A SIGN-BASED PHRASE STRUCTURE GRAMMAR FOR TURKISH

by
Onur Tolga Şehitoğlu

January, 1996
Middle East Technical University
ANKARA

In partial fullfilment of the requirements
for the degree of
Master of Science
in
The Department of Computer
Enginnering

Abstract

A Sign-Based Phrase Structure Grammar for Turkish

Şehitoğlu, Onur Tolga
MS., Department of Computer Engineering
Supervisor: Assist. Prof. Dr. Cem Bozşahin
January 1996, 97 pages

This study analyses Turkish syntax from an informational point of view. Sign based linguistic representation and principles of HPSG (Head-driven Phrase Structure Grammar) theory are adapted to Turkish. The basic informational elements are nested and inherently sorted feature structures called signs.

In the implementation, logic programming tool ALE —Attribute Logic Engine— which is primarily designed for implementing HPSG grammars is used. A type and structure hierarchy of Turkish language is designed. Syntactic phenomena such as subcategorization, relative clauses, constituent order variation, adjuncts, nominal predicates and complement-modifier relations in Turkish are analyzed. A parser is designed and implemented in ALE.

Keywords: syntax, Turkish Grammar, parsing, phrase structure

Öz

Türkçe İçin İm Temelli Öbek Yapısal Sözdizimi

Şehitoğlu, Onur Tolga
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez yöneticisi: Yrd. Doç. Dr. Cem Bozşahin
Ocak 1996, 97 sayfa

Bu çalışmada, Türkçe sözdizimi bilgiye dayalı bir bakış açısıyla değerlendirilmiştir. İme dayalı dilbilimsel gösterim ve HPSG (Baş-sürümlü Öbek Yapısal Dilbilim) kuramı Türkçe'ye uyarlanmıştır. HPSG, dildeki nesnelerin bilgisel içerikleriyle gösterimine dayanan çağdaş bir sözdizimi ve anlambilim kuramıdır. Temel bilgi öğesi im denilen içiçe ve kalıtsal türlendirilmiş özellik yapılarıdır.

Uygulamada mantık programlama dili olarak özellikle HPSG uygulamaları için tasarlanmış olan ALE kullanılmıştır. Türkçe'deki dil öğelerinin bir tür ve yapı tanımı yapılmıştır. Altulamlama, yan cümleler, öbek sıra değişimi, tümleç-niteleyen ilişkileri ve ortaç yapıları ALE'de çalışan bir ayrıştırıcı ile tasarlanmış ve uygulanmıştır.

Anahtar Kelimeler: sözdizimi, Türkçe Dilbilgisi, ayrıştırma, öbek yapısı

Acknowledgments

I would like to thank NATO TU-LANGUAGE and TÜBİTAK EEEAG-90 projects for providing development environment and research materials. Hardware and software resources of the laboratory established by NATO (LcsL) have been used in all stages of the preperation of the thesis.

I would like to thank Dr. Cem Bozşahin and Elvan Göçmen for their contributions with corrections, discussions and especially for Turkish syntax chapter.

Thanks are also due to all of the friends and family for their encouragement, support and friendship during the preperation of the thesis.

Contents

List of Tables

List of Figures

List of Abbreviations

1Sg, 2Sg, 3Sg Agreement suffixes first, second and third person singular

1Pl, 2Pl, 3Pl Agreement suffixes first, second and third person plural

1SP, 2SP, 3SP Possessive suffixes first, second and third person singular

1PP, 2PP, 3PP Possessive suffixes first, second and third person plural

Abl Ablative Case

Acc Accusative Case

Dat Dative Case

Loc Locative Case

Ins Instrumental/commitative Case

Gen Genitive Case

Equ Equitative Case

Mun Munitive Case

Rlvz Noun Relativizer

Cop Copula Suffix

Aux Auxilary Suffix

Pres Present Tense (`-Ar`)

Prog Progressive Tense (`-Hyor`)

Past Past Tense (`-dH, -mHş`)

Fut Future Tense (`-(y)AcAk`)

Asp Aspect markers (`-dH, -mHş, -sA`)

Pass Passive Suffix

Caus Causative Suffix

Neg Negation suffix

Ques Question suffix

Part Complement Participle Suffix (`-DHk, -(y)AcAk`)

Inf Infinitive Suffix (`-mAk`)

Ger Gerundive Suffix (`-mA, -Hş`)

Rel Relative Participle Suffix (`-An, -DHk, -(y)AcAk`)

Cond Conditional Suffix (`-(y)sA`)

Adv Adverbial Suffix (`-ken`)

Nec Necessity Suffix (`-mAlH`)

Chapter 1

INTRODUCTION

This study has two purposes: first, to study Turkish grammar in light of the Head-driven Phrase Structure Grammar (HPSG) formalism, and second, to come up with a computational model of the languages based on the HPSG principles.

Turkish grammar has been analyzed from the perspective of linguistic theories such as Transformational Grammar [18], Government-Binding, and Functional Grammar [23]. Lewis [16], Underhill [27], Banguoğlu [1], and Şimşek [26] are also good sources in the traditional descriptive style. However these studies do not shed any light on how a computational model can be constructed from the linguistic description.

Recent linguistic theories, such as HPSG [20, 21] and Lexical Functional Grammar (LFG) [4], differ from the earlier ones in their rigorous definitions and incorporation of ideas from computer science and artificial intelligence. These ideas range from type-theory in programming languages to unification and knowledge representation. Due to the formal representations, there are meta-tools for constructing computational models from formal descriptions, such as Attribute Logic Engine (ALE)[5], CUF [6], and Typed Feature System (TFS)[15]; Tomita's parser for LFG [19].

This work is one of the early attempts, together with LFG [10] and Categorial Grammar Models [13, 3] to study Turkish from the perspective of modern linguistic theories. Our motivation was to design a parser based on the principled account of Turkish syntax in the HPSG framework. It makes use of the ALE formalism to model HPSG-style definitions.

HPSG makes universal claims about human languages. The main point is that, although the grammars of languages differ in terms of phrase structure and how grammatical functions are realized, certain principles always hold accross the languages. An example of such a principle roughly states that the 'head' of a phrase plays the most prominent role in propagating the syntactic and semantic properties of a phrase. Thus, an HPSG grammar for a language is a collection of specifications for phrase structure, realization of principles in the language, and the signature of the language in terms of linguistic features. This division of linguistic description is also reflected in the computational meta-level tools for writing HPSG-style grammars. We hope that these kinds of experiments point out the advantages and disadvantages of such frameworks for underanalyzed language including Turkish.

We aimed to develop a competence grammar rather than a performance grammar for Turkish. This requires postulating the Turkish realizations of HPSG principles and their computational counterparts. We chose to provide a breadth of coverage in terms of lexical types and phrases instead of a comprehensive study with a large lexicon. Moreover, we have implemented some of the morphosyntactic operations (eg. case marking, possessives, relativization) in the lexicon.

The remainder of the thesis is organized as follows: Chapter 2 introduces the basic concepts of HPSG. Chapter 3 is an outline of Turkish syntax. Chapter 4 describes HPSG model of Turkish and Chapter 5 elaborates on the implementation.

Chapter 2

HEAD-DRIVEN PHRASE STRUCTURE GRAMMAR

HPSG (Head-driven Phrase Structure Grammar) was introduced by Pollard and Sag[20] as an information-based theory of syntax and semantics . HPSG views a human language as a device used for exchanging information, and tries to explain the relation between the phonetic form of a word or a phrase, its grammatical structure, and its informational content. In HPSG, a natural language is defined as a system of correspondences between certain kind of utterances and certain kinds of objects and situations in the world.

HPSG synthesizes most of the recent (principally nonderivational) syntactic theories such as Categorial Grammar (CG), Generalized Phrase Structure Grammar (GPSG), Arc Pair Grammar (APG), and Lexical Functional Grammar (LFG); semantic theories like Situation Semantics, and some basic concepts of computer science (data type theory, knowledge representation, unification).

In HPSG, every linguistic component (words, phrases, rules, etc.) is analyzed with a perspective of the information it provides to the speaker of the language. This information may include not only the syntactic features of the component, but also its grammatical information, semantic content and its background semantics.

HPSG is a system based on *signs*. Any structural element (words and phrases), and principles defining the language are modeled by *sorted feature structures* (ie. feature structures with an associated type or sort) and constraints and operations defined on them. As being one of the most recent examples of the family of the *unification based* grammar theories [25], the most fundamental operation of HPSG is unification, which combines a set of compatible feature structures, and returns a minimum informative feature structure containing all information present in the operands. Phonetic, syntactic and semantic information coded in the lexicon and information coming from other resources like lexical rules, universal and language specific principles of well-formedness, are combined by unification.

Similar to the majority of the contemporary linguistic theories, HPSG defines a language by a finite set of recursively applicable rules which yields the judgment of grammaticality. Basically, principles are divided into two categories: 1) Universally applicable basic set of constraints such as *head feature principle* and *subcategorization principle*; and types of phrases available in any human language. 2) Language specific principles of phrases lexicon itself and further articulation and specification of the principles of the universal grammar such as constituent order[21].

One of the distinctive aspects of HPSG is that it not only models the language syntactically, but also concerns itself with the interactions between all kinds of information of a linguistic component. Both syntactic and semantic information content of a sign is considered. Situation Semantics and Relational Theory of Meaning are chosen for semantic modeling.

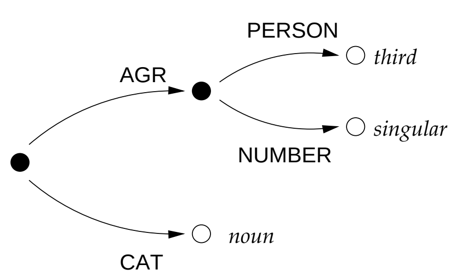
2.1 Feature Structures

HPSG, like other unification-based formalisms, uses recursively embedded feature-value pairs representing linguistic objects. Feature structures have different names in each theory: f-structures in LFG, feature bundles, feature matrices or categories in GPSG, etc. Feature structures are informational objects that consist of feature (attribute)-value pairs. Usually feature structures are represented by *attribute-value matrices* (AVM's). For example:

(1)
$$\begin{bmatrix} \textsf{PHON} & \textit{“kedi”} \quad \% \text{ cat} \\ \textsf{CAT} & \textit{noun} \\ \textsf{AGR} & \begin{bmatrix} \textsf{PERSON} & \textit{third} \\ \textsf{NUMBER} & \textit{singular} \end{bmatrix} \end{bmatrix}$$

In (1), features PHON, CAT and AGR are defined where value of PHON is *“kedi”*, CAT is *noun* and value of AGR is another feature structure with features PERSON and NUMBER which have values *third* and *singular* respectively. As an alternative, feature structures can be represented in graph notation where nodes are the intermediate feature structures, vertices are attributes, and values are sink nodes.

(2)



As a fundamental property, feature structures can be recursively embedded. Value of an attribute can be an atomic value or another feature structure. To represent values embedded in feature structures, "path of attributes" notation is used as a shorthand. A path is an ordered sequence of attributes separated by '|' to reach the value. In the example (2), AGR|PERSON has the value *third*, AGR|NUMBER is *singular*.

A relation defined on feature structures is the *subsumption* relation. When a feature structure A is subsumed by another feature structure B, A is equally or more informative than B. In other words, it contains all of the information provided by B and possibly more. It is often said A *extends* B or B *subsumes* A and written as $A \preceq B$. It indicates that any object described by B can be appropriately described by A. When a feature structure has no information, it *subsumes* every feature structure. It can describe any object whatsoever. This structure is the root element of the subsumption ordering (called *Top*), shown as $\top$:

$\top = [\quad]$

Subsumption relation defines a partial ordering between information structures. It has the properties of reflexivity ($\forall A, A \preceq A$), transitivity ($A \preceq B$ and $B \preceq C$ then $A \preceq C$), and antisymmetricity ($A \preceq B$ and $B \preceq A$ then $A = B$). For subsumption relation to hold between two feature structures, they should have compatible types and compatible values in the corresponding attributes. In the example below, feature structures have incompatible values so, $A \not\preceq B$ and $B \not\preceq A$.

(3) $$A = \begin{bmatrix} \textsf{PERSON} & \textit{third} \end{bmatrix}, \qquad B = \begin{bmatrix} \textsf{PERSON} & \textit{first} \\ \textsf{NUMBER} & \textit{singular} \end{bmatrix}$$

Another important property of feature structures is *structure sharing*. Two attribute paths in a feature structure may describe the same object. This implies the *token identity* which should not be confused with the *structural identity* where only type structure and feature values are equal. Usually structure shared object are denoted by *tags* (boxed numbers).

(4) a. $$\begin{bmatrix} \textsf{HD-DTR} & \begin{bmatrix} \textsf{CAT} & \textit{verb} \\ \textsf{AGR} & \boxed{1} \begin{bmatrix} \textsf{PERSON} & \textit{first} \\ \textsf{NUMBER} & \textit{plur} \end{bmatrix} \end{bmatrix} \\ \textsf{SUBJ-DTR} & \begin{bmatrix} \textsf{CAT} & \textit{noun} \\ \textsf{AGR} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

b. $$\begin{bmatrix} \textsf{HD-DTR} & \begin{bmatrix} \textsf{CAT} & \textit{verb} \\ \textsf{AGR} & \begin{bmatrix} \textsf{PERSON} & \textit{first} \\ \textsf{NUMBER} & \textit{plur} \end{bmatrix} \end{bmatrix} \\ \textsf{SUBJ-DTR} & \begin{bmatrix} \textsf{CAT} & \textit{noun} \\ \textsf{AGR} & \begin{bmatrix} \textsf{PERSON} & \textit{first} \\ \textsf{NUMBER} & \textit{plur} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

In (4a), HD-DTR|AGR attribute shares the same object with the SUBJ-DTR|AGR attribute. Although the paths in question have the same value, the source of the values may be different (i.e., not token identical) contain the same agreement in (4b). Intuitively, it is clear that the structure shared version is more informative than the other; it is subsumed by the other. Similarly, the effect of structure sharing is reflected in the formal definition of the subsumption:

(5) if A and B are atomic, then $A \preceq B$ iff $A = B$.

else, $A \preceq B$ holds iff,

(i) for every path in B, same path exists in A and its value is subsumed by the value in B.

(ii) for every structure sharing path in B, same path is structure sharing in A.

Perhaps the most important operation on feature structures is the *unification*, which constructs a base to a group of linguistic theories. Unification operation builds a new feature structure which contains all but not more of the information contained in its operand feature structures. For feature structures to be unified, they should have compatible types. Result of unification is the least informative (the most general) feature structure which extends all of the operands. Unification is denoted by the symbol $\wedge$ and if $C = A \wedge B$ then, $C \preceq A$ and $C \preceq B$.

(6) $$\begin{bmatrix} \textsf{CAT} & \textit{noun} \\ \textsf{AGR} & \begin{bmatrix} \textsf{PERSON} & \textit{third} \end{bmatrix} \end{bmatrix} \wedge \begin{bmatrix} \textsf{CAT} & \textit{noun} \\ \textsf{AGR} & \begin{bmatrix} \textsf{NUMBER} & \textit{sing} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \textsf{CAT} & \textit{noun} \\ \textsf{AGR} & \begin{bmatrix} \textsf{PERSON} & \textit{third} \\ \textsf{NUMBER} & \textit{sing} \end{bmatrix} \end{bmatrix}$$

When operands of the unification have incompatible types or values, the resulting feature structure does not exist and unification fails. This is indicated by the symbol '$\perp$' (*bottom*) which represents inconsistent information. As $\top$ is the maximal element in the subsumption ordering, $\perp$ is the minimal element that is subsumed by all feature structures.

Also, unary negation operator $\neg$ and disjunction operator $\vee$ are defined. In negation, $\neg a$ means any value other than a. Similarly, disjunction $a \vee b$ means the attribute may be equal to a or b. Attributes may be *list*-or *set*-valued. Lists are denoted by comma separated values enclosed in angle brackets, $\langle a, b, ...\rangle$. Sets are enclosed in curly braces $\{a, b, ...\}$. List valued attributes are unified by unifying corresponding elements by order. Unification of set values is a more complex operation. For detailed information and formal definition about feature structures, consult Rounds and Kasper[22].

The most significant formal property of HPSG feature structures is that they are **sorted**. Every feature structure has a type (sort), and a subtype relation is defined between these sorts. All defined sort symbols are partially ordered by the subsumption relation.

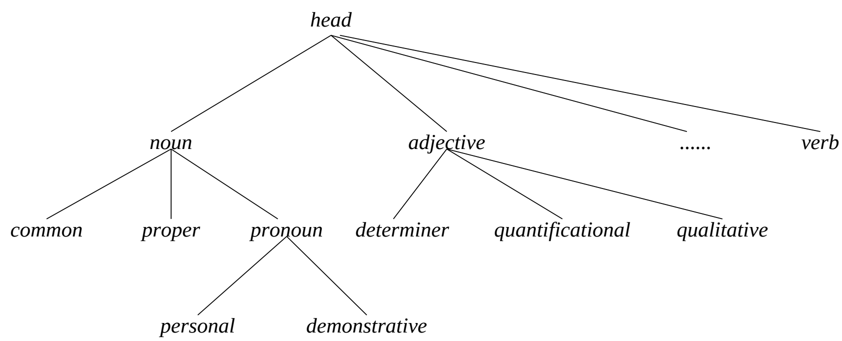


Figure 2.1: Subtype hierarchy for the type defined for HEAD feature

As the second formal property, HPSG feature structures should be *totally well-typed*. For each sort, a set of appropriate features and types is defined, and this set is inherited by the subsorts of the sort. For example, if the CASE feature of sort *case* is defined for the sort *noun* in Figure 2.1, then it is appropriate for the sorts *common, proper, pronoun, personal* and *demonstrative* sorts. Any other feature which is not introduced in the sort is not allowed in the feature structure.

Third, HPSG feature structures should be *sort-resolved* to satisfy the criteria of completeness as models of the linguistic entities. *Sort-resolved* means: for every attribute defined, a sort should be assigned. This sort should be the most specific in the sort ordering (A leaf node in the subtype hierarchy). For example HEAD feature can be assigned *proper* or *common* but not *head* or *noun* which subsume other types in the ordering and actual sort (value) is not clear.

2.2 Sign Structure

Linguistic entities have the general sort *sign*. Information in all intermediate phrases, lexical entries, sentences and even multisentence discourses are described by a corresponding *sign*. The *sign* sort has two subtypes: *word* and *phrase*. *word* describes lexical entries, and *phrase* describes phrasal constructs. *phrase* has an additional feature DTRS (daughters) to represent the phrase structure.

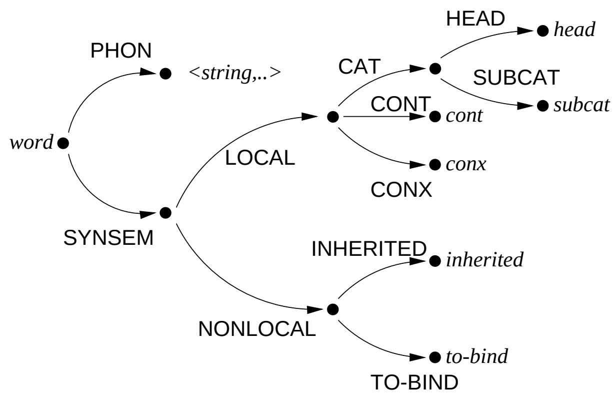


Figure 2.2: Basic Structure of a Lexical Sign (word).

A basic graph briefly describing the structure of a *sign* can be given as in Figure 2.2. All signs should have at least two attributes: PHON and SYNSEM. PHON attribute is a feature representation of the phonetic content of the *phrase* or *word*. Usually, it has a list of strings describing phonological and phonetic structure of the *sign*. SYNSEM attribute contains both syntactic and semantic information of the *sign*. Using SYNSEM instead of two distinct features SYNTAX and SEMANTICS allows packing all information required for subcategorization into one attribute.

SYNSEM value is another structured object which has two attributes LOCAL and NONLOCAL. NONLOCAL represents the information which is not bound to the phrase described by the *sign*. This information is used to handle unbounded dependency constructs like filler-gap dependencies, relative clauses, etc. LOCAL feature describes the local information which consists of the attributes CATEGORY (CAT), CONTENT (CONT) and CONTEXT (CONX).

CAT value includes both the syntactic category of the *sign* and the grammatical arguments it requires. CONT value is the context independent semantic interpretation and semantic contribution of the *sign*.

CONX value contains context-dependent linguistic information such as indexicality, presupposition and conventional implication. The semantic features are not the object of this study so we will not go into details of CONT and CONX features.

CAT attribute consists of two attributes HEAD and SUBCAT. HEAD feature is roughly lexical category (part of speech) of the *sign*. It describes the information to be passed to phrasal projections of the *sign*. Contents of HEAD feature varies according to the category of the *sign*. It typically contains basic features related with the category of the sign e.g. case, agreement, verb form, prepositional form, etc.

SUBCAT (subcategorization) describes the valence of the sign which specifies the group of signs that sign in question requires to become *saturated*. A saturated sign means all the subcategorization requirements are met. Group of signs in SUBCAT feature is described by a list of *synsem* values. *synsem* values are used for identifying the subcategorized objects, so that *sign* can select not only the syntactic category of the complement, but also semantic role and even nonlocal attributes.

The order of the *synsem* values in the SUBCAT list does not correspond to the surface order of the phrase. However, it may define an obliqueness order which can be used to describe the constituent order. For example, in English, linear precedence rules defining surface order is declared by this obliqueness order, as the least oblique element linearly precedes others for non-verbal heads. When the head is a verb, the first oblique element is the subject element which precedes the head. For languages having free constituent order, like Turkish, SUBCAT attribute may have different structure, e.g., unordered list.

2.3 Phrases and Syntactic Structure

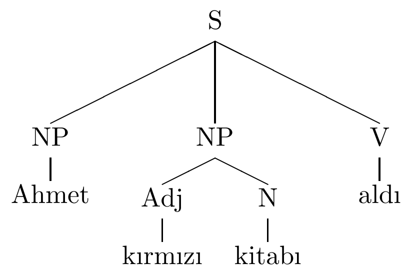
HPSG is a constraint-based theory and constraints are defined by partial descriptions that model linguistic utterances. Descriptions are declarative, order independent and reversible. Judgment of whether a phrase is well-formed or not is done by a set of universal principles and language-specific rules. Universal principles are general constraints on universally available phrase types. The most basic principles in HPSG are **Head Feature Principle** and **Subcategorization Principle**. Language specific phenomenon like *Linear Precedence* (constituent order) is described by a set of language specific constraints and some kind of specialization of universal principles.

As mentioned in the preceding section, a *sign* has two subtypes: *word* and *phrase*. *phrase* has an additional feature DAUGHTERS (DTRS) in which phrase structure is represented. DTRS feature has a value of constituent-structure *(cons-struc)* representing the immediate constituents of the phrase. *cons-struct* may have several subsorts each has characterized by different daughter attribute. The most general sort of *comp-struc* is headed-structure *(head-struct)*.

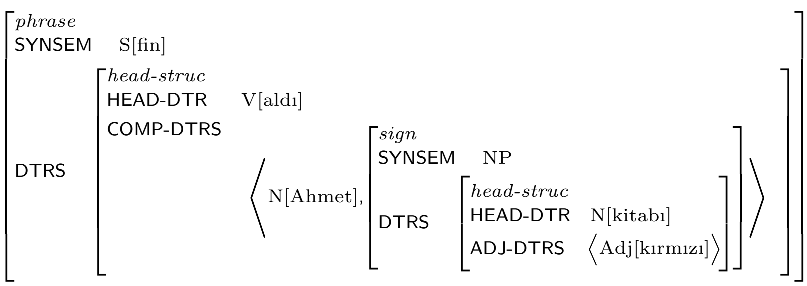
(7) $$\begin{bmatrix} \textit{head-struc} & \\ \textsf{HEAD-DTR} & \textit{sign} \\ \textsf{COMP-DTRS} & \left\langle \textit{sign, ...} \right\rangle \end{bmatrix}$$

Each *head-struc* has one HEAD-DTR attribute and another attribute which is a list of signs which are the sisters of the HEAD-DTR. For example tree and DTRS representation of the sentence "Ahmet kırmızı kitabı aldı." (Ahmet took the red book.) is:

(8) a.

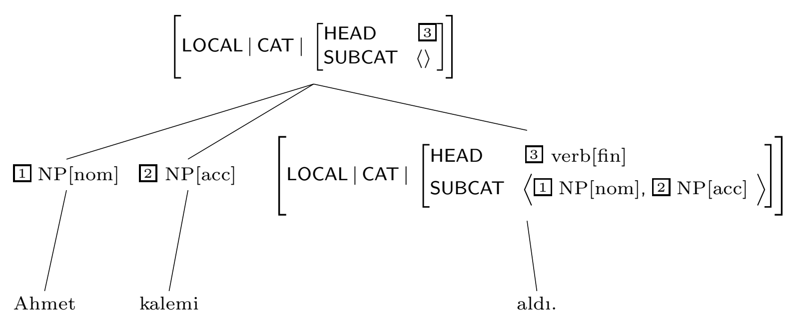


b.



The HEAD value of a phrase is centrally important since it defines the syntactic properties of the mother phrase. For example, the lexical head of a sentence is of the sign *verb*. *verb* combines with its complement sisters and forms a *Verb Phrase (VP)* which takes its syntactic properties from its head daughter (verb). Similarly, verb phrase combines with the subject complement forming a sentence. In other phrase types like Noun Phrase (NP), Prepositional Phrase (PP), Adjective Phrase (AP), HEAD feature is projected —propagated— along the upper phrases until phrase becomes saturated. The key idea behind this projection is the **X-bar** theory[14]. HPSG's **Head Feature Principle** describes this syntactic phenomena which is adopted from the **Head Feature Convention** of GPSG[8].

(9)



Head Feature Principle(HFP) is defined as follows:

(10) The HEAD value of a headed phrase is structure-shared with the HEAD value of the head daughter.
Formally:

$$\begin{bmatrix} \textit{phrase} \\ \text{SYNSEM}\,|\,\text{LOC}\,|\,\text{CAT}\,|\,\text{HEAD} \quad \boxed{1} \\ \text{DTRS}\,|\,\text{HEAD-DTR}\,|\,\text{SYNSEM}\,|\,\text{LOC}\,|\,\text{CAT}\,|\,\text{HEAD} \quad \boxed{1} \end{bmatrix}$$

The other principle which together with HFP describes the basic Immediate Dominance (ID) scheme of HPSG is **Subcategorization Principle**. Subcategorization checks the requirements of the phrasal head to be saturated and allows heads to select its complement sisters by structure sharing the SYNSEM values of the sisters with that in the SUBCAT list. **Subcategorization Principle** is defined as follows:

(11) In a headed phrase, SUBCAT value of the head daughter of the phrase is the concatenation of the SYNSEM values of the complement daughters.
Formally:

$$\begin{bmatrix} \textit{phrase} \\ \text{SYNSEM}\,|\,\text{LOC}\,|\,\text{CAT}\,|\,\text{SUBCAT}\ \boxed{1} \\ \text{DTRS} \quad \begin{bmatrix} \text{HEAD-DTR}\,|\,\text{SYNSEM}\,|\,\text{LOC}\,|\,\text{CAT}\,|\,\text{SUBCAT} \quad \boxed{2} \oplus \ldots. \oplus \boxed{n} \oplus \boxed{1} \\ \text{COMP-DTRS} \quad \left\langle \text{SYNSEM}\ \boxed{2}, \ldots., \text{SYNSEM}\ \boxed{n} \right\rangle \end{bmatrix} \end{bmatrix}$$

Where $\oplus$ is defined to be list concatenation operation.

The **Subcategorization Principle** allows all constraints on the arguments of a phrase to be controlled by an argument. Any kind of argument restriction, complement structure like sentential complements, unbounded dependencies and other constraints can be directly controlled and coded into lexicon. In other words, HPSG crucially relies on the complex descriptions in the lexicon. To deal with the redundancy caused by the complexity of the lexical entries, lexical rules and multiple inheritance hierarchy describing relation between lexical entries can be expressed [20].

Phrase structure rules defining tree structure of phrases are described by immediate dominance (ID) and linear precedence (LP) constraints. There is a general trend in contemporary syntactic theories towards the lexicalization of grammar and elimination of construction-specific rules in favor of schematic immediate dominance templates. These schemata may vary for language-specific phrase types and constituent relations. Examples of typical phrase structures are head-complement, specifier-head, and adjunct-head, conjunct-daughters.

Linear precedence constraints are mostly defined as language-specific rules and constraints on the surface constituent order of the phrases. In English, LP rules are defined on the obliqueness hierarchy of the SUBCAT list. Subject is the least oblique argument of a verb. The direct object and the indirect object come next in the obliqueness order.

Also, least oblique constituents precede the others. LP rules for English can be defined as follows:

(12) 1. Any lexical head sign precedes other signs:
$\left[\text{HEAD-DTR } \boxed{1} \text{ word}\right] \Longrightarrow \left[\boxed{1} \leq \left[\ \right]\right]$

2. Subject complement precedes the Head daughter:
$\left[\text{SUBJ-DTR } \boxed{1}\right] \Longrightarrow \left[\boxed{1} \leq \left[\textit{phrase}\right]\right]$

3. Least oblique elements linearly precede the others:
$\left[\text{COMP-DTRS} \left\langle \ldots,\boxed{1}, \ \ldots,\boxed{2},\ldots \right\rangle\right] \Longrightarrow \left[\boxed{1} < \boxed{2}\right]$
where $\leq$ means *immediately precede* and $<$ means *precede*.

2.4 Lexical Organization

Lexicalization and the use of meta-rules controlled by a set of universal principles results in a few number of simple grammar rules. However, the associated information structures become more complex. Lexical signs in lexicaly-oriented theories should very rich in information content so it is not always possible to enter and maintain a lexicon without any organization.

In HPSG —as the other lexicalized formalisms— it is necessary to organize lexicon such that lexical entries should be represented as compact as possible. Two main devices, lexical type hierarchy and lexical rules, are the basic solutions to redundancy problem in the lexicon.

The main idea behind the lexical type hierarchy is the repetition of information in the lexical items of same category class. Only a small part of a lexical entry carries exceptional information from the other entries having the same category. For example, all nouns have *noun* as the value of the HEAD feature and empty SUBCAT value. All common nouns have *third person* in their agreement. So the idea is to create a hierarchy of types each of that is assigned a set of attribute-value pairs which are inherited along the hieararchy. Lexical entries can be defined by means of these types plus the special features.

Lexical hierarchies solve some portion of the redundancy. However, some specific features of lexical entries may be related to each other by recurrent patterns. These patterns include some derivational and inflectional phenomena in the language like passivization of verbs, case marking, verb inflections, nominalization etc. The solution that has been used by most of the unification-based formalisms is to define functions mapping one class of words to another, called *lexical rules*.

Lexical rules are generally expressed as procedures converting an input form to an output form. So that all inflections and derivations of a word can be generated from a base form by application of lexical rules several times. In the example (13a), a simplified lexical rule for passivization of verbs for English is given where f_{PSP} is the function mapping verb base to its past-participle form. Also, it is possible to generate different readings and syntactic behaviours of the same word. Lexical rule in (13b) duplicates lexical entry of verbs for non-referential objects in preverbal position in Turkish.

(13) a.

$$\begin{bmatrix} \textit{word} & \\ \text{PHON} & \boxed{1} \\ \text{SYNSEM} \mid \text{LOCAL} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \textit{verb} & \\ \text{TENSE} & \textit{base} \\ \text{PASSIVE} & - \end{bmatrix} \\ \text{SUBCAT} & \left\langle \ldots\ [\ ]_{\boxed{2}},\ [\ ]_{\boxed{3}}\ \ldots \right\rangle \\ \text{SUBJ} & \boxed{2} \end{bmatrix} \\ \text{CONT} & \boxed{4} \end{bmatrix} \end{bmatrix} \longmapsto$$

$$\begin{bmatrix} \textit{word} & \\ \text{PHON} & f_{PSP}(\boxed{1}) \\ \text{SYNSEM} \mid \text{LOCAL} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \textit{verb} & \\ \text{PASSIVE} & + \end{bmatrix} \\ \text{SUBCAT} & \left\langle \text{PP[BY]}_{\boxed{2}},\ \ldots,\ [\ ]_{\boxed{3}}\ \ldots \right\rangle \\ \text{SUBJ} & \boxed{3} \end{bmatrix} \\ \text{CONT} & \boxed{4} \end{bmatrix} \end{bmatrix}$$

b.

$$\begin{bmatrix} \textit{word} & \\ \text{PHON} & \boxed{1} \\ \text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} & \begin{bmatrix} \text{HEAD} & \textit{verb} \\ \text{SUBCAT} & \left\{ \text{NP}_{acc}[\ ],\ \boxed{2}\ (\ldots.) \right\} \end{bmatrix} \end{bmatrix} \longmapsto$$

$$\begin{bmatrix} \textit{word} & \\ \text{PHON} & \boxed{1} \\ \text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} & \begin{bmatrix} \text{HEAD} & \textit{verb} \\ \text{SUBCAT} & \left\langle \left\{ \boxed{2} \right\},\ \text{NP}_{nom}[\ ] \right\rangle \end{bmatrix} \end{bmatrix}$$

Chapter 3

TURKISH SYNTAX OVERVIEW

This chapter is adopted from [9]. Turkish is an agglutinative language where words are formed by affixation of derivational and inflectional morphemes to root words. So most of the syntactic properties of a word such as case, agreement, relativization of nouns, tense, modality, aspect of verbs, and even passivization, negation, causatives, reflexives and some auxiliaries are marked by suffixes.

(14) a.

ev	-imiz	-de	-ki	-nin
house	-*1PP*	-*Loc*	-*Rlvz*	-*Gen*

‘of the one that is in our house’

b.

bak	-tır	-a	-mı	-yor	-muş	-sun
look	-*Caus*	-*Able*	-*Neg*	-*Prog*	-*Asp*	-*2Sg*

‘you were not able to make look (reported past)’

As a result, Turkish words —especially heads of phrases— have complex and rich syntactic forms and carry much information.

As another distinct property, Turkish is head-final. Specifiers and modifiers always precede the specified or modified. Similarly complements and arguments precede the head in their usual formation. However when head is a verb or predicative noun, complements and objects may follow the head.

(15) a.

Benim	kapıdaki	kırmızı	arabam
ben-*Gen*	door-*Loc-Rlvz*	red	car-*1SP*

‘my red car at the door’

b. Hicabi kitabı çok çabuk okudu.
Hicabi book-*Acc* very quick read-*Past-3Sg*
'Hicabi read the book very quickly.'

c. Kitapları verdim Ahmet'e.
book-*Plu* give-*Pass-1Sg* Ahmet-*Dat*
'I gave Ahmet the books.'

Also in Turkish, constituents have free order. The most usual sentence order is S-O-V. However they can scramble causing different readings and interpretations. Sentence-initial position marks the *topic*, pre-verbal constituent is the *emphasis* and post-verbal position is for the *background* or *afterthought* information [7].

(16) a. Onur kalemi çocuğa verdi.
Onur pencil-*Acc* child-*Dat* give-*Past-3Sg*
'Onur gave the child the book.'

b. Onur çocuğa kalemi verdi.
'Onur gave the pencil to the child.'

c. Çocuğa kalemi Onur verdi.
'It is Onur who gave the child the book.'

d. Kalemi Onur verdi çocuğa.
(c)

e. Onur verdi kalemi çocuğa.
'Onur did give the child the pencil.'

When the object is non-referential (ie. no case marked or specified), it should immediately precede the verb.

(17) a. Adam bahçede şiir yazıyordu
man garden-*Loc* poem write-*Prog-Asp-3Sg*
'The man was writing poem in the garden'

b. (*) Şiir bahçede adam yazıyordu.
'The poem was writing the man in the garden'

c. * Adam şiir bahçede yazıyordu.

Similarly, adverbs and sentential complements may scramble freely (18a–c). Also order variation of constituents is valid for the embedded sentences such as relative clauses, infinitive and gerundive forms, and sentential complements. Relative clauses are strictly head-final; no constituent belonging to relative clause can follow the head verb (18d–f).

(18) a. Gerçekten onun sınavı kazanmasını herkes istemişti.
really he-*Gen* exam-*Acc* pass-*Inf-3Sg-Acc* everyone want-*Past-Asp*
'Everybody realy wanted him to pass the exam.'

b. Onun sınavı kazanmasını herkes gerçekten istemişti.

c. Herkes onun sınavı kazanmasını gerçekten istemişti.

d. Bakkaldan dün aldığım kalem kırıldı
store-*Abl* yesterday buy-*Rel-1Sg* pencil break-*Pass-Past-3Sg*
'The pencil that I bought yesterday from the store was broken'

e. Dün bakkaldan aldığım kalem kırıldı.

f. * Dün aldığım bakkaldan kalem kırıldı.

3.1 Noun Phrase

Phrases with nominal heads are noun phrases. The head noun is the final constituent of the phrase and determines the syntactic role of the whole phrase. Noun phrases may act as a subject, object or complement of a sentence or modifier or specifier of another noun group. A noun —so a noun phrase— can have the cases listed in Table 3.1.[1]

Table 3.1: Cases for Turkish nouns

case	suffix	examples
nominative		adam, kedi
accusative	`-(y/n)H`	adamı, kediyi
dative/allative	`-(y/n)A`	adama, kediye
locative	`-(n)DA`	adamda, kedide
ablative	`-(n)DAn`	adamdan, kediden
genitive	`-(n)Hn`	adamın, kedinin
comitative/instrumental	`-(y)lA`	adamla, kediyle

Also three suffixes `-cA`, `-lH` and `-sHz` (equative, munitive and privative respectively) are considered as cases by Banguoğlu [1].

[1] We use `A` to stand for `a` or `e`, `H` to stand for `ı, i, u or ü`, and `D` to stand for `d` or `t`.

Nominative case is used for marking subjects (19a), indefinite/nonreferential objects (19b). Also noun with the nominative case can be a classifier for another noun (19c).

(19) a. *Köpek* kediyi kovaladı.
dog cat-*Acc* chase-*Past-3Sg*
'The dog chased the cat.'

b. Adam *kuş* avladı.
man bird hunt-*Past-3Sg*
'The man hunted a bird.'

c. Güzel bir *köpek* evi yaptık.
nice a dog house make-*Past-1Pl*
'We made a nice dog house.'

The accusative case is used for marking definite objects. It is obligatory with pronouns and proper nouns in object position.

(20) a. Çocuk kitabı okumamış.
child book-*Acc* read-*Neg-Past-3Sg*
'The child hasn't read the book.'

b. Köpek Ayşe'yi ısırdı.
dog Ayşe-*Acc* bite-*Past-3Sg*
'The dog bit Ayşe.'

c. Herkes onu suçluyor.
everyone he/she-*Acc* blame-*Prog-3Sg*
'Everyone blames him/her.'

Noun phrases with dative/allative case (`-(y/n)A` suffix) have three roles: they behave as prepositional phrases indicating target or aim (21a–b), mark the indirect object (21c), and they are subcategorized as the oblique object in some verbs (21d).

(21) a. Çocukları Ankara'ya gönderdik.
child-*Plu-Acc* Ankara-*Dat* send-*Pass-2Pl*
'(We) sent the children to Ankara.'

b. Çiçekleri sana aldım.
flower-*Plu-Acc* you-*Dat* buy-*Past-1Sg*
'(I) bought the flowers for you.'

c. Mehmet ekmeği adama verdi.
Mehmet bread-*Acc* man-*Dat* give-*Past-3Sg*
'Mehmet gave the man the bread.'

d. Kadın bahçeye baktı.
woman garden-*Dat* look-*Past-3Sg*
'The woman looked at the garden.'

Noun phrases with locative case (`-DA` suffix) is used to express the location of an action or object (22).

(22) Kitabın masada duruyor.
book-*2SP* table-*Loc* stand-*Prog-3Sg*
'Your book lays on the table'

The ablative case (`-DAn` suffix) indicates the source of an action or object as the English preposition "from" (23a–b). Also can be subcategorized as direct object by a group of verbs (23c) .

(23) a. İstanbul'dan yeni gelmiş.
İstanbul-*Abl* just come-*Past-3Sg*
'He has just come from İstanbul'

b. Genelde bu üzümlerden şarap yapılıyor
usually these grape-*Plu-Abl* wine make-*Pass-Prog-3Sg*
'Usually wine has been done from these grapes'

c. Ahmet kedilerden nefret eder.
Ahmet cat-*Plu-Abl* hate--*Pres-3Sg*
'Ahmet hates cats.'

The genitive case is used to mark the possessor in the possesive-possessor relation. Noun with the genitive case behaves as a specifier of possessed noun which is marked with the possessive suffix. Person and number information of the noun should agree with this possesive suffix.

(24) a. Arabanın anahtarını unuttum.
car-*Gen* key-*3SP-Acc* forget-*Past-1Sg*
'I forgot the key of the car.'

b. Senin kalemini kullandım.
you-*Gen* pencil-*2SP*-*Acc* use-*Past*-*1Sg*
'I have used your pencil.'

c. İlker'in arabasının motoru bozuk
İlker-*Gen* car-*3SP*-*Gen* engine-*3SP* broken
'The engine of the İlker's car is broken.'

The `-(y)lA` suffix is the combined form of the postposition **ile** with the noun. It marks the commutative (25a) and instrumental (25b) relationships.

(25) a. Kitabı Ahmet'le gördük.
book-*Acc* Ahmet-*Ins* see-*Past*-*1Pl*
'Ahmet and I saw the book together.'

b. Kuşları dürbünle seyrediyoruz.
bird-*Plu*-*Acc* binocular-*Ins* watch-*Prog*-*1Pl*
'We could see the birds with telescope.'

`-cA` suffix is used for marking subject of a passive sentence. Postposition **tarafından** is more commonly used compared to equative case.

(26) a. Kampanya vatandaşlarca destekleniyor.
campaign citizen-*Plu*-*Equ* support-*Pass*-*Prog*-*3Sg*
'The campaign is supported by citizens.'

`-lH` and `-sHz` (munitative and privative) suffixes have similar meaning with the prepositional phrases formed by 'with' and 'without' in English. Noun phrases with these suffixes behave as adjective. However, `-lH` suffix saves some of the properties of the noun it is attached to. Noun may be still the head of a noun group and can be modified (27).

(27) a. kırmızı kanatlı böcek
red wing-*Mun* insect
'the insect with red wings'

b. üç tekerlekli bisiklet
three wheel-*Mun* bicycle
'The bicycle with three wheels'

Segments	Alternatives	Examples
Specifier	Quantifier	her, bazı, biraz, kimi, herbir, birçok
	Article	bir
	Demonstrative Adjective	bu, şu, o, diğer, ilk, sonuncu
	Genitive noun	bahçenin
	Classifier noun	*mutfak* dolabı
Modifier	Quantitative Adjective	dört, yarım, ikişer, üçlü
	Qualitative Adjective	güzel, zor
	Relativized noun	evdeki, akşamki
	Relative clause	postadan çıkan, yolda gördüğüm
	Unit noun	bardak, salkım, tane
Head	Common noun	ev, kitap
	Proper noun	Deniz, Ankara
	Pronoun	ben, sen, onlar

Table 3.2: Segments of a noun group.

Another inflection that a noun group may have is the *relativizer* (`-ki` suffix). This suffix is attached to some temporal adverbs and nouns with locative case. A relativized noun becomes a specifier for another noun group (28).

(28) bahçedeki çiçekler
garden-*Loc-Rlvz* flower-*Plu*
'The flowers in the garden'

`-ki` suffix following a genitive noun group behaves as a pronoun meaning 'one that belongs to' and different from the relativizer `-ki`.

(29) Ayşe'ninkiler yarın gelecek
Ayşe-*Gen-Pro-Plu* tomorrow come-*Fut-3Sg*
'Ones that Ayşe owns will come tomorrow.'

A noun group consists of an optional group of specifier and modifiers and a head noun. Head noun can be a common noun, a pronoun or a proper noun. Order and grammatical combinations of specifiers and modifiers change according to the type of specifiers and modifiers. Order and valid combinations of specifiers and modifiers are pragmatically controlled. Some specifiers/modifiers put some restrictions on the specifier/modifier types that can further specify/modify the noun.

General structure of a noun group can be viewed as a sequence of segments, head noun being the last one. These segments are listed in Table 3.2.

Specifier and modifier segments are optional:

(30) a. bahçenin kapısı
garden-*Gen* gate-*3SP*
'the gate of the garden'

b. şu kız
that girl
'that girl'

c. Ankara

The order of the specifier and modifier segments may vary.

(31)

a.	Her	kırmızı	çiçek	b.	kırmızı	her	çiçek
	every	red	flower		red	every	flower
	'every red flower'				'every flower that is red'		
c.	güzel	bir	kız	d.	bir	güzel	kız
	beautiful	a	girl		a	beautiful	girl
	' a beautiful girl'				'one beautiful girl'		

Each segment of the noun group are elaborated below.

3.1.1 Specifier segment

Specifiers pick out noun(s) out of a set of possible nouns. In Turkish, specifier segment position is filled by a specifier that can be a *quantifier* (32a), an *article* (32b), a *demonstrative adjective* (32c), a *genitive noun* (32d) or a *a classifier noun* (32e).

(32) a. Yazdıklarımız *bazı* insanları rahatsız edecek.
write-*Part-1PP* some people-*Acc* disturbed make-*Fut*
'What we have written will disturb some people.'

b. Yolda *bir* kalem buldum.
road-*Loc* a pencil find-*Past-1Sg*
'I found a pencil on the street.'

c. *İlk* sınavımı geçtim.
first exam-*1SP* pass-*Past-1Sg*
'I passed my first exam.'

d. *yazarın* her kitabı
author-*Gen* every book-*3SP*
'every book of the author'

e. Onur'un bulduğu iki *caz* plağı
Onur-*Gen* find-*Part*-*3Sg* two jazz record-*3SP*
'two jazz records that Onur found'

The valid combinations and and order of specifiers are pragmatically controlled. A noun group may have only one quantifier (33a–b). Also quantifiers cannot be used with demonstrative adjectives and article (33c,d).

(33) a. *her çoğu kitap
every most book

b. *birçok kimi öğrenciler
many some student-*Plu*

c. *<u>her</u> <u>bu</u> <u>kitap</u>
Quant. Dem. Adj. -
every this book

d. * <u>kimi</u> <u>bir</u> <u>insan</u>
Quant. Art. -
some a person

The use of the article with demonstrative adjectives and quantifiers depends on some selectional restrictions.

(34) a. *ilk bir kitap
first a book

b. bir ilk kitap
a first book

c. bir üçüncü kitap
a third book
'yet a third book'

d. *sonuncu bir kitap
last a book

e. bir şu kitap
a that book
'only that book'

f. şu bir kitap
that one book
'that single book'

g. diğer bir kitap
other a book
'another book'

There are some points to be underlined here, about the different meanings of "bir" and about some exceptions:
The sequence depicted in (34b) has a limited usage referring to the first book of an author. In (34c), "bir" is an adverb meaning "yet" or "another". In (34e), "bir" is an adverb with a meaning "only". In (34f), "bir" is not an article but a cardinal number (a modifier). In (34g), "diğer" acts as an adverb.

Concerning the demonstrative adjectives, there are two subgroups:

i) bu, şu, o

ii) ilk, sonuncu, ordinal numbers, diğer

Only one element from each group can be used within a noun group. The elements of the first group can sometimes be used in front of the elements of the second group for emphasizing the demonstration.

(35) a. Bu ikinci kitabı pek beğenmedim
this second book-*Acc* much like-*Neg-Past-1Sg*
'I didn't like the second book much'

b. Şu diğer valiz benimki
that other suitcase mine
'The other suitcase is mine'

c. *diğer sonuncu kız
other last girl

Nouns or noun groups with genitive marking also function as specifiers within a noun group. Genitive nouns can be used in combination with other specifiers (36a–b).

The main restriction is that all specifiers and modifiers modifying the possessive marked noun should follow the genitive noun. Otherwise they specify/modify the genitive noun (36c):

(36) a. yazarın bir kitabı
author-*Gen* a book-*3SP*
'a book of the author'

b. kitabın bu sayfası
book-*Gen* this page-*3SP*
'this page of the book'

c. ilk seminerin konuşmacısı
first seminar-*Gen* speaker-*3SP*
'the speaker of the first seminar'

Genitive nouns can rather be interpreted as complements of possessive marked nouns since possessive nouns require a genitive noun which is subject of the owner relation in the possessive group.

Classifier nouns resemble genitive nouns in that they require a possessive-marked noun group modified by the classifier noun. However, classifier nouns take no genitive suffix.[2] The difference between a genitive noun and a classifier noun is that the former provides a definite reading where the latter provides an indefinite or nonreferential one.

(37) a. duvar boyası
wall paint-*3SP*
'wall paint'

b. duvarın boyası
wall-*Gen* paint-*3SP*
'the paint of the wall'

Classifier noun groups can act as specifiers of other classifier nouns:

(38) kredi kartı faiz yüzdesi
credit card-*3SP* interest percentage-*3SP*
'credit card interest rate'

[2]These noun groups are called **izafet** by Lewis [16]

A classifier noun is the immediate predecessor of the head noun. Hence, other specifiers and modifiers precede it.

(39) a. her çocuk arabası
every child car-*3SP*
'every stroller'

b. o dere yatağı
that stream bed-*3SP*
'that river bed'

c. *çocuk her arabası

d. *dere o yatağı

e. *ev bir kapısı

f. *duvar evin boyası

g. evin duvar boyası
home wall paint-*3SP*
'wall paint of the house'

3.1.2 Modifier segment

Modifiers provide information about the properties of the entity or its relations with other entities. A modifier is either an adjective group, or a noun group. More than one modifier may exist within a noun group.

(40) güzel mavi eteğin
beautiful blue skirt-*2SP*
'your beautiful blue skirt'

As a general rule, "whatever precedes modifies" in Turkish. Hence, if a modifier itself is a noun group or a clause containing a noun, any preceding modifier modifies the first of the succeeding nouns.
For example, in the phrase below, the modifier "yaşlı" modifies "adam" rather than the head noun "kadın".

(41) yaşlı adamın konuştuğu kadın
old man-*Gen* talk-*Part*-*3SP* woman
'the woman to which the old man talked/talks'

Certain restrictions apply to the combinations of modifiers. When a noun is modified by both a qualitative and a quantitative adjective, order of the adjectives may vary but the quantitative adjective usually precedes the qualitative one.

(42) a. üç kırmızı kalem
three red pencil
'three red pencils'

b. hassas ikili ilişkiler
sensitive dual relationship-*Plu*
'sensitive dual relationships'

c. ikişer kalın battaniye
by-two thick blanket
'two thick blankets for each'

d. rahat üçlü kanepe
comfortable triple sofa
'a comfortable triple sofa'

e. yarım çürük elma
half rotten apple
'a half rotten apple'

f. çürük yarım elma
rotten half apple
'a half rotten apple'

When used as modifiers, unit nouns are preceded by a cardinal number (43a), a fractional number (43b), or a distributive adjective (43c):

(43) a. iki bardak süt
two glass milk
'two glasses of milk'

b. yarım somun ekmek
half loaf bread
'half loaf of bread'

c. birer dilim pasta
by-one slice cake
'a slice of cake (for each)'

When the unit noun denotes a container, the word *dolusu* ("full"-*3SP*) may optionally be inserted between the unit noun and the head.

(44) üç kaşık dolusu şeker
three spoon full-*3SP* sugar
'three spoonful of sugar'

The other group of modifier is the relativized nouns which are inflected by relativizer suffix `-ki`. If the head noun is modified by a relativized noun, all other modifiers and specifiers of the head come after the relativized noun (45a). Otherwise, any modifier preceding a relativized noun modifies the relativized noun rather than the head (45b).

(45) a. çantamdaki üç küçük anahtar
handbag-*1SP*-*Loc*-*Rlvz* three small key
'three small keys in my handbag'

b. küçük çantamdaki üç anahtar
small handbag-*1SP*-*Loc*-*Rlvz* three key
'three keys in my small handbag'

Noun groups can also be modified by relative clauses. In Turkish, the noun on which the relativization is performed is placed at the final position of the relative clause.

(46) a. Ağabeyim Ankara'da çalışıyor.
brother-*1SP* Ankara-*Loc* work-*Prog-3Sg*
'My elder-brother works in Ankara.'

b. Ankara'da çalışan ağabeyim
Ankara-*Loc* work-*Rel* brother-*1SP*
'My elder-brother who works in Ankara'

As seen above, the main verb of the relative clause is used in participle form. The example depicts the suffix `-en` (phonological realization of `-(y)An` after morphophonemic processes) which is used in producing subject participle (in present). Other subject suffixes are given below:

Suffix	Tense
`-mHş` (olan)	past
`-(y)AcAk` (olan)	future
`-Hyor` (olan)	progressive

The word *olan* ("being") can optionally be used with past, future or progressive participles, but not with present participle.

(47) a. Ankara'da çalışmış olan ağabeyim
Ankara-*Loc* work-*Part* be-*Rel* elder-brother-*1SP*
'my elder brother who have worked in Ankara'

b. Ankara'da çalışacak olan ağabeyim
Ankara-*Loc* work-*Part* be-*Rel* elder-brother-*1SP*
'my elder brother who will work in Ankara'

c. * Ankara'da çalışan olan ağabeyim

Olan can also be used in forming participle form of the copula.

(48) a. Arkadaşımın annesi hasta.
friend-*1SP*-*Gen* mother-*3SP* ill
'My friend's mother is ill.'

annesi hasta olan arkadaşım
mother-*3SP* ill be-*Rel* friend-*1SP*
'my friend whose mother is ill'

b. Evin pencereleri geniş.
house-*Gen* window-*Plu*-*Acc* wide
'Windows of the house are wide.'

pencereleri geniş olan bir ev
window-*Plu*-*3SP* large be-*Rel* a house
'a house which has large windows'

Apart from the subject participle form, the verb of a relative clause may take *complement participle* form, which is obtained by attachment of either `-DHk` or `-yAcAk` suffixes. `-DHk` suffix, as itself, produces adjectives from verbs, although it is not productive:

(49)

bilmek → bildik
'to know' 'known'

umulmamak → umulmadık
'to be not expected' 'unexpected'

When used in complement participles, `-DHk` is always followed by a possessive (marks the agreement in this case) and participle suffix group becomes `DHğ`-*Agr*. The tense of this participle can be past or present, as examples (50) and (51) depicts, respectively. Actual tense is usually determiner from the discourse.

(50) Kitabı kıza geri verdim.
book-*Acc* girl-*Dat* back give-*Past-1Sg*
'I gave back the book to the girl.'

a. kıza geri verdiğim kitap
girl-*Dat* back give-*Rel-1Sg* book-*Acc*
'The book that I gave back to the girl.'

b. kitabı geri verdiğim kız
book-*Acc* back give-*Rel-1Sg* girl
'The girl to whom I gave back the book.'

If there is an overt subject noun in the clause, it is marked with the genitive suffix:

(51) Öğrenci sınıfta şarkı söylüyor.
student class-*Loc* song sing-*Prog-3Sg*
'The student is singing a song in the classroom.'

a. öğrencinin sınıfta söylediği şarkı
student-*Gen* class-*Loc* sing-*Rel-3Sg* song
'the song that the student is singing in the classroom'

b. öğrencinin şarkıyı söylediği sınıf
student-*Gen* song-*Acc* sing-*Rel-3Sg* class
'the classroom in which the student is singing the song'

Complement participles in future tense are formed by attaching `-(y)AcAk` suffix to verb stem. Just like `-DHk` suffix, `-(y)AcAk` combines with a possesive suffix to produce

`-(y)AcAğ`-*Agr* as the future complement participle.

(52) a. öğrencinin söyleyeceği şarkı
student-*Gen* sing-*Rel*-*3Sg* song
'the song that the student will sing'

b. kitabı geri vereceğim kız
book-*Acc* back give-*Rel-1Sg* girl
'the girl to whom I will give back the book'

Relative clauses can be embedded as adnominals:

(53) köyde yaşayan kızın yetiştirdiği ineğin öldüğü yer
village-*Loc* live-*Rel* girl-*Gen* breed-*Rel-3Sg* cow-*Gen* die-*Rel-3Sg* place
'the place at which the cow that was breeded by the girl who lives in the village died '

3.1.3 The head

The last segment of the noun group is the head, and this position is filled either by a common noun (54a), a proper noun (54b) or a pronoun (54c).

(54) a. küçük bir elma
small a apple
'a small apple'

b. güzel Ayşe
beautiful Ayşe
'beautiful Ayşe'

c. unutkanlığıyla bilinen sen
forgetful-*3Sg-Ins* know-*Pass-Rel* you
'you who are known as forgetful'

Pronouns

When the head is a pronoun, no determiner or modifier segments are allowed:

(55) *bazı sen
some you

*sarışın ben
blond I

Proper Nouns

When it is used as the head, a proper noun imposes certain restrictions on the selection of the preceding segments. For example, particular determiners can be used in front of a proper noun, while others are not applicable.

(56) a. *Bu İstanbul* nasıl düzelir?
this İstanbul how get-better-*Pres-3Sg*
'How could this İstanbul get better?'

b. Nerede kaldı *şu Hasan?*
where left that Hasan
'Where on the earth is Hasan?'

c. Trakya'da *birkaç Yeşilköy'e* rastladım
Thrace'-*Loc* several Yeşilköy'-*Dat* come-across-*Past-1Sg*
'I came across more than one Yeşilköy in Thrace'

d. Ailemizdeki *diğer/ikinci Mehmet* dedemdir. .
family-*1PP-Loc-Rlvz* other/second Mehmet grandfather-*1SP-Cop*
'The other/second Mehmet in our family is my grandfather'

3.2 Postposition group

By postposition group, we mean a group of elements whose head is a proposition. Postposition group consists of a head and an optional complement noun group. The former always occupies the final position.

3.2.1 Postpositions

Postpositions form a closed class of words. They can be viewed in subgroups, with respect to the case of the complement they subcategorize for. ([16], pp. 85-89) Various types of postpositions exist which subcategorize for: infinitives or nouns with nominative case (57a,b), nouns with accusative case (57c), dative case (57d) and ablative case (57e).

(57) a. gelmek üzere
come-*Inf* for
'for the purpose of coming'

b. sokak boyunca
street along
'along the street'

c. Sınavı müteakiben
exam-*Acc* following
'after the exam'

d. şimdiye dek
now-*Dat* until
'until now'

e. Dünden beri
yesterday-*Abl* since
'since yesterday'

3.2.2 Postposition Attachment

Attachment of a sequence of postpositions is determined without ambiguity by morphosyntactic cues (e.g., relative suffixes and case marks) and positional cues (head-final structure). However, if a sentence involves relative clauses and postpositions, ambiguities may arise (58a).[3] In "I read the newspaper on the couch", if *on the couch* were an adnominal, it would be relativized in Turkish (cf.,58b-c). Chained postposition groups are not ambiguous because the predecessor modifies the successor.

(58) a. Bu bilgilere göre yazdığımız rapor değişmeyecek.
this data-*Plu-Dat* according write-*Rel-1Pl* report change-*Neg-Fut-3Sg*
'The report that we wrote according to these data will not change.'
'According to these data, the report that we wrote will not change'

[3]In writing, it this may be disambiguated by seperating the sentential complement with a coma (before **yazdığımız** in 58a).

b. Kanepedeki gazeteyi okudum.
couch-*Loc-Rlvz* newspaper-*Acc* read-*Past-1Sg*
'I read [the newspaper on the couch].'

c. Kanepede gazeteyi okudum.
couch-*Loc* newspaper-*Acc* read-*Past-1Sg*
'I read the newspaper [on the couch].'

3.3 Adjective group

Adjective group is a sequence of words last of which is an adjective. Adjective groups are typically formed by comparative and superlative adjectives.

3.3.1 Comparative adjectives

The head of a comparative adjective group is a qualitative adjective. Three comparatives can precede the head: "daha", "az" and "çok" meaning "more", "less" and "very", respectively.

(59) a. Elvan daha büyük bir eve taşındı.
Elvan more big a house-*Dat* move-in-*Past-3Sg*
'Elvan moved in to a bigger house.'

b. Az şekerli kahve içerdi.
less sweet coffee drink-*Pres-Asp*
'(S)he used to drink coffee with a little sugar.'

c. Çok hızlı arabalardan hoşlanmıyorum.
very fast car-*Plu-Abl* like-*Neg-Prog-1Sg*
'I don't like very fast cars.'

d. Annem benden çok daha iyi yemek yapar.
mother-*1SP* I-*Abl* very more good dish make-*Pres-3Sg*
'My mother cooks much better than I do.'

3.3.2 Superlative adjectives

The head adjective is qualitative for this group, too. Superlative form is obtained by preceding the head with “en” (“most”).

(60) Sınıfın en çalışkan öğrencisi Ali’ydi.
class-*Gen* most hardworking student-*3SP* Ali-*Cop*
‘Ali was the most hardworking student of the class.’

3.4 Adverb group

An adverb group is a segment which has an adverb as its head. Modifiers of an adverbial head may be adverb or adjective groups, including the comparative *daha* and the superlative *en*. Adverbial heads may be classified as manner (*alelacele*), temporal (*sonra, önce*), position (*aşağı, beri, ileri*), repetition (*gene, yeniden, tekrar*), sentential (*besbelli, asla, kuşkusuz*), frequency (*seyrek, sık*), possibility (*herhalde, belki*), definiteness (*katiyen, muhakkak*), and question (*nasıl, hani*) adverbs. Basic types of adverb groups are described below.

3.4.1 Reduplications

Nouns and adjectives can be reduplicated to form an adverb group.

(61) a. Yemeğimizi çabuk çabuk yedik.
meal-*1PP-Acc* quick quick eat-*Past-1Pl*
‘We ate our meal quickly.’

b. Akşam akşam canımızı sıktı.
evening evening soul-*1PP-Acc* bother-*Past-3Sg*
‘It bothered us at this time of the evening.’

c. Geçen yaz bu sahilleri koy koy dolaştık.
last summer this shore-*Plu-Acc* cove cove go-around-*Past-1Pl*
‘We visited each and every cove of this shore last summer.’

Some of the reduplicated adverbs are onomatopoeic words:

(62) Şırıl şırıl akan derenin sesini dinledim.
'splashing' flow-*Part* stream-*Gen* sound-*3SP-Acc*
'I listened to the sound of the stream that flows gently.'

Distributive adjectives, when used as adverbs, are reduplicated:

(63) Merdivenleri üçer üçer çıktık.
stairs-*Plu-Acc* three-*Dist* three-*Dist* go-up-*Past-1Pl*
'We went upstairs three steps by three steps.'

Adverbs or adjectives can be intensified by phonological reduplication to produce adverbs as well:

çabuk	quick	**çarçabuk**	very quickly
hızlı	fast	**hıphızlı**	very fast

3.4.2 Case-marked place adverbs

Adverbs of place act as the head of an adverb group either by themselves or by taking a case suffix.

içeri	inside	**dışarı**	outside
yukarı	up	**aşağı**	down
ileri	forward	**geri**	backward
öte	yonder	**beri**	hitter
ön	front	**arka**	behind
karşı	opposite		

(64) a. Evden dışarı çıkmadım.
house-*Abl* outside go-out-*Neg-Past-1Sg*
'I didn't go out from the house.'

b. Yolun ilerisi görülmüyor.
road-*Gen* forward-POSS see-*Pass-Neg*
'The forward part of the road is not visible.'

c. Nehirden öteye nasıl geçilir?
river-*Abl* yonder-*Dat* how pass-*Pass-Pres-3Sg*
'How can one go beyond the river?'

3.4.3 Temporal adverb groups

"sonra" ("after") and "önce" ("before") succeed noun groups denoting a time period or a point in time, and form adverb groups.

(65) a. Dört gün sonra yola çıkacağız.
four day after road go-out-*Fut-1Pl*
'We'll set out on a journey in four days.'

b. Umarım Perşembeden önce burada olmaz.
hope-*Pres-1Sg* Thursday-*Abl* before here be-*Neg*
'I hope he/she won't be here before Thursday.'

Another type of adverb group denoting time is the one that uses special temporal nouns in head position. These temporal nouns are some time units (**gün**:"day", **hafta**:"week", **ay**:"month", **mevsim**:"season", **yıl**:"year", **yüzyıl**:"century", **dönem**: "semester, age", **çağ**:"era, epoch"), days of week, months and year. In such adverb goups, however, the set of words that may modify the head is rather limited: **önceki** ("previous", "before"), **ertesi** ("following", "after"), **geçen** ("last"), **gelecek** ("next"), **bu** ("this"), **o** ("that").

(66) a. Ertesi gün eski bir arkadaşıma rastladım.
following day old a friend-*1SP-Dat* come-across-*Past-1Sg*
'The following day, I came across with an old friend of mine.'

b. Gelecek yaz Paris'e gideceğim.
next summer Paris-*Dat* go-*Fut-1Sg*
'I will go to Paris next summer.'

3.4.4 Verb groups with adverbial use

Verb stems may function as adverbs with the addition of certain suffixes. These suffixes are discussed below.

`-(y)A` suffix denotes a repeated action that takes place at the same time with the main verb. Verb groups in this gerundive form consist of two gerunds (either of the same verb or different verbs).

(67) a. Ağacı budа*ya* buda*ya* biçimlendirdi.
tree-*Acc* prune prune shape-*Past-3Sg*
'He/she shaped the tree pruning.'

b. Çocuk düş*e* kalk*a* büyür.
child fall rise grow-*Pres*
'A child grows falling and rising.'

`-(y)ArAk` suffix denotes a continuous action or a point action which takes place either at the same time with the main verb or just before it.

(68) a. Öpüş*erek* ayrıldılar.
kiss-*Recp* leave-*Past-3Pl*
'They kissed each other as they said goodbye.'

b. Koş*arak* karşıya geçtik.
run opposite-*Dat* pass-*Past-1Pl*
'We crossed the street running.'

`-(y)Hp` suffix is attached to the first of consecutive verb stem pairs and provides a connection (e.g., temporal sequence) between these stems.

(69) a. Şemsiyemi işyerinde unut*up* gelmişim.
umbrella-*1SP-Acc* office-*Loc* forget come-*Past-1Sg*
'I came, having forgotten my umbrella at the office'

b. Otur*up* konuşalım
sit-down talk--*Wish-1Pl*
'Let's sit down and talk.'

`-(y)HncA` suffix marks its stem as the temporal predecessor of the main verb.

(70) a. Eve var*ınca* seni ararım.
house-*Dat* arrive you-*Acc* call-*Pres-1Sg*
'I'll call you when I arrive home.'

b. Haberleri dinle*yince* yolculuğumu erteledim.
news-*Acc* listen travel-*1SP-Acc* postpone-*Past-1Sg*
'I postponed my travel when I listened to the news.'

-DHkçA suffix is a composite one which combines participle suffix -DHk with çA. This composite suffix has the meaning “so long as” or “the more”.

(71) a. Çalışma*dıkça* başarılı olamazsın.
study-*Neg* successful be-*Neg-Pres-2Sg*
‘So long as you don’t study, you cannot be successful.’

b. Ankara’ya gel*dikçe* bize uğrar.
Ankara-*Dat* come we-*Dat* visit-*Pres-3Sg*
‘Every time he/she comes to Ankara, he/she visits us.’

c. İp atla*dıkça* susuyorum.
rope skip be-thirsty-*Prog-1Sg*
‘The more I skip, the more I get thirsty.’

The suffix sequence -(H)r· · ·mAz attach to the same verb stem to produce a verb group that can be used like an adverb. This construction has a meaning similar to “as soon as”.

(72) a. İbibikler öt*er* öt*mez* oradayım.
hoopoe-*Plu* sing sing-*Neg* there-*Loc-Cop(1Sg)*
‘I will be there as soon as the hoopoes sing.’

b. Otobüsten in*er* in*mez* onu gördüm.
bus-*Abl* get-off get-off-*Neg* he/she/it see-*Past-1Sg*
‘I saw her/him/it as soon as I got off the bus.’

-(y)ken suffix is the last one that is to be discussed in this section. It can be translated to English as “as”. This suffix differs from the previous ones as it attaches not to a verb stem, but usually to third person singular inflection of the verb in aorist. It may also attach to narrative past, present and future tense forms for third person singular. The suffix does not harmonize with the vowels of the verb stem.

(73) a. Çayımı içer*ken* gazete başlıklarına göz atarım
tea-*1SP-Acc* drink newspaper headline-*3PP-Dat* eye throw-*Pres-1Sg*
‘I glance through newspaper headlines as I drink my tea’

b. Düşümde dövüşmektey*ken* yanımda yatanı tekmelemişim.
dream-*1Sg-Loc* fight side-*1Sg-Loc* lie-*Rel-Acc* kick-*Past-1Sg*
‘I had kicked the one lying next to me as I was fighting in my dream.’

c. Buraya kadar gelmiş*ken* geri dönmek olmaz
here-*Dat* upto come back turn-*Inf* be-*Neg-Pres*
'It's impossible to go back now that we came up to here.'

3.5 Verb group

3.5.1 Predicate types

Predicates in Turkish can be verbal (74a), nominal with an attached auxiliary suffix (74b), nominal with a copula (74c–d), or existential(74e–f).

(74) a. Adam topa sert vurdu.
man ball-*Dat* hard hit-*Past-3Sg*
'The man hit the ball hard.'

b. Kitabın arabadaydı.
book-*2SP* car-*Loc-Aux*
'(Your) book was in the car.'

c. Benimki en hızlı arabadır.
I-*Gen-Pro* most fast car-*Cop*
'Mine(my car) is the fastest car.'

d. Gökyüzü hep mavidir.
sky always blue-*Cop*
'The sky is always blue.'

e. Ayşe'nin iki çocuğu var.
Ayşe-*Gen* two child-*Acc* exist
'Ayşe has two children (there exist two children of Ayşe).'

f. Sokakta kimse yok.
street-*Loc* nobody not-exist
'There is (there exists) no one on the street.'

3.5.2 Subcategorization

Every verb—except the intransitives—subcategorize for a noun group or a set of noun groups. These noun groups may be in accusative (75a), dative (75b), locative (75c), ablative (75d) or instrumental/commitative case (75e).

(75) a. Raporu henüz bitirmedik.
report-*Acc* yet finish-*Neg-Past-1Pl*
'We haven't finished the report yet.'

b. Yarın sinemaya gidelim.
tomorrow cinema-*Dat* go-*Wish-1Pl*
'Let's go to the cinema tomorrow.'

c. Buzdolabında hiçbir şey kalmamıştı.
refrigerator-*Loc* no-at-all thing remain-*Neg-Past-Asp-3Sg*
'Nothing was left at all in the refrigerator.'

d. Atakule'den dönerken Evrim'i gördüm.
Atakule-*Abl* return-*Adv* Evrim-*Acc* see-*Past-1Sg*
'I saw Evrim as I was coming back from Atakule.'

e. Çocuklar oyuncaklarıyla oynuyorlar.
child-*Plu* toy-*3PP-Ins* play-*Prog-3Pl*
'The children are playing with their toys.'
Ufuk bir arkadaşıyla çalışacak.
Ufuk a friend-*3SP-Ins* work-*Fut-3Sg*
'Ufuk will work with a friend of his.'

The number of required noun groups depend on the valency of the verb.

Transitive verb: Kitap okuyordu.
book read-*Prog-Asp-3Sg*
'He/she was reading a book.'

Ditransitive verb: Mehmet'e gitarımı verdim.
Mehmet-*Dat* guitar-*1SP-Acc* give-*Past-1Sg*
'I gave my guitar to Mehmet.'

More noun groups may be provided to increase the amount of information provided; they act as complements.

(76) Sandıkları İzmir'den Samsun'a gemiyle yolladık
chest-*Plu-Acc* İzmir-*Abl* Samsun-*Dat* ship-*Ins* send-*Past-2Pl*
'We sent the chests from İzmir to Samsun by ship.'

Some verbs subcategorize for a complement clause:
söylemek: "to say", **söz vermek**: "to promise", **iddia etmek**: "to claim", **inanmak**: "to believe", **zannetmek**: "to assume", **tahmin etmek**: "to guess", **sanmak**: "to suppose", **ispat etmek**: "to prove", **inkar etmek**: "to deny", **yemin etmek**: "to swear", **düşünmek**:

"to think", **emin olmak**: "to be sure", **kuşkulanmak**: "to suspect" etc.

(77) a. Dosyayı bulacağına söz vermiştin.
file-*Acc* find-*Part-2Sg-Dat* promise-*Past-Asp-2Sg*
'You had promised that you would find the file.'

b. Randevumuzu unuttuğumu iddia ediyor.
Appointment-*1PP-Acc* forget-*Part-1Sg-Acc* claim-*Prog-3Sg*
'She/he claims that I forgot our appointment.'

Some verbs (**yeğlemek**: "to prefer", **kabul etmek**: "to accept", **çalışmak**: "to try", **çabalamak**: "to struggle", **alışmak**: "to get accustomed to", **özenmek**: "to desire", **karar vermek**: "to decide", **niyetlenmek**: "to intend", **bahsetmek**: "to mention", **vazgeçmek**: "to give up", **anlamak**: "to understand" etc.) subcategorize for an infinitive form of the verb.

(78) a. Bu işi bitirmeye söz verdik.
this job-*Acc* finish-*Inf-Dat* promise-*Past-1Pl*
'We promised to finish this job.'

b. Bugün alışveriş yapmaktan vazgeçtik.
today shopping make-*Inf-Abl* give-up-*Past-1Pl*
'We gave up (the idea of) shopping today.'

3.5.3 Auxiliary verbs

In Turkish, some verbs are composed of a noun and an auxiliary verb. The auxiliary verbs used in such constructions are **etmek**: "to do" and **yapmak**: "to make", the former being more frequent.

(79) a. alay → alay etmek
'mockery' 'to mock'

b. kabul → kabul etmek
'acceptance' 'to accept'

c. alışveriş → alışveriş yapmak
'shopping' 'to shop/to do shopping'

There is another auxiliary which attaches to nouns to form nominal predicates: **olmak** ("to be"). This auxiliary differs from **etmek** and **yapmak** in two respects. First, it does not appear as a separate word, but rather a morpheme when the sentence is in past or present tense. Second, its inflection does not resemble to that of a verb but the copula.

(80) a. Babam geçen ay yurtdışındaydı.
father-*1SP* last month abroad-*Loc-Aux*
'My father was abroad last month.'

b. Üç gündür çok uykusuzdum.
three day very sleepless-*Aux*
'I have been very sleepless for three days.'

c. Kitaplar masanın üstündeymiş.
book-*Plu* table-*Gen* top-*1SP-Loc-Aux*
'The books were on the table.'

This auxiliary is not present for third person form if the sentence is in the present tense.

(81) Bardaklar rafta.
glass-*Plu* shelf-*Loc-(Cop)*
'The glasses are on the shelf.'

For future tense as well as conditional and necessitative forms, **olmak** succeeds the noun as a separate word.

(82) a. Babam geçen ay yurtdışında olmasaydı...
father-*1SP* last month abroad-*Loc* be-*Neg-Cond-Asp*
'If my father weren't abroad last month...'

b. Kitaplar masanın üzerinde olacak.
book-*Plu* table-*Gen* top-*1SP-Loc* be-*Fut-3Sg*
'The books will be on the table.'

Another point to be emphasized about this auxiliary is that it has a different negative form than the other verbs when the sentence is in past or present tense. Negativization is performed by introducing the word **değil** ("not") just after the nominal. The tense marker, if exists, attaches to **değil**.

(83) Cem evde değildi.
Cem house-*Loc* not-*Aux-3Sg*
'Cem wasn't at home.'

An ambiguity may arise with negative questions of predicates. This ambiguity is resolved by stress in speech and by a comma preceding **değil** in writing.

(84) Kedi bahçede değil mi?
cat garden-*Loc* not *Ques*
'The cat is in the garden, isn't it?'
'Isn't the cat in the garden?'

3.5.4 Existential predicates

Existential predicates are formed using **var** ("existent") and **yok** ("non-existent").

(85) a. Odada dört koltuk vardı.
room-*Loc* four armchair exist-*Aux*
'There were four armchairs in the room.'

b. Burada kimse yok.
here anybody non-existent
'There isn't anybody here.'

c. Arabası yokmuş.
car-POSS non-existent
'She/he doesn't have a car.'

var and **yok** cannot be used in future, conditional or necessitative forms. For these cases, **olmak** replaces **var** and **yok**.

(86) a. Odada dört koltuk olmalı
room-*Loc* four armchair be-*Nec*
'There should be four armchairs in the room.'

b. Burada kimse olmayacak.
here nobody be-*Fut-3Sg*
'There won't be anybody here.'

3.5.5 Infinitive form of the verbs

Infinitive form of a verb is formed with suffix `-mAk` attached to the stem.

(87) Fransa'ya gitmek çok para ister.
France-*Dat* go-*Inf* much money require-*Pres*
'Going to France costs much'

Infinitive suffix cannot be followed by genitive or possesive suffixes. However case suffixes are allowed.

(88) a. Koşmaktan yoruldum.
run-*Inf-Abl* tire-*Pass-Past-1Sg*
'I got tired of running.'

b. Kızmakta haklısın.
get-angry-*Inf-Loc* right-*Cop(2Sg)*
'You are right to be angry.'

3.5.6 Gerundive forms of the verbs

There are a couple of suffixes for producing gerundive forms of a verb: `-mA` and `-(y)Hş`. `-mA` is used for referring to the action or its result. Genitive and possesive suffixes can be attached to `-mA`.

(89) a. Onunla görüşmenin bana faydası olmaz.
he/she-*Gen-Ins* meet-*Ger-Gen* me use be-*Neg-Pres-3Sg*
'Meeting with him/her is of no use to me.'

b. Okuması düzeliyor.
read-*Ger-3SP* improve-*Prog-3Sg*
'His/her reading is improving.'

`-(y)Hş` produces a gerundive which emphasizes the manner the action is performed. This suffix can also be succeeded by genitive and possesive suffixes.

(90) Gülüşünü hatırlıyorum.
smile-*Inf-3SP-Acc* remember-*Prog-1Sg*
'I remember the way you/(s)he smile/s.'

3.5.7 Syntax of causative verbs

In Turkish, verbs are causativized by attaching the causative suffixes `-DHr, -Hr, -t, -Ht` and `-Ar` to the stem. Using these suffixes, one can obtain a causative verb almost from any verb, including the causatives themselves.

(91) a. inanmak → inandırmak
'to believe' 'to persuade'

b. doğmak → doğurmak
'to be born' 'to give birth to'

c. oturmak → oturtmak
'to sit' 'to seat'

d. korkmak → korkutmak
'to fear' 'to frighten'

e. çıkmak → çıkartmak
'to go out/to go up' 'to remove/to raise'

Appropriate combinations of causative suffixes allow production of multiple causatives:

(92) ölmek → öldürmek → öldürtmek
'to die' 'to kill' 'to have someone killed'

From syntactical point of view, causativization process has two important results: increase in the valency of the verb, and the changes in the grammatical functions of the noun groups.

Any verb form of valency n will require $n+1$ noun groups after causativization:

Intransitive verb: uyumak → uyutmak
'to sleep' ' to send to sleep'

Transitive verb: bir şeyi okumak → birine birşeyi okutmak
'to read something' ' to make someone read something'

Ditransitive verb:
bir şeyi bir yere koymak → birine birşeyi bir yere koydurtmak
'to put something to somewhere' ' to make someone put something to somewhere'

Another effect of causativization is that the noun groups of the original clause change their grammatical functions:

Causativization of an Intransitive Verb: The subject of the intransitive verb becomes the direct object (accusative-marked noun group) in the causative clause. A new noun group is introduced for subject position of the causative clause.

(93) Yiğitcan güldü → Işık Yiğitcan'ı güldürdü
NOM NOM ACC
'Yiğitcan laughed' 'Işık caused Yiğitcan to laugh'

Causativization of a Transitive Verb: The subject of the transitive verb becomes the dative-marked indirect object in the causative clause, whereas the direct object (e.g., *şiir* in 94)) preserves its grammatical function in the causative clause. Also, a new noun group is introduced for subject position.

(94) Arzu şiir okudu → Öğretmen Arzu'ya şiir okuttu
NOM NOM NOM DAT NOM
'Arzu read a poem' 'The teacher made Arzu read a poem'

The indirect object of the causative clause may sometimes be omitted:

(95) Öğretmen şiir okuttu.
teacher poem read-*Caus-Past-3Sg*
'The teacher caused a poem to be read.'

If the main verb subcategorizes for a dative noun group, this noun group remains unaltered in the causative clause. In such a case, the subject of the main verb is marked as accusative and becomes the direct object of the causative clause.

(96) Çocuk okula başladı → Çocuğu okula başlattık
NOM DAT ACC NOM
'The child started school' 'We made the child start school'

Causativization of a Ditransitive Verb: The subject of the ditransitive verb becomes the dative-marked indirect object in the causative clause, whereas the accusative-marked direct object and the dative-marked object of the main verb preserve their grammatical functions in the causative clause. Subject position is again filled by a new noun group.

(97)	Hakan kitabı masaya koydu	→	Ali Hakan'a kitabı masaya koydurdu
	NOM ACC DAT		NOM DAT ACC DAT
	'Hakan put the book on the table'		'Ali made Hakan put the book on the table'

Just as for the tansitive verb, the subject of the main verb may be omitted:

(98) Ali kitabı masaya koydurdu.
Ali book-*Acc* table-*Dat* put-*Caus-Past-3Sg*

Chapter 4

DESIGN

Every language is expected to have a different realization of the language-independent principles. Some solutions proposed by Pollard and Sag [21] for English have to be modified to model Turkish. The basic features that distinguish Turkish from English are: importance of morphology in the specification of grammatical functions, overt case and agreement marking, final head position, free constituent order, pronoun drop, complement drops, and the nature of unbounded dependency constraints. Some of these items are resolved by extending the sign structure and introducing Turkish-specific versions of some principles.

4.1 Sign Structure

The template for phrases is given in Figure 4.1. A structure similar to that of English is used for signs in Turkish, with minor changes. In category (*cat*) type, an additional feature SUBJ is added, which is coindexed with the subject complement if it exists in the subcat list. It is used to distinguish subject complement from other complements which may have the same case and agreement. It refers to the subject complement directly. This is necessary for implementing some syntactic phenomena like subject raising relative clauses and other bindings to subject complement. Example (99) shows a simplified version of the CAT feature of a lexical entry for the verb "seviyor" (love-*Prog-3Sg*).

$$
\begin{bmatrix}
\textit{phrase} & \\
\text{PHON} & \left\langle \textit{string}, \ \ldots \right\rangle \\
\text{SYNSEM} & \begin{bmatrix}
\textit{synsem} & \\
\text{LOCAL} & \begin{bmatrix}
\textit{cat} & \\
\text{CAT} & \begin{bmatrix}
\textit{cat} & \\
\text{HEAD} & \textit{head} \\
\text{SUBJ} & \textit{synsem} \\
\text{SUBCAT} & \textit{subcat-type}
\end{bmatrix} \\
\text{CONT} & \textit{sem-obj}
\end{bmatrix} \\
\text{NONLOCAL} & \begin{bmatrix}
\text{INHERITED} \mid \text{SLASH: } \textit{null} \vee \textit{synsem} \\
\text{TOBIND} \mid \text{SLASH: } \textit{null} \vee \textit{synsem}
\end{bmatrix}
\end{bmatrix} \\
\text{DTRS} & \begin{bmatrix}
\textit{daughters} & \\
\text{HD-DTR} & \textit{sign} \\
\text{SPEC-DTRS} & \left\langle \textit{sign}, \ldots \right\rangle \\
\text{COMP-DTRS} & \left\langle \textit{sign}, \ldots \right\rangle
\end{bmatrix}
\end{bmatrix}
$$

Figure 4.1: Sample sign structure for Turkish

(99)
$$
\begin{bmatrix}
\textit{word} & \\
\text{PHON} & \left\langle \text{“seviyor”} \right\rangle \quad \text{\% love-}\textit{Prog-3Sg} \\
\text{SYNSEM} \mid \text{CAT} & \begin{bmatrix}
\textit{cat} & \\
\text{HEAD} & \textit{head} \\
\text{SUBJ} & \boxed{1} \\
\text{SUBCAT} & \left\{ \boxed{1}\ \text{NP[nom], NP[acc]} \right\}
\end{bmatrix}
\end{bmatrix}
$$

One of the major differences in sign structure is the SUBCAT feature. Since Turkish has free constituent order,[1] a type which is a nested combination of ordered and unordered lists is used instead of lists indicating obliqueness of the complements. Unordered lists are denoted with curly braces similar to sets.

Only the SLASH feature of type *null* ∨ *local* is defined as a nonlocal feature. Nonlocal features are used to process the information coming from arbitrary daughters (not only head daughter) which will be transmitted to upper phrases and bound to outer structures. Unbounded dependency and other binding constraints are defined by nonlocal features. HPSG defines three basic nonlocal features: SLASH, REL and QUE which are used to implement filler-gap dependencies, relative clauses and questions respectively. In this study, we only used the SLASH feature.

Since adjuncts may exist in any position preceding the head and probably between any subcategorized constituent, we have chosen to combine adjunct and complements of the phrase into one “daughters” attribute. Daughters (DTRS) attribute consists of adjunct daughters (ADJ-DTRS) and complement daughters (COMP-DTRS) which are lists

[1] For more information on order-freeness, see Chapter 3 and Section 4.3.

of *sign*. Head daughter (HD-DTR) is of type *sign*. If phrase has a subject complement, there is another feature subject complement (SUBJ-COMP) which is coindexed with the subject in the COMP-DTRS list.

4.2 Major Categories and Head Features

Since derivations involving category change is possible in Turkish, (e.g. relativizer `-ki` turning a noun into a specifier), and derived words preserve their syntactic behavior, head features must be extendable. For example, finite verbs are not nominalized hence do not carry case. However, sentential complements have inflections which make them behave as nominals and take case:

(100)	Eve	girdiğimizdeki	manzarayı	gördünüz.
	house-*Dat*	enter-*Part-1Pl-Loc-Rlvz*	view-*Acc*	see-*Past-2Pl*
	'You saw the view of the house when we entered.'			

Basic major categories are: nouns, verbs, adjectives, adverbs and conjunctives. Category information consists of HEAD, SUBJ and SUBCAT features. Each category has its own set of appropriate features in head attribute. HEAD feature is of type *head* which has the following categories defined as subsorts.

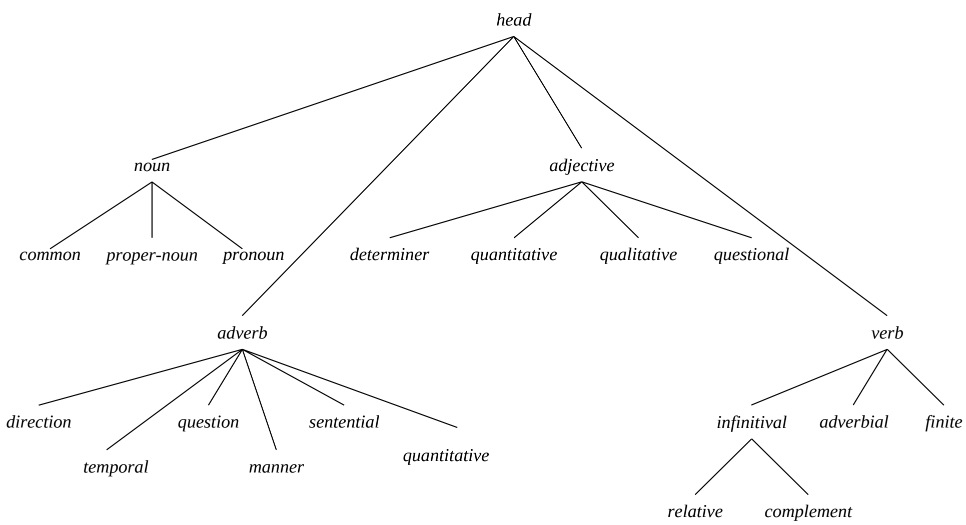


Figure 4.2: Sort hierarchy of type *head*

Appropriate features for the head sort *noun* are case, agreement, relativized, nominal-index and possession (cf. 101). CASE can be nominative, objective, genitive, dative, ablative, genitive or instrumental. In Turkish, there is no gender of objects so agreement(AGR) consists of person and number information. Since possession (if noun is used in the possessive group) is overtly marked, it is included as a head feature of type either *none* or agreement. Also feature nominal index (N-IND) is defined to handle adjuncts modifying the derived noun (see Section 4.5) and coindexed with the semantic index of the root noun.

(101)
$$\begin{bmatrix} \text{PHON} & \left\langle \text{``bahçelerimizin''} \right\rangle \quad \%\ \text{``of our garden''} \\ \text{SYNSEM} & \begin{bmatrix} \text{CAT} \mid \text{HEAD} & \begin{bmatrix} \mathit{common\text{-}noun} & \\ \text{CASE} & \mathit{genitive} \\ \text{AGR} & \begin{bmatrix} \mathit{agr} & \\ \text{PERSON} & \mathit{third} \\ \text{NUMBER} & \mathit{plural} \end{bmatrix} \\ \text{N-IND} & \boxed{1} \\ \text{POSS} & \begin{bmatrix} \mathit{agr} & \\ \text{PERSON} & \mathit{third} \\ \text{NUMBER} & \mathit{plural} \end{bmatrix} \end{bmatrix} \\ \text{CONT} \mid \text{INDEX} & \mathit{agr}\ \boxed{1} \end{bmatrix} \end{bmatrix}$$

As can be seen in Figure 4.2, head sort *verb* has three subsorts, *finite*, *infinitival* and *adverbial*. Finite verbs are the heads of the finite sentences. Infinitivals are verbal heads which modify or specify other phrases or subcategorized by other heads as complements. However, they still have the properties of verbs and construct embedded sentences. Relative clauses are verbs inflected by suffixes `-An, -dHk-`*Poss*`, -AcAk+`*Poss* and contain a gap which is filled by a following noun phrase. Another group of verbal heads are sentential complements which can be arguments of some verbs. These are verbs inflected by suffixes `-mA(k), -Hş-`*Poss*`, -dHk-`*Poss*, and `-AcAk-`*Poss* where the possessive suffix marks agreement rather than possession. Similarly, sentential adverbs modify the matrix verb.

In addition to the syntactic roles, there are structural differences in these three sorts. Finite verbs take an secondary tense or aspect marker which is one of *none, past, dubitative* or *conditional*. Infinitivals carry case information so they have the CASE feature. The following attribute definitions are appropriate for these groups:

(102) **verb:**

AGR	*null* ∨ *agr*
NEG	*plus* ∨ *minus*
TENSE	*base* ∨ *present* ∨ *continuous* ∨ *past* ∨ *dubitative* ∨ *wish*

finite:

AGR	*agr*
AUX-TENSE	*past* ∨ *dubitative* ∨ *conditional*

infinitival:

CASE	*case*

4.3 Complement Selection and Linear Precedence

Heads select their arguments using the SUBCAT feature. The SUBCAT feature is a structured type consisting of arguments of sort *synsem*. Therefore, a head can select any syntactic property of its arguments like category, case, agreement, nonlocal features and even semantic content. Any type of category is allowed including sentential complements, adverbs, adjectives, etc.

As mentioned in the preceding section, scrambling of constituents is handled by unordered lists. Linear precedence constraints of Turkish can be described generally as:

(103) a. $\begin{bmatrix} \text{HD-DTR} & \boxed{1}\begin{bmatrix}\text{SYNSEM}\,|\,\text{CAT}\,|\,\text{HEAD} & \neg\, verb\end{bmatrix} \\ \text{COMP-DTRS} & \left\langle ....,\boxed{2}\ sign\ ,....\right\rangle \end{bmatrix} \Longrightarrow \begin{bmatrix}\boxed{2}\end{bmatrix} < \begin{bmatrix}\boxed{1}\end{bmatrix}$

b. $\begin{bmatrix} \text{HD-DTR} & \boxed{1} \\ \text{ADJ-DTRS} & \left\langle ....,\boxed{2}\ sign\ ,....\right\rangle \end{bmatrix} \Longrightarrow \begin{bmatrix}\boxed{2}\end{bmatrix} < \begin{bmatrix}\boxed{1}\end{bmatrix}$

(103a) describes the constraint "complement daughters should precede the head daughter when head daughter is not a verb" and (103b) describes the constraint "adjunct daughters should precede the head daughter".

To handle the cases such as 'nonreferential object should immediately precede the verb', we use a special sort for *subcat* feature. This sort has a nested mixed structure of *list* and *set*.[2] Subcat type can be either a list or a set, and recursively, element of a set can be either a *synsem-arg* value or a *list*. Element of a list can be either a *synsem-arg* value or a *set* (104b). *synsem-arg* sort is used to enable optional arguments (104a). If

[2]Set is used to indicate the property of order-freeness where all permutations of the members of the set is possible at the surface.

OPT attribute of an argument is *plus* than it is optional and can be omitted. Optional arguments are generally denoted with enclosing parentheses.

(104) a.
$$\begin{bmatrix} \textit{synsem-arg} & \\ \textsf{OPT} & \textit{plus} \vee \textit{minus} \\ \textsf{ARG} & \textit{synsem} \end{bmatrix}$$

b. *subcat-type*

$$\begin{bmatrix} \textit{list-subcat} & \\ \textsf{HD} & \textit{synsem-arg} \vee \textit{set-subcat} \\ \textsf{TL} & \textit{e-list} \vee \textit{list-subcat} \end{bmatrix} \qquad \begin{bmatrix} \textit{set-subcat} & \\ \textsf{EL} & \textit{synsem-arg} \vee \textit{list-subcat} \\ \textsf{ELS} & \textit{e-list} \vee \textit{set-subcat} \end{bmatrix}$$

In the surface form, lists are ordered and sets are permuted. For example, the sign in (105a) has the surface forms listed in (105b). It is assumed that *adam* (man) is substituted for the subject, *çocuğa* (child-*Dat*) is substituted for the dative object, *evden* (house-*Abl*) is for *ablative* argument and *kalem* (pencil) is the nonreferential object.

(105) a.
$$\begin{bmatrix} \textsf{PHON} & \langle \text{`getirdi'} \rangle \quad \%\ \text{bring-}\textit{Past-3Sg} \\ \textsf{SYNSEM} \mid \textsf{LOCAL} \mid \textsf{CAT} & \begin{bmatrix} \textsf{HEAD} & \textit{verb} \\ \textsf{SUBJ} & \boxed{1} \\ \textsf{SUBCAT} & \left\langle \left\{ \boxed{1}\ \text{NP[nom], NP[dat], NP[abl]} \right\}, \text{NP[nom]} \right\rangle \end{bmatrix} \end{bmatrix}$$

b. “Adam çocuğa evden kalem getirdi”
“Adam evden çocuğa kalem getirdi”
“Çocuğa adam evden kalem getirdi”
“Çocuğa evden adam kalem getirdi”
“Evden çocuğa adam kalem getirdi”
“Evden adam çocuğa kalem getirdi”

Sentential complements and any kind of argument-head relation can be declared in this manner. For example the verb “söyledi” (told) can be defined as:

(106) $\begin{bmatrix} \textsf{PHON} & \langle \text{`söyledi'} \rangle \quad \text{\% tell-}\textit{Past-3Sg} \\ \textsf{SYNSEM} \mid \textsf{LOCAL} \mid \textsf{CAT} & \begin{bmatrix} \textsf{HEAD} & \textit{finite-verb} \\ \textsf{SUBJ} & \boxed{1} \\ \textsf{SUBCAT} & \{\text{NP[nom], NP[dat], S[inf,acc]}\} \end{bmatrix} \end{bmatrix}$

Where:

S[inf,acc]: $\begin{bmatrix} \textsf{SYNSEM} \mid \textsf{LOCAL} \mid \textsf{CAT} & \begin{bmatrix} \textsf{HEAD} & \begin{bmatrix} \textit{infinitival} & \\ \textsf{CASE} & \textit{acc} \end{bmatrix} \\ \textsf{SUBCAT} & \langle \rangle \end{bmatrix} \end{bmatrix}$

Verbal categories have the most complex complement structures. All verbal heads (finite verbs, infinitivals, sentential adverbs) require one or more complements according to their valence. Turkish is a complement-drop language so complements can be dropped even if they are obligatory arguments.

Other typical complement-head relationship is in the possessive noun group (107a). A possessive marked noun subcategorizes for a genitive noun and the part of speech of the complement should agree with the possessive suffix (107b).

(107) sarayın kapısı
palace-*Gen* door-*3SP*
'door of the palace'

$\begin{bmatrix} \textsf{PHON} & \langle \text{`kapısı'} \rangle \quad \text{\% ``door-}\textit{3SP}\text{''} \\ \textsf{SYNSEM} \mid \textsf{LOCAL} \mid \textsf{CAT} & \begin{bmatrix} \textsf{HEAD} & \begin{bmatrix} \textsf{common} & \\ \textsf{POSS} & \boxed{1}\ \textit{agr} \end{bmatrix} \\ \textsf{SUBCAT} & \langle \textsf{NP}_{gen}[\textsf{AGR}\ \boxed{1}] \rangle \end{bmatrix} \end{bmatrix}$

The order of the complements and adjuncts are variable which means adjuncts specifying the head can be in any position. So, instead of generating the surface form from the subcat list directly by a phrase structure rule, we chose to retrieve the complements one at a time. This allows the adjunct rule which will be described in following sections to be applied to the head at any position.

(108) $\begin{bmatrix} \textsf{SYNSEM} \mid \textsf{LOCAL} \mid \textsf{CAT} & \begin{bmatrix} \textsf{HEAD} & \boxed{1} \\ \textsf{SUBCAT} & \boxed{2} \end{bmatrix} \end{bmatrix} \longrightarrow$

$\begin{bmatrix} \textsf{SYNSEM} & \boxed{3} \end{bmatrix}, \begin{bmatrix} \textsf{SYNSEM} \mid \textsf{LOCAL} \mid \textsf{CAT} & \begin{bmatrix} \textsf{HEAD} & \boxed{1} \\ \textsf{SUBCAT} & \boxed{4} \end{bmatrix} \end{bmatrix}, \texttt{selectlast}(\boxed{3},\boxed{4},\boxed{2})$

Where `selectlast` selects the last *synsem* value ($\boxed{3}$) from the SUBCAT structure ($\boxed{4}$),

and rest is stored in third parameter ($\boxed{2}$).

This rule applies to the head-final complements. Handling scrambling of verbal head to pre-complement position is made possible by another schema:

(109)
$$\left[\text{SYNSEM}\,|\,\text{LOCAL}\,|\,\text{CAT}\ \left[\begin{array}{ll}\text{HEAD} & \boxed{1}\\ \text{SUBCAT} & \boxed{2}\end{array}\right]\right] \longrightarrow \left[\text{SYNSEM}\,|\,\text{LOCAL}\,|\,\text{CAT}\ \left[\begin{array}{ll}\text{HEAD} & \boxed{1}\ \textit{verb}\\ \text{SUBCAT} & \boxed{4}\end{array}\right]\right] \left[\text{SYNSEM}\ \boxed{3}\right], \texttt{selectfirst}(\boxed{3},\boxed{4},\boxed{2})$$

Where `selectfirst` selects the first *synsem* value ($\boxed{3}$) from the SUBCAT structure ($\boxed{4}$), and rest is stored in third parameter ($\boxed{2}$).

4.4 Pronoun Drop

One of the distinct properties of Turkish is the pronoun drop; pronoun in the subject position can be omitted since it is marked by agreement of the head. There are three constructs where pronouns drop: subject of the verbal heads, substantive predicates and possessive noun groups. In both cases, including embedded sentences in which the subject has genitive case, the dropped pronoun has either nominal or genitive case.

(110) a. (biz) Treni gördük.
We train-*Acc* see-*Past-1Pl*
'We saw the train.'

b. (benim) Güzel bahçem.
I-*Gen* nice garden-*1SP*
'My beautiful garden.'

c. (o) (benim) Eve gittiğimi gördü.
he I-*Gen* house go-*Part-1Sg-Acc* see-*Past-3Sg*
'He saw that I went to house.'

d. (o) (benim) En yakın arkadaşımdır.
(he) (I-*Gen*) most close friend-*1SP-Cop(3Sg)*
'He is my best friend.'

A solution to pro-drop is using *empty categories*, which have null surface forms. A possible declaration for dropped pronoun as empty category is:

(111)
$$\begin{bmatrix} \text{PHON} & \langle\rangle \\ \text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \textit{pronoun} & \\ \text{CASE} & \textit{nominative} \lor \textit{genitive} \end{bmatrix} \\ \text{SUBCAT} & \langle\rangle \end{bmatrix} \end{bmatrix}$$

This declaration will fill the subject position required by any head feature. However, empty categories usually cause major problems. In most of the implementations, they are inserted into any position available in the sentence. This is simply inefficient. More critically, when the order of the complement filled by the empty category has free order as it is in Turkish, superfluous parses are generated for each possible position that the subject can occupy. Therefore more constraints may be necessary to deal with the empty categories. The same problem also exists for management of the trace in relative clauses (Section 4.6). For the time being we have chosen the keep dropped pronouns as empty categories.

4.5 Adjuncts

Adjuncts are optional elements in the phrase structure. Adjuncts cannot be modeled in the same way as the complements. Their most distinct property is that they do not change the valence of the phrase they combined with. In other words, a head can be specified/modified by any number of adjuncts, which may possibly have the same category.

Another problem about adjuncts is whether the heads should select their adjuncts or adjuncts should select their heads. One solution proposed by Pollard and Sag [20] takes the approach where heads select their adjuncts. A new set-typed feature called *adjuncts* is added to sort *cat*, and adjunct is checked by whether it is unified with one of the elements of the set. The number of elements in the set does not change. However, adjuncts may come in many different varieties and this set may grow to an unmanageable size.

In the other approach, adjuncts select their heads [21]. This provides a simpler solution because the heads that an adjunct can modify are more restricted. MOD attribute of type *synsem* defined in the lexical entry fot the adjunct is used to select the syntactic category of the head. MOD is a head feature containing the restrictions for the head to be modified, and is unified with the SYNSEM value of the head. In examples (112a–b), the adjunct category of adjective/adverb subcategorizes for the head noun/verb respectively.

(112) a.
$$\begin{bmatrix} \text{PHON} & \langle\text{`mavi'}\rangle \quad \text{\% blue} \\ \text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \textit{qualitative-adj} & \\ \text{MOD} & \begin{bmatrix} \text{LOCAL} \mid \text{CAT} \mid \text{HEAD} & \textit{noun} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

b. $$\begin{bmatrix} \textsf{PHON} & \left\langle \text{‘çabuk’} \right\rangle \quad \text{\% fast} \\ \textsf{SYNSEM} \mid \textsf{LOCAL} \mid \textsf{CAT} & \begin{bmatrix} \textsf{HEAD} & \begin{bmatrix} \textit{adverb} & \\ \textsf{MOD} & \begin{bmatrix} \textsf{LOCAL} \mid \textsf{CAT} \mid \textsf{HEAD} & \textit{verb} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

With this model, all adjuncts have similar structure and can be handled by the same rule. In Turkish, an adjunct with the appropriate MOD attribute can precede the phrase anywhere. So a preliminary version of adjunct principle can be written as:

(113) $$\begin{bmatrix} \textsf{DTRS} & \begin{bmatrix} \textsf{HD-DTR} & \boxed{1} \\ \textsf{ADJ-DTRS} & \boxed{2} \oplus \boxed{3} \end{bmatrix} \\ \textsf{SYNSEM} \mid \textsf{LOCAL} \mid \textsf{CAT} \mid \textsf{HEAD} & \boxed{4} \end{bmatrix} \longrightarrow$$
$$\boxed{3} \begin{bmatrix} \textsf{SYNSEM} \mid \textsf{LOCAL} \mid \textsf{CAT} \mid \textsf{HEAD} \mid \textsf{MOD} \ \boxed{5} \end{bmatrix}, \boxed{1} \begin{bmatrix} \textsf{SYNSEM} & \boxed{5} \begin{bmatrix} \textsf{LOCAL} \mid \textsf{CAT} \mid \textsf{HEAD} & \boxed{4} \end{bmatrix} \\ \textsf{DTRS} & \begin{bmatrix} \textsf{ADJ-DTRS} & \boxed{2} \end{bmatrix} \end{bmatrix}$$

Although adjuncts can modify a phrase in any preceding position, there are restrictions on the possible combinations and order of the adjuncts modifying the same head. Rules defining the grammatical combinations vary; an adjunct modifying the head may prevent other adjuncts to modify the same head. In (114a) “güzel” modifies “bahçedeki” and does not modify “çiçek”. Similarly, the quantitative adjective “iki” cannot modify the noun phrase “bu çiçek”. However “iki” does not prevent “bu” from specifying “çiçek” (114b–c).

(114) a. güzel bahçedeki çiçek
beautiful garden-*Loc-Rlvz* flower
‘The beautiful flower in the garden’

b. * iki bu çiçek
two this flower

c. bu iki çiçek
this two flower
’These two flowers’

In order to control the combinations of adjuncts, we introduce a new feature for all categories under the CAT feature called ADJUNCTS. This structure consists of a group of boolean attributes that keep track of the adjuncts that have been applied to the category. In the adjunct part, the MOD attribute is divided into two attributes: a *synsem* value (MODSYN) with the same purpose of MOD in (113), and MODADJ defining the resulting

ADJUNCTS structure which will be projected to the mother phrase. Adjunct still selects the head together with the ADJUNCT value included in the SYNSEM of the head, and defines which flags will be set and passed to the mother phrase. For example, assume that ADJUNCTS consist of three flags: RLV indicating that the relativized noun has been applied, DEM indicating the demonstrative adjective has been applied and QLT indicating that the qualitative adjective is applied. Simplified lexical entries for each category could be as in the example (115).

(115) a.

$$\begin{bmatrix} \text{PHON} & \langle \textit{'güzel'} \rangle \quad \text{\% beautiful} \\ \text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \mid \text{HEAD} \mid \text{MOD} & \begin{bmatrix} \text{MODSYN} & \text{LOCAL} \mid \text{CAT} \mid \text{ADJUNCTS} \begin{bmatrix} \text{RLV} & - \\ \text{DEM} & - \end{bmatrix} \\ \text{MODADJ} & \begin{bmatrix} \text{RLV} & - \\ \text{DEM} & - \\ \text{QLT} & + \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

b.

$$\begin{bmatrix} \text{PHON} & \langle \text{'bu'} \rangle \quad \text{\% this} \\ \text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \mid \text{HEAD} \mid \text{MOD} & \begin{bmatrix} \text{MODSYN} & \text{LOCAL} \mid \text{CAT} \mid \text{ADJUNCTS} \begin{bmatrix} \text{RLV} & - \\ \text{DEM} & - \\ \text{QLT} & \boxed{1} \end{bmatrix} \\ \text{MODADJ} & \begin{bmatrix} \text{RLV} & - \\ \text{DEM} & + \\ \text{QLT} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

c.

$$\begin{bmatrix} \text{PHON} & \langle \text{'bahçedeki'} \rangle \quad \text{\% one that is in the garden} \\ \text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \mid \text{HEAD} \mid \text{MOD} & \begin{bmatrix} \text{MODSYN} & \text{LOCAL} \mid \text{CAT} \mid \text{ADJUNCTS} \begin{bmatrix} \text{RLV} & - \\ \text{DEM} & \boxed{1} \\ \text{QLT} & \boxed{2} \end{bmatrix} \\ \text{MODADJ} & \begin{bmatrix} \text{RLV} & + \\ \text{DEM} & \boxed{1} \\ \text{QLT} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

With these definitions, a revised adjunct principle can be written as:

(116)

$$\begin{bmatrix} \text{DTRS} & \begin{bmatrix} \text{HD-DTR} & \boxed{1} \\ \text{ADJ-DTRS} & \boxed{2} \oplus \boxed{3} \end{bmatrix} \\ \text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} & \begin{bmatrix} \text{HEAD} & \boxed{4} \\ \text{ADJUNCTS} & \boxed{6} \end{bmatrix} \end{bmatrix} \longrightarrow$$

$$\boxed{3} \begin{bmatrix} \text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \mid \text{HEAD} \mid \text{MOD} & \begin{bmatrix} \text{MODSYN} & \boxed{5} \\ \text{MODADJ} & \boxed{6} \end{bmatrix} \end{bmatrix},$$
$$\boxed{1} \begin{bmatrix} \text{SYNSEM} & \boxed{5} \begin{bmatrix} \text{LOCAL} \mid \text{CAT} \mid \text{HEAD} & \boxed{4} \end{bmatrix} \\ \text{DTRS} & \begin{bmatrix} \text{ADJ-DTRS} & \boxed{2} \end{bmatrix} \end{bmatrix}$$

When relative clauses, quantifiers, article 'bir', classifier nouns, and quantitative adjectives are defined, all noun phrase combinations can be covered. On the other hand, genitive noun in possessive noun group is not a specifier. It is an argument of the possessive noun. Thus it requires a special interpretation. Specifiers and modifiers can specify/modify the possessive marked noun as long as they are between the genitive noun and the possessive noun. Otherwise they specify/modify the genitive noun. To prevent adjuncts from passing over the genitive noun, we defined another constraint which can be informally expressed as: "a noun modifier/specifier modify/specify a possessive marked noun if it is not saturated". This constraint can be shown as:

(117)
$$\left[\text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \mid \text{HEAD} \mid \text{MOD} \mid \text{LOCAL} \mid \text{CAT} \quad \boxed{1}\left[\text{HEAD} \quad \begin{bmatrix} \textit{noun} \\ \text{POSS} \quad \neg\, none \end{bmatrix}\right]\right] \Longrightarrow \boxed{1}\left[\text{SUBCAT} \quad \neg \langle\rangle\right]$$

4.6 Relative Clauses

Filler-gap dependencies are the contracts in which elements are extracted from their positions (leaving gaps) and appear in other positions (filler). In Turkish, typical filler-gap construction is the relative clauses. Two basic strategies exist for relative clauses which are called *wa* and *ga* by Hankamer and Knecht [12] which are realized respectively by `-(y)An` and `-DHk-`*Agr* or `-(A)cAk-`*Agr* relative participles:

(118) i. When the gap is the relative clause subject, or a subconstituent of the relative clause subject, use the *wa* strategy.

ii. When there is no relative clause subject, use the *wa* strategy.

iii. When th gap is not a part of relative clause subject, use the *ga* strategy.

(119a–b) are examples of *wa*, (119c–d) are examples of *ga* strategy.

(119) a. $\_\_\__1$ Adama kalemi veren çocuğu$_1$ gördüm.
man-*Dat* pencil-*Acc* give-*Rel* child-*Gen* see-*Past-1Sg*
'I saw the child who gave man the pencil.'

b. $\_\_\__1$ yakınına köprü yapılan ev$_1$
near-*3SP-Dat* bridge build-*Pass-Rel* house
'The house$_1$ to which a bridge is built next $\_\_\__1$'

c. Çocuğun adama ___$_1$ vereceği kalemi$_1$ gördüm.
child-*Gen* man-*Dat* give-*Rel-3Sg* pencil-*Acc* see-*Past-1Sg*
'I saw the pencil$_1$ that the child will give ___$_1$ to the man.'

d. Çocuğun kalemi ___$_1$ verdiği adamı$_1$ gördüm.
child-*Gen* pencil-*Acc* give-*Rel-3Sg* man-*Acc* see-*Past-1Sg*
'I saw the man$_1$ to whom the child gave ___$_1$ the pencil.'

(118ii) introduces a special condition where the relative clause has no subject. In Turkish there are two cases for clauses with no subject [2]: impersonal passives and verbs with incorporated subject. In these cases, the real agent of the verb does not exist. The noun in the subject position incorporates to the verb. In the example (119b), 'köprü' is an incorporated subject. Similarly, examples below show the relativization of an adjunct NP—a locative adjunct in this case, with subject incorporation (120a) and no incorporation (120b).

(120) a.1 Kedi çocuğun yatağında uyudu.
cat child-*Gen* bed-*3SP-Loc* sleep-*Past-3Sg*
'The cat slept in the child's bed.'

a.2 yatağında kedi uyuyan çocuk
bed-*3SP-Loc* cat sleep-*Rel(wa)* child
'the child whose bed cat slept in'

b.1 Ayşe çocuğun yatağında uyudu.
child-*Gen* bed-*3SP-Loc* sleep-*Past-3Sg*
'Ayşe slept in the child's bed.'

b.2 yatağ-ın-da Ayşe'nin uyu-duğ-u çocuk
bed-*3SP-Loc* Ayşe-*Gen* sleep-*Rel(ga)-3Sg* child
'the child whose bed Ayşe slept in'

Gaps in relative clauses may involve dependencies which exist in nested constituents (121a–b). Infinitival verbs and possessives produce gaps from missing noun phrase constituents and pass them to the upper clause. This gap information is nonlocal to phrase, and projected until a verb with the relative suffix is reached. The clause headed by the verb behaves as a modifier and gap is filled (i.e, structure-shared) by the noun phrase at modified position.

(121) a. ___$_1$ Çocuğu kaybolan kadın$_1$ çok telaşlandı.
child-*3SP* lost-*Rel* woman very panic-*Past-3Sg*
'The woman whose child is lost has panicked.'

b. Babama ___$_1$ beğendiğimi söylediğim araba$_1$ satılmış.
father-*1SP-Dat* like-*Part-1Sg-Acc* tell-*Rel-1Sg* car sell-*Pass-Past-3Sg*
'The car$_1$ that I told my father that I like ___$_1$ is sold.'

Such dependencies and information interaction with the other phrases over the local phrase boundary are called *non-local features* by HPSG. These features are ruled by a principle called **Non-local Feature Principle** [21] which is adapted from **Foot Feature Principle** of GPSG. For filler-gap dependencies, a nonlocal feature called SLASH is introduced. In English more than one gap is possible in a clause, so set type is used for SLASH attribute. However in Turkish, a relative clause can contain only one trace at any intermediate phrase. In case of nested relative clauses, the gap is always filled and bound to a sister NP. Therefore in our design, SLASH attribute can be *null* or of type *local*. When the trace (empty category) is introduced, non-local feature SLASH is coindexed with the LOCAL feature of the gapped argument position.

(122)
$$\begin{bmatrix} \text{PHON} & \langle\rangle \\ \text{SYNSEM} & \begin{bmatrix} \text{LOCAL} & \boxed{1} \\ \text{NONLOCAL}\,|\,\text{INHERITED}\,|\,\text{SLASH} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Slash feature introduced by the trace is inherited to upper levels. However, in some position, inheritance should be broken and filler should be searched. In case of Turkish, this is the level where a relative verb is the head of the phrase. HPSG marks these positions by dividing NONLOCAL feature into two attributes INHERITED and TO-BIND of the same structure. TO-BIND|SLASH feature of relative verbs are marked as *local* (not *null*) and coindexed with the INHERITED|SLASH feature. The resulting phrase becomes a modifier of which the LOCAL feature of the modified phrase is also coindexed with the slash, so that the filler and its trace are combined (Figure 4.3).

A lexical entry for relativized verbs is given in (123). However, some head features such as case, relativization, possession and subcategorization are not supposed to be the same for the filler and the trace. To handle this, selected features CONT|INDEX, HEAD|AGR are passed to modified structure instead of LOCAL feature.

(123)
$$\begin{bmatrix} \text{PHON} & \langle\text{`söylediğim'}\rangle \quad \%\ \text{tell-}\textit{Rel-1Sg} \\ \text{SYNSEM} & \begin{bmatrix} \text{LOCAL}\,|\,\text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \textit{obj-rel-verb} & \\ \text{AGR} & \begin{bmatrix} \text{PERSON} & \textit{first} \\ \text{NUMBER} & \textit{sing} \end{bmatrix} \\ \text{MOD}\,|\,\text{MODSYN}\,|\,\text{LOCAL}\,|\,\text{CONT}\,|\,\text{INDEX} & \boxed{1} \end{bmatrix} \end{bmatrix} \\ \text{NONLOCAL} & \begin{bmatrix} \text{INHERITED}\,|\,\text{SLASH} & \textit{null} \\ \text{TO-BIND}\,|\,\text{SLASH} & \begin{bmatrix} \textit{local} & \\ \text{CONT}\,|\,\text{INDEX} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

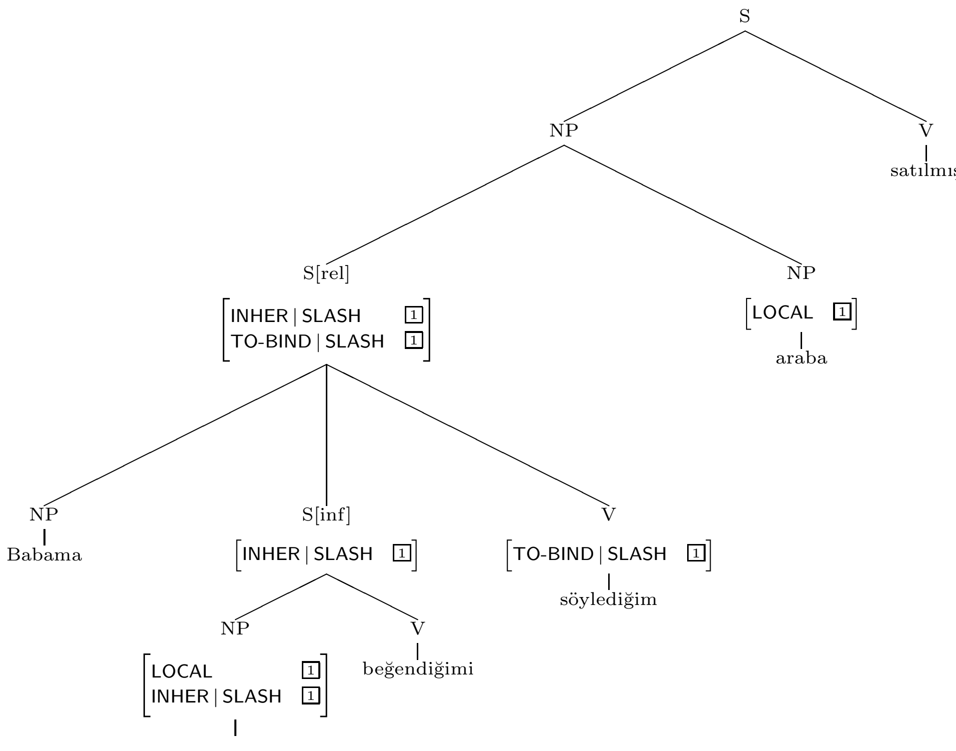


Figure 4.3: Projection of the SLASH feature

The problem with the dropped pronouns also exists in relative clauses. When trace is realized with empty category, efficiency and superfluous ambiguity problems may arise. In our design, we used a simple technique for raising slash feature, relying on two properties: First is valid for most of the languages. Every trace should be subcategorized by a head. Second is the free constituent order of Turkish. Since complement order is relatively free in Turkish, we could assume that the missing item is the last constituent. Because any constituent may be in the last (first in the surface but retrieved last) position in the complement list. So we have introduced the following rule to introduce trace instead of empty category:

(124)
$$\begin{bmatrix} \textsf{SYNSEM} \mid \textsf{LOCAL} \mid \textsf{CAT} & \begin{bmatrix} \textsf{HEAD} & \textit{head} \\ \textsf{SUBJ} & \boxed{1} \\ \textsf{SUBCAT} & \langle\rangle \end{bmatrix} \\ \textsf{DTRS} & \begin{bmatrix} \textsf{COMP-DTR} & \boxed{2} \oplus \boxed{1} \end{bmatrix} \end{bmatrix} \longrightarrow$$
$$\begin{bmatrix} \textsf{SYNSEM} \mid \textsf{LOCAL} \mid \textsf{CAT} & \begin{bmatrix} \textsf{HEAD} & \textit{head} \\ \textsf{SUBCAT} & \boxed{3} \end{bmatrix} \\ \textsf{DTRS} \mid \textsf{COMP-DTRS} & \boxed{2} \end{bmatrix},$$
$$\texttt{select}\left(\boxed{1} \begin{bmatrix} \textsf{LOCAL} & \boxed{4} \\ \textsf{NONLOCAL} \mid \textsf{INHERITED} \mid \textsf{SLASH} & \boxed{4} \end{bmatrix}, \boxed{3}, \langle\rangle \right)$$

When the argument is the last item in the SUBCAT list, it is deleted, and the trace is introduced. This solves the ambiguity in subcategorized constituents. However yet another problem exists with the traces which may occur in adverbs, which are not subcategorized for. In English, prepositions subcategorize for an NP so that the trace could be generated from subcategorization. However two case suffixes `-dA` (locative) and `-(y)lA` (instrumental) in Turkish produce nominal adjuncts that act as VP modifiers. When they are missing, since they have no surface form, it is impossible to introduce them by the rule above. We introduce the trace as an empty category for these two cases:

(125)
$$\begin{bmatrix} \textsf{PHON} & \langle\rangle \\ \textsf{SYNSEM} & \begin{bmatrix} \textsf{LOCAL} & \boxed{1} \begin{bmatrix} \textsf{CAT} \mid \textsf{HEAD} & \begin{bmatrix} \textit{noun} & \\ \textsf{CASE} & \textit{inst} \vee \textit{locative} \\ \textsf{MOD} \mid \textsf{MODSYN} \mid \textsf{LOCAL} \mid \textsf{CAT} & \begin{bmatrix} \textsf{HEAD} & \textit{verb} \\ \textsf{SUBCAT} & \langle\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \textsf{NONLOCAL} \mid \textsf{INHERITED} \mid \textsf{SLASH} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

The second problem is the definition of the constraints for the use of *wa* and *ga* strategies described in (118). When the slash value is introduced in the subject position (subject daughter or one of its daughters is missing), *wa* strategy is used, otherwise *ga* strategy is used. These are expressed as:

Relative Clause Principle

(126) a)
$$\begin{bmatrix} \textsf{SYNSEM} & \boxed{1} \begin{bmatrix} \textsf{LOCAL} \mid \textsf{CAT} & \begin{bmatrix} \textsf{HEAD} & \textit{subject-relative} \\ \textsf{SUBCAT} & \langle\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix} \Longrightarrow$$
$$\boxed{1} \begin{bmatrix} \textsf{LOCAL} \mid \textsf{CAT} \mid \textsf{SUBJ} & \begin{bmatrix} \textsf{NONLOCAL} \mid \textsf{INHERITED} \mid \textsf{SLASH} & \boxed{2} \end{bmatrix} \\ \textsf{NONLOCAL} \mid \textsf{TO-BIND} & \boxed{2} \end{bmatrix}$$

b) $$\left[\text{SYNSEM}\ \left[\boxed{1}\ \text{LOCAL}\,|\,\text{CAT}\ \begin{bmatrix}\text{HEAD} & \textit{object-relative}\\ \text{SUBCAT} & \langle\rangle\end{bmatrix}\right]\right] \Longrightarrow$$
$$\boxed{1}\begin{bmatrix}\text{LOCAL}\,|\,\text{CAT}\,|\,\text{SUBJ} & \left[\text{NONLOCAL}\,|\,\text{INHERITED}\,|\,\text{SLASH}\ \ \boxed{2}\right]\\ \text{NONLOCAL}\,|\,\text{TO-BIND} & \neg\,\boxed{2}\end{bmatrix}$$

In cases where the subject NP is incorporated, the *wa* strategy can be used even though the gap is not the subconstituent of the subject. Such verbs are marked with a boolean head feature called N-INCORP standing for the noun incorporation. The main constraint on this type of relative clause is that the type of the noun in the subject position should be indefinite (or nonreferential) because it is incorporated. The following additional constraint solves the problem:

(127) c) $$\left[\text{SYNSEM}\ \ \boxed{1}\left[\text{LOCAL}\,|\,\text{CAT}\ \begin{bmatrix}\text{HEAD} & \begin{bmatrix}\textit{subject-relative}\\ \text{N-INCORP} \ \ +\end{bmatrix}\\ \text{SUBCAT} & \langle\rangle\end{bmatrix}\right]\right] \Longrightarrow$$
$$\boxed{1}\left[\text{LOCAL}\,|\,\text{CAT}\,|\,\text{SUBJ}\ \ \left[\text{LOCAL}\,|\,\text{CAT}\,|\,\text{ADJUNCTS}\,|\,\text{DEFINITE}\ \ -\right]\right]$$

When a relative clause satisfying these constraints is saturated, and its TO-BIND feature is bound to the INHERITED feature, it acts as a noun modifier. The content index of the modified noun is coindexed with the index of the gap to bind the semantic features of the relative clause and the filler. The rest is handled by the adjunct schema. Nested relative clauses can modify the same noun so that multiple gaps may be bound to the same filler (128).

(128)

annemin	$\_\_\_\__1$	yaptığı	$\_\_\_\__1$	çok	sevdiğim	kurabiyeler$_1$
mother-POSS		cook-*Rel*		much	like-*Rel*	cookie-*Plu*

‘the cookies that my mother cooked, that I like’

4.7 Substantive Predicates

As mentioned in Chapter 3, Turkish sentences may have verbal, existential or substantive heads. Substantive predicates are formed by substantive heads with auxiliary (`-DH-`*Agr*, `-mHş-`*Agr*) or copula suffixes. Syntactically, substantive heads subcategorize for an NP which have the same semantic index. In other words, substantive head and the subcategorized NP describe the same nominal object (129a–b). Copula and agreement suffixes marks the agreement of the categorized NP.

(129) a. Ben çok hastayım.
I much ill-*Cop(1Sg)*
'I'm too sick.'

b. Bütün kadınlar çiçektir.
every woman-*Plu* flower-*Cop(3Sg)*
'Every woman is a flower.'

Another design consideration is to distinguish predicative NP's from the others. First, this is necessary to determine whether a saturated NP forms a sentence or not. Second, the same problem with the possessive NP exists for substantive predicates. A saturated predicative NP should not be further modified by another adjective. For these two reasons, we have added a boolean type head feature for substantial types called PREDICATIVE. The following is a sample entry for a predicative noun "insanım".

(130)
$$\begin{bmatrix} \text{PHON} & \text{“insanım”} \quad \text{\% Human-}\textit{Cop(1Sg)}\text{,‘I am a human’} \\ \text{SYNSEM}\,|\,\text{LOCAL} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \textit{noun} & \\ \text{PREDICATIVE} & + \end{bmatrix} \\ \text{SUBCAT} & \left\langle \text{NP}_{1sg\boxed{1}} \right\rangle \end{bmatrix} \\ \text{CONT}\,|\,\text{INDEX} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

After this feature is defined, the following constraint is added to the constraint (117) for the adjunct rule:

(131)
$$\begin{bmatrix} \text{SYNSEM}\,|\,\text{LOCAL}\,|\,\text{CAT}\,|\,\text{HEAD}\,|\,\text{MOD}\,|\,\text{LOCAL}\,|\,\text{CAT} & \boxed{1}\begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{noun} & \\ \text{PREDICATIVE} & + \end{bmatrix} \end{bmatrix} \end{bmatrix} \Longrightarrow \boxed{1}\begin{bmatrix} \text{SUBCAT} & \neg\langle\rangle \end{bmatrix}$$

In the implementation, substantive predicates are realized by a lexical rule which maps lexical entry for a non-predicative substantive word to substantive predicate by an auxiliary or copula suffix:

(132)
$$\begin{bmatrix} \text{PHON} & \boxed{1} \\ \text{SYNSEM}\,|\,\text{LOCAL} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \textit{subst} & \\ \text{PREDICATIVE} & - \end{bmatrix} \\ \text{SUBCAT} & \boxed{2} \end{bmatrix} \\ \text{CONT}\,|\,\text{INDEX} & \boxed{3} \end{bmatrix} \end{bmatrix} \longmapsto$$
$$\begin{bmatrix} \text{PHON} & \texttt{apply-cop}(\boxed{1},\boxed{5}) \\ \text{SYNSEM}\,|\,\text{LOCAL} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{PREDICATIVE} & + \end{bmatrix} \\ \text{SUBCAT} & \text{NP[AGR }\boxed{5}\text{,INDEX }\boxed{3}] \oplus \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Where `apply-cop` is a general predicate applying the copula suffix corresponding to agreement feature marked with the second argument to the first argument and returning the resulting string.

Chapter 5

ALE IMPLEMENTATION

In the implementation, we have used ALE (Attribute Logic Engine)[5]. ALE is an integrated system of definite clause logic programming and phrase structure parsing. All operations and declarations in ALE use *typed feature structures* as terms. ALE is designed and suited for implementations of unification-based language formalisms.

ALE is a strongly typed language. Every structure must have a declared type. Types are defined by an inheritance structure and subtype relation. Basic representation scheme used is the typed feature structures. Types are assigned to appropriate feature-value pairs. Type structure of ALE is very similar to the HPSG including properties like inheritance, nesting, and well-typedness. However it is most restricted in favor of efficiency and implementation considerations.

ALE allows definition of general constraints on types. One can put restrictions on the feature structures of a particular type. Another feature of ALE is the definite clauses in which all functionality of PROLOG definite clauses is provided with feature structure unification instead of simple term unification. Also complex descriptions can be simplified by the use of macros.

One of the most distinct features of ALE from other tools like TFS and CUF [17] is the support for phrase structure grammars. ALE provides phrase structure rules to be coded like Definite Clause Grammars of PROLOG. It has a built-in bottom-up chart parser in addition to feature structure unification. DCG's are top-down and depth-first. However ALE parser works in a combined manner asserting edges to chart right to left while applying rules left to right. ALE also allows lexical rules for dealing with lexical redundancy. Lexical rules can be defined for inflectional or derivational morphology as well as zero derivations like nominalization of adjectives. Morphological constraints (suffixation, affixation etc.) can be controlled by some built in mechanisms or PROLOG predicates. It also allows empty categories to be integrated into grammar.

5.1 Grammar Rules and Principles

We have four phrase structure rules, each corresponding to a schema that we have introduced in the preceding chapter. First two (108, 109) handle the complement retrieval and subcategorization. Applying both rules cause superfluous parses due to the application order. The third is the adjunct schema (116) which handles the adjunct-head relation. And the fourth is the rule introducing the slash (124).

Rules are coded by standard Immediate Dominance and Linear Precedence notation. The application of rules are governed and constrained by a set of ALE definite clauses. These include two simple clauses modifying the DAUGHTER and PHON features of the mother phrase. The others are constraints and basic principles.

`head-feature-principle` applies the **Head Feature Principle** of HPSG; head feature of the mother is structure shared with the head feature of the head daughter. `selectlast` and `selectfirst` implements the **Subcategorization Principle** of HPSG. The surface form of the combined SUBCAT structure of list and sets with optional arguments is generated, one item is selected, and rest is returned as the SUBCAT feature of the mother phrase. Figure 5.1 shows the source for the simplified versions of these two principles.

```
%
%  head-feature-principle(MothSign,HeadSign)
%
head-feature-principle(synsem:local:cat:head:X,synsem:local:cat:head:X)
                  if true.


%
%  subject-retrieval-rule
%
subcat_retr1 rule
(Mother, synsem:local:cat:subcat:SubcatRest) ===>
            cat>  (Complement,synsem:CompSyn),
            cat>  (Head,synsem:local:cat:subcat:Subcat),
            goal> (head-feature-principle(Mother,Head),
                   seleclast(CompSyn,Subcat,SubcatRest)).
```

Figure 5.1: Sample Source for Head Feature and Subcategorization Principles

Linear precedence and word order constraints of the system are realized by the definite clauses `removeop` and `surface`. The mixed subcat structure (cf. 104) consisting of nested sets and lists with optional arguments is converted into permutations of the surface form by these clauses. `removeop` produces omitted and existed permutations of optional arguments, and `surface` produces the flattened lists from the resulting structure.

`nonlocal-principle` combines the NONLOCAL features of the daughters and modifies the mother. It inherits the gap information in one of the daughters from INHERITED feature to the mother phrase. If TO-BIND feature of the head is not *null* it binds the gap and applies the constraint for the relative clauses. `adjunct-principle` puts the constraints in schema (116) and two constraints introduced in (117) and (131). Grammar rules are coded in file `T.rule` and principles are coded in `T.clause` in Appendix A.

5.2 Lexicon and Lexical Rules

Lexical redundancy becomes a crucially important problem in agglutinative languages where a large number of derivations and inflections of a root word exists. It is almost impossible to store all derivations and inflections of Turkish words into a lexical database. Therefore some sort of morphological analysis and application of lexical rules are essential. Also application of a lexical inheritance hierarchy could be used to deal with redundancy.

ALE does not suggest a standart mechanism for implementing lexical type hierarchies. Two methods seem to be applicable: use of macros, and type hierarchies with general constraints. Macro definitions in ALE allow for variable substitution and —not recursive— nesting. Each node in the lexical hierarchy can be defined by a macro which contains the calls for the parent macros. At the lexical level, items are defined by one or more macros. Macros are expanded at run time, and ALE operates on the expanded descriptions. Second solution is to construct a type hieararchy under the type *word* which is the type of the lexical elements, and put general constraints on these types. However, in this approach, it is impossible to assign a lexical item to several nodes in the type hierarchy since each item can belong to only one type. Besides, ALE has a very restricted type mechanism so multiple inheritence is very limited. On the other hand, constraints are evaluted at compile time which is efficient compared to macros.

In implementation, we have used a lexical type hierarchy. We defined a lexical type tree under the type *word* for the lexical entries. Each common class of lexical entries are defined as a node in the tree (Figure 5.2). We have used "`_l`" as the last two characters in the names of these lexical types to distinguish them from *head* types.

We also defined constraints on these types (see source files `T.type` and `T.cons` in Appendix A). After all these declarations, defining a lexical entry as one of the types above will apply all constraints associated with the supertypes along the path to *word*.

ALE supports lexical rules. At the feature structure level, unification and use of user defined ALE definite clauses are provided. At the surface level, the user can define string operations by concatenation and user defined PROLOG predicates. Use of PROLOG at string level enables user to implement any kind of complex morphological phenomena like affixation, vowel harmony, drops etc. Also complex structure changes in signs can be coded by the help of the ALE's definite clauses.

However, ALE lexical rules are inefficient for an agglutinative language like Turkish. First, ALE applies all rules at compile time and asserts all generated combinations as PROLOG predicates which consumes too much memory and increases compilation time

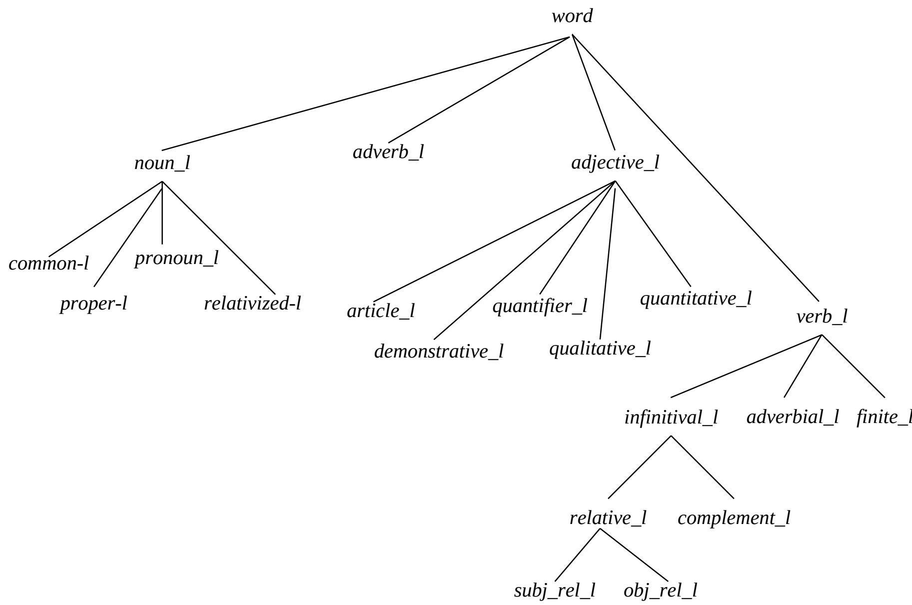


Figure 5.2: Lexical hierarchy

considerably. Second, it implements all kinds of string-to-string mappings to handle a wide range of languages. However this makes it very slow for a large number of lexical rules and lexical entries. Both inefficiencies make the developement very difficult. So we have tried to edit some portion of ALE source code and made some changes which will apply lexical rules at runtime and apply lexical mappings only to closed class of words. This increased the efficiency to a reasonable level.

We have coded lexical rules for nominal casess, possesive suffixes (which mark agreement in case of sentential complements and relative clauses), and noun relativizer suffix. Zero derivations like adjective-to-noun promotion and production of non-referential object case of verbs are implemented by lexical rules. These lexical rules are defined in the file `T.lex_rule` (Appendix A). For reusability considerations, similar feature structure transformations are grouped into ALE definite clauses which are defined in the file `T.clause`. For example, rules for application of case suffixes are implemented in clause `apply-case/3` and lexical rules for all cases call this clause with case passed as argument.

In the appendix A, some part of grammar code is given. It requires Quintus or SICStus Prolog. Full system can be obtained via anonymous ftp from `ftp.lcsl.metu.edu.tr` in path `/pub/theses/sehitoglu-ms-96.tar.gz`.

Chapter 6

CONCLUSION

In recent years, computational studies on Turkish have proliferated. These studies are important in two respects: First, building foundations of linguistic description of Turkish within the light of the contemporary linguistic theories. Second, providing basic tools for natural language processing which has applications in computer science ranging from simple text processing utilities to translation and learning tools.

HPSG is the synthesis of the some of the recent linguistics theories. It is a developing theory, and new principles and approaches are being introduced for expanding the universal coverage. Being one of the most powerful among the other unification based and phrase structure formalisms, it models the language in informational perspective and describes the linguistic events by a set of universal principles and metarules. It is a general theory trying to be as flexible as possible to cover principles of all natural languages.

In this study, we have worked on a computational sign-based model of Turkish, following and adapting the HPSG framework. HPSG uses feature structures to describe linguistic phenomena. This allows the grammar designers to concentrate on the *constraints* imposed by a particular language on a well-defined set of linguistic features. This is in contrast to earlier context-free grammar rules, where language-specific rules do not allow generalizations. Postulating principles and writing constraints on these principles show how different languages model the same phenomenon in different ways. To this end, we have analyzed and implemented the general principles such as subcategorization, adjunct-head selection, relative extraction. We have also studied the principles such as word-order variation, pronoun and complement drops and unbounded dependencies, which are particularly important for Turkish.

For the time being, the parser has not been combined with a lexical analyzer and tested on a real corpus. Since most of the syntactic information is coded in the lexicon, an intelligent mechanism for gathering all lexical entries for Turkish should be employed. HPSG proposes solutions like lexical inheritance hierarchy, and lexical rules. Turkish is an agglutinative language and has many syntactically effective and productive suffixes. This means that there is more interactions between morphology and syntax, compared to a language such as English.

The computational tool we have used for HPSG, ALE, supports lexical rules with morphological analysis. However it is inefficient for running a grammar with large lexicon and all lexical rules. As the main problem about lexicon, ALE does all lexical processing at compile time and generates all possible results of lexical rules statically, which is not suitable for agglutinative languages. Also, since morphological rules are defined in PROLOG, they are very inefficient. We made some changes to apply lexical rules at run time and make morphology a little bit faster. However, for an efficient lexical analysis, use of an external lexicon and morphological analyser is necessary. Necessary interface routines can be coded into PROLOG source code of ALE as the changes we have already done.

Another approach could be integration of syntactic and morphological analysis. This is achived by encoding morphological analysis combined with syntactic rules in the style of HPSG principles. This is also desirable from the linguistic point of view; morphological and syntactic phrasing can affect each other in a principled way.[1]

ALE has some drawbacks as well as powerful features. The strong typing cause description domain to be restricted. Type hierarchies requiring latice-like multiple inheritances cannot be coded efficiently. Also it disallows the usage of atomic types without type declaration. It has a unification based description language and type inferencing mechanism provided with definite clauses with all functionality of PROLOG. However ALE lacks some sort of *overwrite* operation especially in lexical rules which are procedural in nature. Overwrite operation changes some part of a feature rather than unifying it. Such an operator may ease the formulations and descriptions of lexical rules.

We have limited semantic analysis of signs to minimum. Since HPSG is a complete linguistic theory for both syntax and semantics, for a complete HPSG analysis of Turkish, semantic principles and model should be analyzed.

[1] for an integrated analysis of morphology and syntax cf. [3]

REFERENCES

[1] Tahsin Banguoğlu. *Türkçenin Grameri*. Türk Dil Kurumu, 1986.

[2] Chris Barker, Jorge Hankamer, and John Moore. Wa and ga in turkish. In K. Dziwirek, P. Farrell, and E. Mejias-Bikandi, editors, *Grammatical Relations*. 1990.

[3] Cem Bozşahin and Elvan Göçmen. A categorial framework for composition in multiple linguistic domains. In *Proceedings of the Fourth International Conference on Cognitive Science of NLP (CSNLP'95)*, Dublin, Ireland, July 1995.

[4] Joan Bresnan and R. Kaplan. *The Mental Representation of Grammatical Relations*, chapter Lexical-Functional Grammar, pages 173–281. MIT Press, 1982.

[5] Bob Carpenter and Gerald Penn. *The Attribute Logic Engine User's Guide, Version 2.0*. Carnegie Mellon University, Pittsburgh, August 1994.

[6] Jochen Dörre and Micheal Dorna. *CUF: A Formalism for Linguistic Knowledge Representation*. Dyana, August 1993.

[7] E. E. Erguvanlı. *The Function of Word Order in Turkish Grammar*. PhD thesis, UCLA, 1979.

[8] G. Gazdar, E. Klein, G. K. Pullum, and I. A. Sag. *Generalized Phrase Structure Grammar*. Harvard University Press, 1985.

[9] Elvan Göçmen and Onur T. Şehitoğlu and Cem Bozşahin. An outline of turkish syntax. Technical Report 95–2, METU Department of Computer Engineering, Ankara, Turkey, March 1995.

[10] Zelal Güngördü. A lexical-functional grammar for turkish. Master's thesis, Dept. of Comp. Eng., Bilkent University, July 1993.

[11] Zelal Güngördü and Kemal Oflazer. Parsing turkish using the lexical-functional grammar. In *Proceedings of COLING'94, The 15th Conference on Computational Linguistics*, Kyoto, Japan, August 1994.

[12] Jorge Hankamer and Laura Knecht. The role of the subject/non-subject distinction in determining the choice of relative clause participle in turkish. In *Harvard Syntax and Semantics 2*. Cambridge, 1976.

[13] Beril Hoffman. *The Computational Analysis of the Syntax and Interpretation of 'Free' Word Order in Turkish*. PhD thesis, University of Pennsylvania, 1995.

[14] Ray Jackendoff. *$\overline{X}$-syntax: A Study of Phrase Structure*. MIT Press, Cambridge, 1977.

[15] Jonas Kuhn. Encoding HPSG grammars in TFS. Technical report, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart, Germany, March 1993.

[16] Geoffrey L. Lewis. *Turkish Grammar*. Oxford University Press., Oxford, UK, 1967.

[17] Suresh Manandhar. CUF in context. Technical report, Human Comm. Research Centre, University of Edinburgh.

[18] Robert H. Meskill. *A Transformational Analysis of Turkish Syntax*. Mouton, 1970.

[19] Teruko Mitamura, Hiroyuki Muska, and Marion Kee. *The Generalized LR Parser/Compiler Version 8.1: User's Guide*. Carnegie Mellon, April 1988.

[20] Carl Polard and Ivan A. Sag. *Information Based Syntax and Semantics*. CSLI, 1987.

[21] Carl Polard and Ivan A. Sag. *Head-driven Phrase Stucture Grammar*. CSLI Chicago, 1994.

[22] W. Rounds and R. Kasper. A complete logical calculus for record structures representing linguistic information. In *Proceedings of the IEEE Symposium on Logic in Computer Science*, June 1986.

[23] G. J. Schaaik. *Studies in Turkish Grammar*. PhD thesis, University of Amsterdam, February 1995.

[24] Peter Sells. *Lectures on contemporary Syntactic Theories*. CSLI, 1985.

[25] Stuart M. Shieber. *An Introduction to Unification Based Approaches to Grammar*. CSLI, 1986.

[26] Rasim Şimşek. *Örneklerle Türkçe Sözdizimi*. Kuzey Matbaacılık, 1987.

[27] Robert Underhill. *Turkish Grammar*. MIT Press, Cambridge, 1975.

Appendix A

PARSER SOURCE

A.1 Type Definitions

```
%======================================================================
%
%  @(#)T.type               Rev:1.6                 1/7/96
%
%======================================================================
%  Type definitions
%======================================================================


bot sub [bool,sign,null_synsem,cat,head,case,null_agr,per,num,posses,
                list,char,set,tense,aux_tense,nonloc,null_adjstr,null_mod,
                list_or_set_subcat,subcat_or_ne_set,subcat_or_ne_list,
                psoa_arg,qfpsoa,sem_det,sem_obj,null_local,arg_type,
                subcat,conx,null_local,nonlocal,daughters].



sign sub [lexical,phrase]
   intro [phon:list_string,
          synsem:synsem,
          qstore:set_quant,
          qretr:list_quant].
  lexical sub [word].
      word sub [noun_l,adj_l,adv_l,verb_l].
        noun_l sub [common_l,proper_l,pronoun_l,relativized_l].
           common_l sub [].
           proper_l sub [].
           pronoun_l sub [].
           relativized_l sub [].
        adj_l sub [quantif_l,article_l,demonstra_l,quantitive_l,
                                             qualitative_l].
           quantif_l sub [].
           article_l sub [].
           demonstra_l sub [].
```

```
        quantitive_l sub [].
        qualitative_l sub [].
      adv_l sub [].
      verb_l sub [finite_l,sadv_l,inf_l].
        finite_l sub [].
        sadv_l sub [].
        inf_l sub [relcl_l,complement_l].
          relcl_l sub [subj_rel_l,obj_rel_l].
            subj_rel_l sub [].
            obj_rel_l sub [].
          complement_l sub [].
  phrase sub []
     intro [ dtrs: daughters ].

null_synsem sub [null,synsem].
  null sub [].
  synsem sub []
     intro [local:local,
              nonlocal:nonlocal].

null_local sub [null,local].
local sub []
   intro [cat:cat,cont:sem_obj,conx:conx].


conx sub [].

nonlocal sub []
   intro [inherited:nonloc,tobind:nonloc].

nonloc sub []
   intro [slash:null_local].

daughters sub [hd_subj_st,hd_st]
    intro [hd_dtr: sign,
           comp_dtrs: list_sign,
           spec_dtrs: list_sign].
 hd_subj_st sub []
    intro [subj_dtr:sign].
 hd_st sub [].


bool sub [plus,minus].
  plus sub [].
  minus sub [].

cat sub []
   intro [head:head,
          subj:null_synsem,
          adjuncts:null_adjstr,
          subcat:list_or_set_subcat].

head sub [subst,prep,adverb,verb]
    intro [mod:null_mod].
  subst sub [adj,noun]
    intro [pred:bool].
  adj sub [determiner,quantitative_adj,qualitative_adj,questional_adj]
      intro [countable:bool,gradable:bool].
    determiner                sub [article,demonstrative_adj,quantifier].
      article                sub [].
      demonstrative_adj        sub [].
      quantifier       sub [].
    quantitative_adj        sub [number,distributive_adj,grouping_adj].
      number                 sub [cardinal,fractional].
        cardinal               sub [].
```

```
        fractional        sub [].
      distributive_adj        sub [].
      grouping_adj        sub [].
    qualitative_adj        sub [].
    questional_adj        sub [].

  noun sub [common,proper_noun,pronoun]
     intro [case:case,
            agr:agr,
            n_ind:agr,
            rel:bool,
            poss:posses].
    common sub [].
    proper_noun sub [].
    pronoun sub [personal_pr,demonstrative_pr,reflexive_pr,indefinite_pr
                        ,quantificational_pr,questional_pr].
        personal_pr                sub [].
        demonstrative_pr         sub [].
        reflexive_pr                sub [].
        indefinite_pr                sub [].
        quantificational_pr         sub [].
        questional_pr                sub [].
  prep sub [].
  adverb sub [dir_adv,dir_adv,temp_adv,manr_adv,quant_adv,sent_adv,quest_adv].
    dir_adv sub []
       intro  [dir:direction].
    temp_adv sub [t_unit_adv,pot_adv,t_per_adv].
      t_unit_adv sub [].
      pot_adv sub [].
      t_per_adv sub [dayt,dayw,seas].
        dayt sub [].
        dayw sub [].
        seas sub [].
    manr_adv sub [qual_adv,rep_adv].
      qual_adv sub [].
      rep_adv sub [].
    quant_adv sub [approx,comp,superl,excess].
      approx sub [].
      comp sub [].
      superl sub [].
      excess sub [].
    sent_adv sub [].
    quest_adv sub [].


  verb sub [infinitival,adverbial,finite]
      intro [tense:tense,neg:bool,vagr:null_agr,n_inc:bool].
    infinitival sub [relative,complementary]
        intro [vcase:case].
      relative sub [subj_rel,obj_rel].
        subj_rel sub [].
        obj_rel sub [].
      complementary sub [mak,ış,complemented].
        mak  sub [].
        ış sub [].
        complemented sub [].
    adverbial sub [].
    finite sub []
        intro [aux_tense:aux_tense].

case sub [nom,obj,gen,loc,direction,ins].
  nom sub [].
  obj sub [].
  gen sub [].
  loc sub [].
```

```
  direction sub [dat,abl].
    dat sub [].
    abl sub [].
  ins sub [].

null_agr sub [null,agr].
  agr sub []
    intro [per:per,
           num:num].

per sub [first,second,third].
  first sub [].
  second sub [].
  third sub [].

num sub [sing,plur].
  sing sub [].
  plur sub [].

posses sub [none,poss].
  none sub [].
  poss sub []
     intro [by:agr].

null_adjstr sub [null,adjstr].
  adjstr sub []
     intro [qtfd:bool,dmstrtd:bool,rltvzd:bool,rltclsd:bool,qntfcd:bool,
                               qltfd:bool,non_ref:bool].

null_mod sub [null,mod].
  mod sub []
     intro [modsyn:synsem,modadj:null_adjstr].

tense sub [base,future,contin,pres,past,rep_past].
  base sub [].
  future sub [].
  contin sub [].
  pres sub [].
  past sub [].
  rep_past sub [].

aux_tense sub [null,hikaye,rivayet,condition].
  hikaye sub [].
  rivayet sub [].
  condition sub [].

psoa_arg sub []
    intro [argname: string,arg: arg_type].

arg_type sub [agr,psoa].

qfpsoa sub [property, relation]
    intro [name:string].
  property sub []
        intro [inst:agr].
  relation sub []
        intro [args:list_psoa_arg].

sem_det sub [forall,exists,the].
  forall sub [].
  exists sub [].
  the sub [].
sem_obj sub [nom_obj, psoa, quant].
  nom_obj sub [npro, pron]
          intro [index:agr,
```

```
             restr:set_psoa].
    npro sub [].
    pron sub [ana, ppro].
      ana sub [recp, refl].
        recp sub [].
        refl sub [].
      ppro sub [].
  quant sub []
        intro [det:sem_det,
               restind:npro].
  psoa sub []
      intro [quants:list_quant,nucleus:qfpsoa].



subcat sub [optionalcat,subcat_type].
  optionalcat sub [opt,obl]
    intro [s_arg:subcat_type].
    opt sub [].
    obl sub [].

subcat_type sub [char,synsem,sign].

list_or_set_subcat sub [set_subcat,list_subcat,list_xxx].
subcat_or_ne_set sub [subcat,ne_set_subcat].
subcat_or_ne_list sub [subcat,ne_list_subcat].

list sub [e_list,ne_list,list_cat,string,list_string,list_sign,
          list_quant,list_xxx,list_psoa_arg].
  e_list sub [].
  ne_list sub [ne_list_cat,ne_string,ne_list_string,
                ne_list_xxx,ne_list_sign,ne_list_quant,ne_list_psoa_arg]
     intro [hd:bot,
            tl:list].
  ne_list_xxx sub [ne_list_subcat,ne_list_synsem].
  list_cat sub [e_list,ne_list_cat].
     ne_list_cat sub []
        intro [hd:cat,
               tl:list_cat].
  string sub [e_list,ne_string].
     ne_string sub []
        intro [hd:char,
               tl:string].
  list_xxx sub [list_subcat,list_synsem,ne_list_xxx].
  list_subcat sub [e_list,ne_list_subcat].
     ne_list_subcat sub []
         intro [hd: subcat_or_ne_set,
                tl: list_subcat].
  list_synsem sub [e_list,ne_list_synsem].
     ne_list_synsem sub []
        intro [hd:synsem,
               tl:list_synsem].
  list_string sub [e_list,ne_list_string].
     ne_list_string sub []
        intro [hd:string,
               tl:list_string].
  list_sign sub [e_list,ne_list_sign].
     ne_list_sign sub []
        intro [hd:sign,
               tl:list_sign].
  list_quant sub [e_list,ne_list_quant].
     ne_list_quant sub []
        intro [hd:quant,
               tl:list_quant].
  list_psoa_arg sub [e_list,ne_list_psoa_arg].
```

```
    ne_list_psoa_arg sub []
       intro [hd:psoa_arg,
              tl:list_psoa_arg].


set sub [e_list,ne_set,set_char,set_subcat,set_psoa,set_quant].
  ne_set sub [ne_set_char,ne_set_subcat,ne_set_psoa,ne_set_quant]
     intro [el:bot,
            els:set].
  set_char sub [e_list,ne_set_char].
     ne_set_char sub []
        intro [el:char,
               els:set_char].
  set_subcat sub [e_list,ne_set_subcat].
     ne_set_subcat sub []
         intro [el: subcat_or_ne_list,
                els: set_subcat].
  set_psoa sub [e_list,ne_set_psoa].
     ne_set_psoa sub []
         intro [el: psoa,
                els: set_psoa].
  set_quant sub [e_list,ne_set_quant].
     ne_set_quant sub []
         intro [el:quant,
                els: set_quant].



char sub [a,b,c,ç,d,e,f,g,ğ,h,ı,i,j,k,l,m,n,o,ö,p,q,r,s,ş,t,u,ü,v,w,x,y,z,].
   a sub [].
   ........
   .......
```

A.2 Phrase Structure Rules

```
%==============================================================
%
%  @(#)T.rule               Rev:1.6                1/7/96
%
%==============================================================
% Grammar Rules
%==============================================================

subcat_retr1 rule
(Mother,phrase,phon:PhonMot,
               synsem:local:cat:(subcat:SubMot,adjuncts:Adjs,subj:Subj),
               dtrs:DtrsMot)
===>
cat> (Arg,phon:PhonArg,synsem:SynArg),
cat> (Head,phon:PhonHead,synsem:local:cat:(subcat:SubHead,adjuncts:Adjs,
                                           subj:Subj)),
goal> (append(PhonArg,PhonHead,PhonMot),
       head_feature_principle(Mother,Head),
       sselectlast(SynArg,SubHead,SubMot),
       combine_semantics(Head,Arg,Mother),
       append_comp(DtrsMot,Head,Arg),
       nonlocal_principle(Arg,Head,Mother)).

subcat_retr2 rule
(Mother,phrase,phon:PhonMot,
```

```
              synsem:local:cat:(subcat:SubMot,adjuncts:Adjs,subj:Subj),
              dtrs:DtrsMot)
===>
cat> (Head,phon:PhonHead,synsem:local:cat:(head:verb,
                                        subcat:SubHead,adjuncts:Adjs,
                                        subj:Subj)),
cat> (Arg,phon:PhonArg,synsem:SynArg),
goal> (append(PhonArg,PhonHead,PhonMot),
       head_feature_principle(Mother,Head),
       sselectlast(SynArg,SubHead,SubMot),
       combine_semantics(Head,Arg,Mother),
       append_comp(DtrsMot,Head,Arg),
       nonlocal_principle(Arg,Head,Mother)).

adj_head rule
(Mother,phrase,phon:PhonMot,dtrs:DtrsMot)
===>
cat> (Adjunct,phon:PhonAdj),
cat> (Head,phon:PhonHead),
goal> (append(PhonAdj,PhonHead,PhonMot),
       combine_semantics(Head,Adjunct,Mother),
       head_feature_principle(Mother,Head),
       adjunct_principle(Mother,Adjunct,Head),
       append_spec(DtrsMot,Head,Adjunct),
       nonlocal_principle(Adjunct,Head,Mother)).

slash rule
(Mother,phrase,phon:PhonMot,
              synsem:(local:cat:(subcat:SubMot,subj:Subj),
                      nonlocal:(inherited:slash:Local,
                                tobind:slash:HT)),
              dtrs:DtrsMot)
===>
cat>(Head,phon:PhonHead,synsem:(local:cat:(head:(noun;infinitival),
                                           subcat:SubHead,
                                           subj:Subj),
                                nonlocal:(inherited:slash:null,
                                          tobind:slash:HT))),
goal> (append((PhonSl,[e_list]),PhonHead,PhonMot),
       head_feature_principle(Mother,Head),
       nonlocal_principle(synsem:Slsynsem,Head,Mother),
       sselectlast((Slsynsem,local:(Local,
                                cat:head:(agr:per:third,n_ind:SlInd),
                                cont:index:SlInd),
                   nonlocal:(inherited:slash:Local,
                             tobind:slash:null)),SubHead,(SubMot,e_list)),
       append_comp(DtrsMot,Head,(Slash,phon:PhonSl,qretr:e_list,qstore:e_list,
                                 synsem:Slsynsem)),
       combine_semantics(Head,synsem:local:cont:(index:SlInd,restr:e_list)
                                                              ,Mother)).
```

A.3 Constraints and Macros

```
%===============================================================
%
%  @(#)T.cons              Rev:1.8              1/7/96
%
%===============================================================
```

```
%      Constraints
%==============================================================


determiner cons (gradable: minus).
article    cons (countable: plus).
quantitative_adj  cons (gradable: minus,countable: plus).
word cons (qretr:e_list,synsem:nonlocal:inherited:slash:null).
subj_rel cons (tense:base,vagr:null).
%obj_rel cons (tense:(future;past)).
mak cons (tense:base).
ış cons (tense:base,vagr:agr).
complemented cons (tense:(future;past),vagr:agr).
noun_l cons (synsem:local:cat:head:(n_ind:I,[agr] == [n_ind])).
common_l cons (synsem:local:(cat:(head:n_ind:I,
                                  adjuncts:(qtfd:minus,
                                     dmstrtd:minus,
                                     rltvzd:minus,
                                     rltclsd:minus,
                                     qntfcd:minus,
                                     qltfd:minus,
                                     non_ref:plus)),
                              cont:index:I)).

pronoun_l cons (synsem:local:(cat:(head:n_ind:I,
                                  adjuncts:(
                                     qtfd:minus,
                                     dmstrtd:minus,
                                     rltvzd:minus,
                                     rltclsd:minus,
                                     qntfcd:minus,
                                     qltfd:minus,
                                     non_ref:minus)),
                              cont:index:I)).


quantif_l cons
  (synsem:local:cat:head:(quantifier,
                            mod:(modsyn:(local:cat:(head:(common),
                                                   adjuncts:(qtfd:minus,
                                                     dmstrtd:minus,
                                                     rltvzd:minus,
                                                rltclsd:A,
                                                qntfcd:B,
                                                qltfd:C))),
                                modadj:(qtfd:plus,
                                     dmstrtd:minus,
                                     rltvzd:minus,
                                     rltclsd:A,
                                     qntfcd:B,
                                     qltfd:C,non_ref:minus)))).


demonstra_l cons
  (synsem:local:cat:head:(demonstrative_adj,
                            mod:(modsyn:(local:cat:(head:(common),
                                                   adjuncts:(qtfd:minus,
                                                     dmstrtd:minus,
                                                     rltvzd:minus,
                                                rltclsd:A,
                                                qntfcd:B,
                                                qltfd:C))),
                                modadj:(qtfd:minus,
                                     dmstrtd:plus,
                                     rltvzd:minus,
```

```
                                    rltclsd:A,
                                    qntfcd:B,
                                    qltfd:C,
                                    non_ref:minus)))).

qualitative_l cons
  (synsem:local:cat:head:(qualitative_adj,
                           mod:(modsyn:(local:cat:(head:(common),
                                                   adjuncts:(qtfd:A,
                                                     dmstrtd:minus,
                                                     rltvzd:minus,
                                                rltclsd:minus,
                                                qntfcd:B,
                                                non_ref:C))),
                               modadj:(qtfd:A,
                                    dmstrtd:minus,
                                    rltvzd:minus,
                                    rltclsd:minus,
                                    qntfcd:B,
                                    qltfd:plus,
                                    non_ref:C)))).

relativized_l cons
  (synsem:local:cat:head:( mod:(modsyn:(local:cat:(head:(common),
                                                   adjuncts:(qtfd:A,
                                                     dmstrtd:B,
                                                rltclsd:minus,
                                                qntfcd:D,
                                                qltfd:E)
                                       )),
                               modadj:(qtfd:A,
                                    dmstrtd:B,
                                    rltvzd:plus,
                                    rltclsd:minus,
                                    qntfcd:D,
                                    qltfd:E,
                                    non_ref:minus)))).

subj_rel_l cons
  (synsem:(local:(cat:(head:(subj_rel,
                               mod:(modsyn:(local:(cat:(head:(common,
                                                              n_ind:NInd),
                                                   adjuncts:(
                                                qtfd:A,
                                                     dmstrtd:B,
                                                rltvzd:minus,
                                                qntfcd:D,
                                                qltfd:E)
                                                ),
                                                cont:(Cont,index:Ind))),
                               modadj:(qtfd:A,
                                    dmstrtd:B,
                                    rltvzd:minus,
                                    rltclsd:plus,
                                    qntfcd:D,
                                    qltfd:E,
                                    non_ref:minus)))),
                 cont:_),%index:Ind),
          nonlocal:tobind:slash:(cat:head:(common,n_ind:NInd
                                                )%,
                                     % cont:Cont
                                      ))).
obj_rel_l cons
  (synsem:(local:(cat:(head:(obj_rel,
                            mod:(modsyn:(local:(cat:(head:(common,
```

```
                                                            n_ind:NInd),
                                                       adjuncts:(
                                                    qtfd:A,
                                                         dmstrtd:B,
                                                    rltvzd:minus,
                                                    qntfcd:D,
                                                    qltfd:E)),
                                                    cont:Cont)),
                                modadj:(qtfd:A,
                                     dmstrtd:B,
                                     rltvzd:minus,
                                     rltclsd:plus,
                                     qntfcd:D,
                                     qltfd:E,
                                     non_ref:minus)))),
                   cont:index:NInd),
           nonlocal:tobind:slash:(cat:head:(common,
                                                    n_ind:NInd),
                                  cont:Cont))).

finite_l cons
  (synsem:(local:(cat:head:finite),
           nonlocal:tobind:slash:null)).

%================================================================
%
%  @(#)T.macro               Rev:1.5                1/7/96
%
%================================================================
%      Macros
%================================================================

common_noun macro
     (common_l,
      synsem:(local:(cat:(head:(common,
                              case:nom,
                              agr:(num:sing,
                                   per:third),
                              mod:null,
                              n_ind:NInd,
                              pred:minus,
                              rel:minus,
                              poss:none),
                         subcat:e_list,
                         subj:null),
                     cont:(Cont,index:NInd)),
              nonlocal:(inherited:slash:null,
                        tobind:slash:null)
             ),
       qstore:e_list
      ).

opt(X) macro
   (opt,s_arg:X).
obl(X) macro
   (obl,s_arg:X).

np(Head,Ind) macro
   (local:(cat:(head:(Head,noun,mod:null,rel:minus,pred:minus),
                subcat:e_list),
           cont:index:Ind),
    nonlocal:( tobind:slash:null)
   ).

vp(Head,Cont) macro
```

```
     (local:(cat:(head:(Head,mod:null),
                  subcat:e_list),
             cont:Cont),
      nonlocal:(tobind:slash:null)
     ).

slashinh(X) macro
 (synsem:nonlocal:inherited:slash:X).
slashtob(X) macro
 (synsem:nonlocal:tobind:slash:X).

f_phrase macro
  (phrase,
  synsem:local:cat:subcat:e_list,synsem:nonlocal:inherited:slash:null,
      synsem:nonlocal:tobind:slash:null).

f_sent macro
  (@f_phrase,synsem:local:cat:head:(finite;pred:plus)).
```

A.4 Definite Clauses

```
%==============================================================
%
%  @(#)T.clause                Rev:1.9                1/7/96
%
%==============================================================
% Principles & Clauses
%==============================================================


%--------
% Head Feature Principle:
%       Head Feature of the mother is token identical to head feature of
%       the Head.
head_feature_principle(synsem:local:cat:head:X,synsem:local:cat:head:X) if
        true.

%--------
% Combine Semantics: ( Head,Dtr,Mother)
%
combine_semantics( synsem:local:cont:(index:HInd,restr:Hrest),
                    synsem:local:cont:(index:DInd,restr:Drest),
                    synsem:local:cont:(index:HInd,restr:MRest)) if
        appendset(Hrest,Drest,MRest).

combine_semantics( synsem:local:cont:(index:HInd,restr:Hrest),
                    synsem:local:cont:(Drest,psoa),
                    synsem:local:cont:(index:HInd,restr:MRest)) if
        appendset(Hrest,(el:Drest,els:[]),MRest).

combine_semantics( synsem:local:cont:(nucleus:HNuc,quants:HQ),
                   synsem:local:cont:(DCont),
                   synsem:local:cont:(nucleus:HNuc,
                        quants:([(det:the,restind:DCont) |HQ]     ))) if
                true.

combine_semantics( synsem:local:cont:(nucleus:HNuc,quants:HQ),
                   synsem:local:cont:(psoa),
                   synsem:local:cont:(nucleus:HNuc,quants:HQ)) if true.
```

```
%---------
%  adjunct_principle(mother,adjunct,head)
%
adjunct_principle((synsem:local:cat:(subj:Subj,subcat:Subcat,adjuncts:MAdjs)),
                  (synsem:local:cat:(head:mod:(modsyn:(Mod),
                                               modadj:MAdjs),
                                     subcat:[])),
                  synsem:(Mod,local:cat:(CatH,subj:Subj,subcat:Subcat))) if
        checkposs(Mod),
        checksubst(CatH).

checkposs(local:cat:head: =\= noun) if true,!.
checkposs(local:cat:head:poss:none) if true.
checkposs(local:cat:(head:(poss:poss,pred:minus),subcat:ne_set)) if true.
checkposs(local:cat:(head:(poss:poss,pred:minus),subcat:ne_list)) if true.
checkposs(local:cat:(head:(poss:poss,pred:plus),subcat:tl:ne_set)) if true.
checkposs(local:cat:(head:(poss:poss,pred:plus),subcat:tl:ne_list)) if true.

checksubst(head: =\= noun) if true,!.
checksubst(head:(pred:minus)) if true.
checksubst((head:(pred:plus),subcat:ne_set)) if true.
checksubst((head:(pred:plus),subcat:ne_list)) if true.



%---------
%  nonlocal_principle
%
nonlocal_principle((@slashtob((local,LAdj)),@slashinh((LAdj))),
                   (@slashtob((HT,null)),@slashinh(null)),
                   (@slashtob(HT),@slashinh(null))) if true.

nonlocal_principle((@slashinh(null),@slashtob(null)),
                   (@slashinh(HI),@slashtob(HT)),
                   (@slashinh(HI),@slashtob(HT))) if true,!.

nonlocal_principle((@slashinh(null),@slashtob(null)),
                   (@slashtob((HT)),@slashinh(null)),
                   (@slashinh(null),@slashtob(HT))) if true.

nonlocal_principle((@slashinh(AI),@slashtob(null)),
                   (@slashtob((null)),@slashinh(null)),
                   (@slashinh(AI),@slashtob(HT))) if true.

nonlocal_principle((@slashinh(HT),Arg),
                   (@slashtob((HT,local)),Head),
                   (Mother,@slashtob(HT),@slashinh(HT))) if
                                check_rel(Arg,Mother).

check_rel((@slashinh(S)),
          (synsem:(local:cat:(head:subj_rel,
                              subj:nonlocal:inherited:slash:S),
                   nonlocal:tobind:slash:S))) if true.

check_rel((@slashinh(S), synsem:local: =\= S),
          (synsem:(local:(cat:(head:subj_rel,n_inc:plus,
                               subj:local:cat:adjuncts:non_ref:plus)),
                   nonlocal:tobind:slash:S))) if true.

check_rel((@slashinh(S)),
          (synsem:(local:cat:(head:obj_rel,
                              subj:nonlocal:inherited:slash: =\= S),
                   nonlocal:tobind:slash:S))) if true.
```

```
%=============================================================
%   Clauses
%=============================================================

append([],Xs,Xs) if
  true.
append([H|T1],L2,[H|T2]) if
  append(T1,L2,T2).

appends(e_list,X,X) if true,!.
appends([A],(X,ne_set),[A|[X]]) if true,!.
appends(X,[],X) if true,!.
appends((X,set),(L,list),[X|L]) if true.
appends([X|Rx],Y,[X|Res]) if
                appends(Rx,Y,Res).
appends((el:X,els:Rx),(Y,set),(el:X,els:Res)) if
                appends(Rx,Y,Res).

listlast(X,(hd:X,tl:e_list),e_list) if true,!.
listlast(X,[H|T],[H|R]) if
        listlast(X,T,R).

permut(e_list,e_list) if true , !.
permut((X,set),(X,set)) if true.
permut((el:X,els:R),(el:Y,els:(el:X,els:R2))) if
                permut(R,(el:Y,els:R2)).

selectlast(Arg,(ne_list_synsem,Sursub),Reslist) if
        listlast(Arg,Sursub,Reslist),!.

selectlast(Arg,Sub,Reslist) if
        removeop(Sub,SubRem),
        surface(SubRem,Sursub),
        listlast(Arg,Sursub,Reslist).

selectfirst(Arg,(ne_list_synsem,[Arg|Rest]),Rest) if !,true.

selectfirst(Arg,Sub,Reslist) if
        removeop(Sub,SubRem),
        surface(SubRem,[Arg|Reslist]).

select(Arg,(s_arg:Arg),[]) if true.
select(Arg,(Arg,subcat_type),[]) if true.
select(T,(X,set),Z) if
                permut(X,(el:Ct,els:Cr)),
                select(T,Ct,Res),
                appends(Res,Cr,Z).
select(T,[Xt|R],Z) if
                select(T,Xt,Res),
                appends(Res,R,Z).


appendset(e_list,(set,X),X) if true.
appendset((el:El,els:Rels),(set,S2),(el:El,els:Res)) if
        appendset(Rels,S2,Res).

removeop([],[]) if true.
removeop(s_arg:X,@obl(X)) if true.
removeop([(opt)|R],R2) if removeop(R,R2).
removeop([H|R],[H2|R2]) if removeop(H,H2),removeop(R,R2).
removeop((el:(opt),els:R),R2) if removeop(R,R2).
removeop((el:H,els:R),(el:H2,els:R2)) if removeop(H,H2),removeop(R,R2).
```

```
surface(e_list,e_list) if true.
surface(X,[T|Rs]) if
                select(T,X,R),surface(R,Rs).

append_comp((hd_dtr:Head,comp_dtrs:[],spec_dtrs:[],subj_dtr:Comp),
            (Head,word,(synsem:local:cat:subj:Subj)),
            (Comp,synsem:Subj)) if true,!.
append_comp((subj_dtr:Comp,comp_dtrs:Cdtrs,spec_dtrs:Sdtrs,hd_dtr:Hdtr),
            (synsem:local:cat:subj:Subj,
               dtrs:(comp_dtrs:Cdtrs,spec_dtrs:Sdtrs,hd_dtr:Hdtr)),
            (Comp,synsem:Subj)) if true,!.
append_comp((hd_dtr:Head,comp_dtrs:[Comp],spec_dtrs:[]),
                (Head,word), Comp) if true,!.
append_comp((hd_dtr:Hdtr,comp_dtrs:ResComp,spec_dtrs:Sdtrs),
            (dtrs:(comp_dtrs:Comp1,hd_dtr:Hdtr,spec_dtrs:Sdtrs)),
            Comp) if
                append([Comp],Comp1,ResComp).

append_spec((hd_dtr:Head,comp_dtrs:[],spec_dtrs:[Adjunct]),
                (Head,word), Adjunct) if true,!.
append_spec((hd_dtr:Hdtr,comp_dtrs:Cdtrs,subj_dtr:SUdtr,spec_dtrs:ResSpec),
            dtrs:(hd_dtr:Hdtr,comp_dtrs:Cdtrs,subj_dtr:SUdtr,spec_dtrs:Spec1),
            Spec) if
                append([Spec],Spec1,ResSpec).



%====================================================================
% Clauses applying the constraints of the Lexical Rules....
%====================================================================

apply_case(
(word,
 synsem:(local:(cat:(head:(common,
                           case:nom,
                           agr:(num:Num,per:third),
                           mod:Mod,
                           rel:(Rel,minus),
                           pred:(Pred,minus),
                           n_ind:NInd,
                           poss:Poss),
                     subcat:Subcat,
                     adjuncts:Adjuncts,
                     subj:Subj),
                cont:Cont,
                conx:Conx),
          nonlocal:Nonlocal),
    qstore:Qs) ,
(word,
 synsem:(local:(cat:(head:(common,
                           case:Case,
                           agr:(num:Num,per:third),
                           mod:Mod,
                           rel:Rel,
                           pred:Pred,
                           n_ind:NInd,
                           poss:Poss),
                     subj:Subj,
                     adjuncts:Adjuncts,
                     subcat:Subcat),
                cont:Cont,
                conx:Conx),
```

```
          nonlocal:Nonlocal),
    qstore:Qs) , Case , CMod ) if check_case_mod(Poss,CMod).

check_case_mod(none,a) if true.
check_case_mod(by:per:first,a) if true.
check_case_mod(by:per:second,a) if true.
check_case_mod(by:per:third,b) if true.


apply_poss(
(word,
 synsem:(local:(cat:(head:(common,
                           case:nom,
                           agr:(Agr,per:third),
                           mod:Mod,
                           rel:(Rel,minus),
                           pred:(Pred,minus),
                           poss:none),
                     subcat:Subcat),
                cont:(index:Ind,restr:(el:EL)),
                conx:Conx),
          nonlocal:Nonlocal),
  qstore:Qs) ,
(word,
 synsem:(local:(cat:(head:(common,
                           case:nom,
                           agr:Agr,
                           mod:Mod,
                           rel:Rel,
                           pred:Pred,
                           n_ind:Ind,
                           poss:by:By),
                     subj:Subj,
                     subcat:Subcat2),
                cont:(index:Ind,restr:(el:EL,els:(el:(nucleus:(relation,
                                                     name:[p,o,s,s,e,s,s],
                                                     args:
                                        [(argname:[o,w,n,e,r],arg:Ind2),
                                         (argname:[o,w,n,e,d],arg:Ind)]),
                                         quants:e_list),els:e_list))),
                conx:Conx),
          nonlocal:Nonlocal),
  qstore:Qs) , By ) if
           appends([@obl((Subj,@np((case:gen,agr:By),Ind2)))],Subcat,Subcat2).

%-------------------------------------------------------------------

apply_copula(
(word,
 synsem:(local:(cat:(head:(common,
                           case:nom,
                           rel:(Rel,minus),
                           pred:minus,
                           poss:Poss),
                     subj:Subj,
                     subcat:Subcat),
                cont:(C1,restr:Restr),
                conx:Conx),
          nonlocal:Nonlocal),
  qstore:Qs) ,
(noun_l,
 synsem:(local:(cat:(head:(common,
                           case:nom,
                           agr:(Agr),
                           mod:null,
```

```
                        rel:Rel,
                        pred:plus,
                        n_ind:Ind,
                        poss:Poss),
                  subj:Subj2,
                  subcat:Subcat2),
              cont:C2,%(index:Ind,restr:Restr),
              conx:Conx),
        nonlocal:Nonlocal),
  qstore:Qs), Agr) if
              contentcop(C1,C2,Ind),
              appends([@obl((Subj2,@np((case:nom,agr:Agr),_)))],Subcat,Subcat2).

%-----------------------------------------------------------------------

apply_adj2noun(
(word,
 synsem:(local:(cat:(head:((qualitative_adj;rel:plus),
                        mod:(modsyn:Mod,modadj:Modadj)),
                  subj:Subj,
                  subcat:Subcat),
              cont:(index:Ind,Cont),
              conx:Conx),
         nonlocal:Nonlocal),
  qstore:Qs),
(noun_l,
 synsem:(Mod,local:(cat:(head:(mod:null,
                              case:nom,
                              agr:(per:third,num:sing),
                              rel:minus,
                              pred:minus,
                              n_ind:Ind,
                              poss:none),
                         subj:Subj,
                         subcat:Subcat),
                   cont:Cont,
                   conx:Conx),
             nonlocal:Nonlocal),
   qstore:Qs)) if true.
```

A.5 Lexicon

```
%=============================================================
%
%  @(#)T.lex               Rev:1.10               1/7/96
%
%=============================================================
% Lexicon
%=============================================================


kırmızı --->
    (qualitative_l,phon:[[k,ı,r,m,ı,z,ı]],
     synsem:(local:(cat:(head:(countable:plus,gradable:plus,
                             mod:modsyn:(local:(cat:head:n_ind:NInd,
                                          cont:(index:Ind)))),
                         subcat:[],
                        subj:null),
```

```
                   cont:(index:Ind,
                         restr:(el:(quants:e_list,
                                nucleus:(name:[r,e,d],inst:NInd)),
                                els:e_list)
                        )
                   ),
            nonlocal:tobind:slash:null)
     ).


ben --->
    (word,phon:[[b,e,n]],
     synsem:(local:(cat:(head:(personal_pr,
                         case:nom,
                         rel:minus,
                            agr:(num:sing,
                              per:first),
                         mod:null,
                         poss:none),
                            subcat:e_list,
                       subj:null),
                   cont:(npro,
                         index:(Ind,per:first,num:sing),
                         restr:e_list),
                   conx:conx
                   ),
             nonlocal:tobind:slash:null)
                 ).

kapı --->
     ( @common_noun,
      phon:[[k,a,p,ı]],
      synsem:local:cont:(npro,
                         index:(agr,Ind,per:third,num:sing),
                         restr:(el:(nucleus:(name:[d,o,o,r],inst:Ind),
                                 quants:[]),els:[]))
     ).


ev --->
    (@common_noun,
     phon:[[e,v]],
     synsem:local:cont:(npro,
                         index:(agr,Ind,per:third,num:sing),
                         restr:(el:(nucleus:(name:[h,o,u,s,e],inst:Ind),
                                 quants:[]),els:[])
                       )
     ).



gitti --->
    (finite_l,
     phon:[[g,i,t,t,i]],
     synsem:local:(cat:(head:(finite,mod:null,
           tense:past,
           vagr:(Agr,(per:third,num:sing))),
      subcat:  @obl((Subj,@np((agr:Agr,case:nom),SInd))),
                            @opt(@np(case:abl,FInd)),
                            @opt(@np(case:dat,TInd)),
      subj:Subj
           ),
      cont:(quants:[],nucleus:(name:[g,o],args:[(argname:[g,o,e,r],arg:SInd),
                                                (argname:[t,o],arg:TInd),
                                                (argname:[f,r,o,m],arg:FInd)])
```

```
          ))).


giden --->
    (subj_rel_l,
     phon:[[g,i,d,e,n]],
     synsem:(local:(cat:(head:(subj_rel,
           vcase:nom),
     subcat:  @obl((Subj,@np(case:nom,SInd))),
                          @opt(@np(case:abl,FInd)),
                          @opt(@np(case:dat,TInd)),
      subj:Subj
            ),
      cont:restr:(el:(quants:[],nucleus:(name:[g,o],
                                    args:[(argname:[g,o,e,r],arg:SInd),
                                              (argname:[t,o],arg:TInd),
                                              (argname:[f,r,o,m],arg:FInd)])
           ),els:[]))
      )).

geldiği --->
    (obj_rel_l,
     phon:[[g,e,l,d,i,ğ,i]],
     synsem:(local:(cat:(head:(obj_rel,
           tense:past,
           vagr:(Agr,(per:third,num:sing))),
     subcat:  @obl((Subj,@np((agr:Agr,case:gen),SInd))),
                               @opt(@np(case:abl,FInd)),
                               @opt(@np(case:dat,TInd)),
      subj:Subj
            ),
      cont:restr:(el:(quants:[],nucleus:(name:[g,o],
                                    args:[(argname:[c,o,m,e,r],arg:SInd),
                                              (argname:[t,o],arg:TInd),
                                              (argname:[f,r,o,m],arg:FInd)])
        ),els:[]))
    )).

söylüyor --->
    (finite_l,
     phon:[[s,ö,y,l,ü,y,o,r]],
     synsem:(local:(cat:(head:(finite,
           tense:past,
           mod:null,
           vagr:(Agr,(per:third,num:sing))),
     subcat:  @obl((Subj,@np((agr:Agr,case:nom),SInd))),
                               @obl(@vp((vcase:obj,complemented),Spsoa)),
                               @opt(@np(case:dat,TInd)),
      subj:Subj
            ),
      cont:(quants:[],nucleus:(name:[t,e,l,l],
                                    args:[(argname:[t,e,l,l,e,r],arg:SInd),
                                              (argname:[t,o],arg:TInd),
                                            (argname:[w,h,a,t],arg:Spsoa)])
        ))
    )).

empty
    (word,phon:[[p,r,o]],
     synsem:(local:(cat:(head:(pronoun,
                              case:(nom;gen),
                                 agr:Agr,
                             mod:null,
                             n_ind:Agr,
                             pred:minus,
```

```
                        rel:minus,
                        poss:none),
                  subcat:e_list,
                  subj:null),
             cont:(npro,
                   index:(Agr),
                   restr:[])),
       nonlocal:(inherited:slash:null,
                 tobind:slash:null)
           )).
```

A.6 Lexical Rules

```
%=================================================================
%
%  @(#)T.lex_rule               Rev:1.9               1/7/96
%
%=================================================================
% Lexical Rules
%--
%  Morphological clauses used in lexical rules...
%---

back(a).
back(ı).
back(o).
back(u).
kalin_hece([X]) :- back(X),!.
kalin_hece([X,_]) :- back(X).

front(e).
front(i).
front(ö).
front(ü).
ince_hece([X]) :- front(X),!.
ince_hece([X,_]) :- front(X).
wovel(X) :- front(X),!.
wovel(X) :- back(X).

backrounded(o).
backrounded(u).
b_r_hece([X]) :- backrounded(X),!.
b_r_hece([X,_]) :- backrounded(X).

frontrounded(ö).
frontrounded(ü).
f_r_hece([X]) :- frontrounded(X),!.
f_r_hece([X,_]) :- frontrounded(X).

backunrounded(a).
backunrounded(ı).
b_u_hece([X]) :- backunrounded(X),!.
b_u_hece([X,_]) :- backunrounded(X).

frontunrounded(e).
frontunrounded(i).
f_u_hece([X]) :- frontunrounded(X),!.
f_u_hece([X,_]) :- frontunrounded(X).

yumusa(p,b).
```

```
yumusa(ç,c).
yumusa(t,d).
yumusa(k,ğ).

kal_yum([X,Y],Yum) :- back(X),yumusa(Y,Yum).
ince_yum([X,Y],Yum) :- front(X),yumusa(Y,Yum).

f_u_yum([X,Y],Yum) :- frontunrounded(X),yumusa(Y,Yum).
b_u_yum([X,Y],Yum) :- backunrounded(X),yumusa(Y,Yum).
f_r_yum([X,Y],Yum) :- frontrounded(X),yumusa(Y,Yum).
b_r_yum([X,Y],Yum) :- backrounded(X),yumusa(Y,Yum).


%==============================================================
% Lexical rules...
%==============================================================
:-lex_rule_depth(4).

plural lex_rule
  Cat1 **> Cat2
if apply_plural((Cat1,phon:[Phon]),(Cat2,phon:[Phon2])),
   append(Phon,[,p,l,u],Phon2)
morphs
  (X,L2) becomes (X,L2,lar) when kalin_hece(L2),
  (X,L2) becomes (X,L2,ler) when ince_hece(L2).

%--

accusative_a lex_rule
  Cat1 **> Cat2
if apply_case((Cat1,phon:[Phon]),(Cat2,phon:[Phon2]),obj,a),
   append(Phon,[,o,b,j],Phon2)
morphs
  (X,[L]) becomes (X,[L],yı) when backunrounded(L),
  (X,[L]) becomes (X,[L],yi) when frontunrounded(L),
  (X,[L]) becomes (X,[L],yu) when backrounded(L),
  (X,[L]) becomes (X,[L],yü) when frontrounded(L),
  (X,[L1,L2]) becomes (X,L1,[Y],[ı]) when b_u_yum([L1,L2],Y),
  (X,[L1,L2]) becomes (X,L1,[Y],[i]) when f_u_yum([L1,L2],Y),
  (X,[L1,L2]) becomes (X,L1,[Y],[u]) when b_r_yum([L1,L2],Y),
  (X,[L1,L2]) becomes (X,L1,[Y],[ü]) when f_r_yum([L1,L2],Y),
  (X,L2) becomes (X,L2,[ı]) when b_u_hece(L2),
  (X,L2) becomes (X,L2,i) when f_u_hece(L2),
  (X,L2) becomes (X,L2,u) when b_r_hece(L2),
  (X,L2) becomes (X,L2,[ü]) when f_r_hece(L2).

accusative_b lex_rule
  Cat1 **> Cat2
if apply_case((Cat1,phon:[Phon]),(Cat2,phon:[Phon2]),obj,b),
   append(Phon,[,o,b,j],Phon2)
morphs
  (X,[L]) becomes (X,[L],nı) when backunrounded(L),
  (X,[L]) becomes (X,[L],ni) when frontunrounded(L),
  (X,[L]) becomes (X,[L],nu) when backrounded(L),
  (X,[L]) becomes (X,[L],nü) when frontrounded(L).


%--

possessive_3_s lex_rule
  Cat1 **> Cat2
if
apply_poss((Cat1,phon:[Phon]),(Cat2,phon:[Phon2]),(num:sing,per:third)),
  append(Phon,[,t,s,g],Phon2)
morphs
```

```
  (X,[L]) becomes (X,[L],sı) when backunrounded(L),
  (X,[L]) becomes (X,[L],si) when frontunrounded(L),
  (X,[L]) becomes (X,[L],su) when backrounded(L),
  (X,[L]) becomes (X,[L],sü) when frontrounded(L),
  (X,[L1,L2]) becomes (X,L1,[Y],[ı]) when b_u_yum([L1,L2],Y),
  (X,[L1,L2]) becomes (X,L1,[Y],[i]) when f_u_yum([L1,L2],Y),
  (X,[L1,L2]) becomes (X,L1,[Y],[u]) when b_r_yum([L1,L2],Y),
  (X,[L1,L2]) becomes (X,L1,[Y],[ü]) when f_r_yum([L1,L2],Y),
  (X,L2) becomes (X,L2,[ı]) when b_u_hece(L2),
  (X,L2) becomes (X,L2,i) when f_u_hece(L2),
  (X,L2) becomes (X,L2,u) when b_r_hece(L2),
  (X,L2) becomes (X,L2,[ü]) when f_r_hece(L2).

%--

obj_rel_to_compl lex_rule
(obj_rel_l,
 phon:[Phon],
 synsem:(local:(cat:(head:(obj_rel,
                           vcase:nom,
                           neg:Neg,
                           vagr:Agr,
                           n_inc:N_Inc,
                           tense:Tense
                           ),
                     subj:Subj,
                     adjuncts:Adj,
                     subcat:Subcat),
                cont:restr:el:Rest,
                conx:Conx))
) **>
(complement_l,
 phon:[Phon],
 synsem:(local:(cat:(head:(complemented,
                           vcase:nom,
                           neg:Neg,
                           mod:null,
                           vagr:Agr,
                           n_inc:N_Inc,
                           tense:Tense
                           ),
                     subj:Subj,
                     adjuncts:Adj,
                     subcat:Subcat),
                cont:Rest,
                conx:Conx),
         nonlocal:tobind:slash:null)
)
morphs
   X becomes X.

%--

relativizer lex_rule
  Cat1 **> Cat2
if apply_reltvzr((Cat1,phon:[Phon]),(Cat2,phon:[Phon2])),
   append(Phon,[,r,l,t],Phon2)
morphs
   (X) becomes (X,ki).

%--

copula1_s lex_rule
  Cat1 **> Cat2
if apply_copula((Cat1,phon:[Phon]),(Cat2,phon:[Phon2]),(num:sing,per:first)),
```

```
   append(Phon,[,c,o,p],Phon2)
morphs
  (X,[L2]) becomes (X,[L2],yım) when backunrounded(L2),
  (X,[L2]) becomes (X,[L2],yim) when frontunrounded(L2),
  (X,[L2]) becomes (X,[L2],yum) when backrounded(L2),
  (X,[L2]) becomes (X,[L2],yüm) when frontrounded(L2),
  (X,[L1,L2]) becomes (X,L1,[Y],[ı,m]) when b_u_yum([L1,L2],Y),
  (X,[L1,L2]) becomes (X,L1,[Y],[i,m]) when f_u_yum([L1,L2],Y),
  (X,[L1,L2]) becomes (X,L1,[Y],um) when b_r_yum([L1,L2],Y),
  (X,[L1,L2]) becomes (X,L1,[Y],[ü,m]) when f_r_yum([L1,L2],Y),
  (X,L2) becomes (X,L2,[ı],m) when b_u_hece(L2),
  (X,L2) becomes (X,L2,im) when f_u_hece(L2),
  (X,L2) becomes (X,L2,um) when b_r_hece(L2),
  (X,L2) becomes (X,L2,[ü],m) when f_r_hece(L2).

%--

copula3_s lex_rule
  Cat1 **> Cat2
if apply_copula((Cat1,phon:[Phon]),(Cat2,phon:[Phon2]),(num:sing,per:third)),
   append(Phon,[,c,o,p],Phon2)
morphs
  (X,[L1,L2]) becomes (X,L1,[L2],tır) when b_u_yum([L1,L2],Y),
  (X,[L1,L2]) becomes (X,L1,[L2],tir) when f_u_yum([L1,L2],Y),
  (X,[L1,L2]) becomes (X,L1,[L2],tur) when b_r_yum([L1,L2],Y),
  (X,[L1,L2]) becomes (X,L1,[L2],tür) when f_r_yum([L1,L2],Y),
  (X,L2) becomes (X,L2,dır) when b_u_hece(L2),
  (X,L2) becomes (X,L2,dir) when f_u_hece(L2),
  (X,L2) becomes (X,L2,dur) when b_r_hece(L2),
  (X,L2) becomes (X,L2,dür) when f_r_hece(L2).





adj_promotion lex_rule
  Cat1 **> Cat2
if apply_adj2noun((Cat1,phon:[Phon]),(Cat2,phon:[Phon2])),
   append(Phon,[,a],Phon2)
morphs
   X becomes X.


%---

non_ref_object lex_rule
(verb_l,
 phon:[PhonV],
 synsem:(local:(cat:(head:HeadV,
                     subj:SubjV,
                     adjuncts:AdjV,
                     subcat:Subcat1V),
                cont:ContV,
                conx:ConxV),
         nonlocal:NonlocalV)
) **>
(verb_l,
 phon:[PhonV],
 synsem:(local:(cat:(head:HeadV,
                     subj:SubjV,
                     adjuncts:AdjV,
                     subcat:[ SubcatRV,
                             @obl((local:(cat:(head:(common,
                                                     case:nom,
```

```
                                                      agr:AgrN,
                                                      mod:ModN,
                                                      rel:(RelN),
                                                      pred:(PredN),
                                                      n_ind:NIndN,
                                                      poss:PossN),
                                                 subcat:SubcatN,
                                                 adjuncts:(AdjunctsN,
                                                           non_ref:plus),
                                                 subj:SubjN),
                                           cont:ContN,
                                           conx:ConxN),
                                    nonlocal:NonlocalN))]
                     ),
                cont:ContV,
                conx:ConxV),
         nonlocal:NonlocalV)
)
if selectlast(
 (local:(cat:(head:(common,
                     case:obj,
                     agr:AgrN,
                     mod:ModN,
                     rel:(RelN),
                     pred:(PredN),
                     n_ind:NIndN,
                     poss:PossN),
                subcat:SubcatN,
                adjuncts:AdjunctsN,
                subj:SubjN),
          cont:ContN,
          conx:ConxN),
  nonlocal:NonlocalN), Subcat1V, SubcatRV)
morphs
   X becomes X.
```