Underspecification in type-logical grammars. *Lecture Notes in Computer Science*, 1582:180–199.

Pollard, Carl and Ivan Sag. 1987. *Information-Based Syntax and Semantics, Volume 1: Fundamentals*. Stanford, CA: CSLI.

Pollard, Carl and Ivan Sag. 1994. *Head-Driven Phrase Structure Grammar*. Chicago: University of Chicago Press. Draft distributed at the Third European Summer School in Language, Logic and Information, Saarbrücken, 1991.

Shieber, Stuart. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343.

Steedman, Mark. 2000. *The Syntactic Process*. Cambridge, Massachusetts: The MIT Press.

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

Wood, Mary McGee. 1993. *Categorial Grammars*. London: Routledge.