TAGGING AND MORPHOLOGICAL DISAMBIGUATION OF TURKISH TEXT

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER

ENGINEERING AND INFORMATION SCIENCE

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BİLKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

İlker Kuruöz

July, 1994

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

Asst. Prof. Kemal Oflazer (Advisor)

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

Asst. Prof. Halil Altay Güvenir

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

Asst. Prof. İlyas Çiçekli

Approved for the Institute of Engineering and Science:

Prof. Mehmet Baray
Director of the Institute

ABSTRACT

TAGGING AND MORPHOLOGICAL DISAMBIGUATION OF TURKISH TEXT

İlker Kuruöz
M.S. in Computer Engineering and Information Science
Advisor: Asst. Prof. Kemal Oflazer
July, 1994

A part-of-speech (POS) tagger is a system that uses various sources information to assign possibly unique POS to words. Automatic text tagging is an important component in higher level analysis of text corpora. Its output can also be used in many natural language processing applications. In languages like Turkish or Finnish, with agglutinative morphology, morphological disambiguation is a very crucial process in tagging as the structures of many lexical forms are morphologically ambiguous. This thesis presents a POS tagger for Turkish text based on a full-scale two-level specification of Turkish morphology. The tagger is augmented with a multi-word and idiomatic construct recognizer, and most importantly morphological disambiguator based on local lexical neighborhood constraints, heuristics and limited amount of statistical information. The tagger also has additional functionality for statistics compilation and fine tuning of the morphological analyzer, such as logging erroneous morphological parses, commonly used roots, etc. Test results indicate that the tagger can tag about 97% to 99% of the texts accurately with very minimal user intervention. Furthermore for sentences morphologically disambiguated with the tagger, an LFG parser developed for Turkish, on the average, generates 50% less ambiguous parses and parses almost 2.5 times faster.

Keywords: Tagging, Morphological Analysis, Corpus Development

ÖZET

TÜRKÇE METİNLERİN İŞARETLENMESİ VE BİÇİMBİRİMSEL ÇOKYAPILILIK ÇÖZÜMLEMESİ

İlker Kuruöz
Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans
Danışman: Yrd. Doç. Dr. Kemal Oflazer
Temmuz, 1994

Sözcük türlerinin işaretlenmesi için kullanılan sistemler metin bilgilerini kullanarak o metinde bulunan her sözcüğü tek bir tür ile işaretlemeye çalışırlar. Otomatik olarak işaretleme, metinlerin üst düzey çözümlemesi açısından önemli bir adımdır ve bu adımın çıktıları pek çok doğal dil işleme uygulamasında kullanılabilir. Türkçe ve Fince gibi çekimli ve bitişken biçimbirimlere sahip dillerde, sözcükler çoğunlukla biçimbirimsel olarak çokyapılı olduğu için biçimbirimsel çokyapılılık çözümlemesi önemli bir işlemdir. Bu tez, Türkçe'nin tam kapsamlı iki aşamalı biçimbirimsel tanımlamasına dayanılarak geliştirilen bir sözcük türü işaretleyicisini sunmaktadır. İşaretleyici aynı zamanda çok kelimeli ve deyimsel yapıları tanımlayabilmekte, daha önemlisi sözcüklerin komşularının biçimbirimsel bilgileri ve bir kısım sezgisel bilgiler (heuristics) kullanarak biçimbirimsel çokyapılılık çözümlemesi yapabilmektedir. İşaretleyici istatistiksel bilgiler toplamak, biçimbirimsel çözümleyicinin bazı hatalarını düzeltmek gibi ek işlevlere de sahiptir. Deney sonuçları, işaretleyicinin metinlerin %97 ila %99'unu çok az kullanıcı yardımı alarak doğru işaretlediğini göstermiş, bir başka deneyde ise biçimbirimsel çokyapılılık çözümlemesi yapılan cümlelerin Türkçe için geliştirilen sözcüksel-işlevsel gramer (LFG) sözdizimsel çözümleyicisi tarafından işlenmesi sonucunda yarıya yakın daha az çözüm yapısı üretildiği ve bu işlemin 2.5 kez daha hızlı gerçekleştiği gözlenmiştir.

Anahtar Sözcükler: İşaretleme, Biçimbirimsel inceleme

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my supervisor Dr. Kemal Oflazer for his guidance, suggestions, and invaluable encouragement throughout the development of this thesis.

I would like to thank Dr. Halil Altay Güvenir and Dr. İlyas Çiçekli for reading and commenting on the thesis.

I am grateful to my family, my wife and my friends for their infinite moral support and help.

Aile'me ve eşim Işıl'a

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Natural Language Processing (NLP) is a research discipline at the juncture of artificial intelligence, linguistics, philosophy, and psychology that aims to *build systems capable of understanding and interpreting the computational mechanisms of natural languages.* Research in natural language processing has been motivated by two main aims:

- to lead to a better understanding of the structure and functions of human language, and
- to support the construction of natural language interfaces and thus to facilitate communication between humans and computers.

The main problem in front of NLP which has kept it from full accomplishment is the sheer size and complexity of human languages. However, once accomplished, NLP will open the door for direct human-computer dialogs, which would bypass normal programming and operating system protocols.

There are mainly four kinds of knowledge used in understanding natural language: morphological, syntactic, semantic and pragmatic knowledge. *Morphology* is concerned with the forms of words. *Syntax* is the description of the ways in which words must be ordered to make structurally acceptable sentences in the language. *Semantics* describe the ways in which words are related to the concepts. It helps us in selecting correct word senses and in eliminating syntactically correct but semantically incorrect analysis. Finally, *pragmatic knowledge* deals with the way we see the world.

Though it seems to be a very difficult task to develop computational systems that process and understand natural language, considerable progress has been achieved.

In this thesis automatic text tagging, which is an important step in discovering the linguistic structure of large text corpora, will be explored. Basic tagging involves annotating the words in a given text with various pieces of information, such as part-of-speech (POS) and other lexical features. POS tagging is an important practical problem with potential applications in many areas including speech synthesis, speech recognition, spelling correction, proofreading, query answering, machine translation and searching large text data bases. It facilitates higher-level analysis essentially by performing a certain amount of ambiguity resolution using relatively cheaper methods. This, however, is not a very trivial task since many words are in general ambiguous in their part-of-speech for various reasons. In English, for example a word such as *table* can be a verb in certain contexts (e.g., He will *table* the motion.) and a noun in others (e.g., The *table* is ready). A program which tags each word in an input sentence with the most likely part-of-speech, would produce the following output for the two example sentences just mentioned:

- He/PPS will/MD table/VB the/AT motion/NN ./.
- The/AT table/NN is/BEZ ready/ADJ ./.

where, PPS = subject pronoun, MD = modal, VB = verb (no inflection), AT = article, NN = noun, BEZ = present 3rd person singular form of "to be" and ADJ = adjective.

In Turkish, there are ambiguities of the sort above. However, the agglutinative nature of the language usually helps resolution of such ambiguities due to restrictions on morphotactics. On the other hand, this very nature introduces another kind of ambiguity, where a lexical form can be morphologically interpreted in many ways. Table 1.1 presents distribution of the number of morphological parses in a sample Turkish text. For example, the word *evin*, can be broken down as:[1]

[1]Output of the morphological analyzer is edited for clarity.

Table 1.1. Some statistics on morphological ambiguity in a sample Turkish text.

No. of	Morphological Parse Distribution					
Words	0	1	2	3	4	≥ 5
7004	3.9%	17.2%	41.5%	15.6%	11.7%	10.1%

Note: Words with zero parses are mostly proper names which are not in the lexicon of the morphological analyzer.

	evin	**Gloss**	**POS**	**English**
1.	ev+in	N(ev)+2SG-POSS	N	your house
2.	ev+[n]in	N(ev)+GEN	N	of the house
3.	evin	N(evin)	N	wheat germ

If, however, the local context is considered it may be possible to resolve the ambiguity as in:

..	senin	**evin**	..	your house
	PN(you)+GEN	N(ev)+2SG-POSS		
..	**evin**	kapısı	..	door of the house
	N(ev)+GEN	N(door)+3SG-POSS		

As a more complex case we can give the following:

	alınmış	**Gloss**	**POS**	**English**
1.	al+ın+[y]mış	ADJ(al)+2SG-POSS	V	it was your
		+NtoV()+NARR+3SG[2]		red one
2.	al+[n]ın+[y]mış	ADJ(al)+GEN+NtoV()	V	it belongs to
		+NARR+3SG		the red one
3.	alın+mış	N(alın)+NtoV()+NARR+3SG	V	it was a forehead
4.	al+ın+mış	V(al)+PASS+VtoAdj(mış)	ADJ	a taken object
5.	al+ın+mış	V(al)+PASS+NARR+3SG	V	it was taken
6.	alın+mış	V(alın)+VtoAdj(mış)	ADJ	an offended person
7.	alın+mış	V(alın)+NARR+3SG	V	s/he was offended

It is in general rather hard to select one of these interpretations without doing substantial analysis of the local context, and even then one can not fully resolve such ambiguities.

[2]In Turkish, all adjectives can be used as nouns, hence with very minor differences adjectives have the same morphotactics as nouns.

In this thesis, a part-of-speech tagger for Turkish text is presented. It is based on a full-scale two-level Turkish morphological analysis, augmented with a multi-word and idiomatic construct recognizer, and most importantly morphological disambiguator, based on local lexical neighborhood constraints and heuristics. Test results indicate that the tagger can tag about 97% to 99% of the texts accurately with very minimal user intervention, i.e., almost only 1% of the text is left ambiguous. Tagging accuracy is very important because on a corpus of about one million words, a tagger with a 98% accuracy leaves 20,000 words wrongly tagged, which then has to be manually tagged.

As mentioned earlier, part-of-speech tagging facilitates higher level analysis, such as syntactic parsing. We tested the impact of morphological disambiguation on the performance of a LFG parser developed for Turkish [6, 7]. The input to the parser was disambiguated using the tool developed and results were compared to the case when the parser had to consider all possible morphological ambiguities. For a set of 80 sentences we observed that morphological disambiguation enables almost a factor of two reduction in the average number of parses generated and over a factor of two speed-up in time.

The outline of the thesis is as follows: Chapter 2 contains an extensive review of previous work and an example of tagging process. In Chapter 3, an overview of morphosyntactic ambiguities in Turkish is presented. In Chapter 4, functionality of the tool with the implementation details are described. Experiments conducted with the tool are described and the results are discussed in Chapter 5. And finally, Chapter 6 contains the conclusions with suggestions for further research.

Chapter 2

Text Tagging

In every computer system that accepts natural language input, it is a must to decide on the grammatical category of each input word. In almost all languages, words are usually ambiguous in their parts-of-speech. They may represent lexical items of different categories, or morphological structures depending on their syntactic and semantic context.

A part-of-speech tagger is a system that uses any available (contextual, statistical, heuristic etc.) information to assign possibly unique parts-of-speech to words in a text. Several methods have been developed to do this task.

2.1 An Example of Tagging Process

We can describe the process of tagging by showing the analysis for the following sentence,

> *İşten döner dönmez evimizin yakınında bulunan derin gölde yüzerek gevşemek en büyük zevkimdi.*
>
> (Relaxing by swimming the deep lake near our house, as soon as I return from work was my greatest pleasure.)

which we assume has been processed by the morphological analyzer with the output[1] given in Figure 2.1.

Although there are a number of choices for tags for the lexical items in the sentence, almost all except one set of choices give rise to ungrammatical or implausible sentence structures.[3] There are a number of points that are of interest here:

- the construct *döner dönmez* formed by two tensed verbs, is actually a temporal adverb meaning *... as soon as .. return(s)* hence these two lexical items can be coalesced into a single lexical item and tagged as a temporal adverb.
- The second person singular possessive (2SG-POSS) interpretation of *yakınında* is not possible since this word forms a simple compound noun phrase with the previous lexical item and the third person singular possessive functions as the compound marker.
- The word *derin* (*deep*) is the modifier of a simple compound noun *derin göl* (*deep lake*) hence the second choice can safely be selected. The verbal root in the third interpretation is very unlikely to be used in text, let alone in second person imperative form. The fourth and the fifth interpretations are not plausible, as adjectives from aorist verbal forms almost never take any further inflectional suffixes. The first interpretation (meaning *your skin*) may be a possible choice but can be discarded in the middle of a longer compound noun phrase.
- The word *en* preceding an adjective indicates a superlative construction and hence the noun reading can be discarded.
- However, there exists a semantic ambiguity for the lexical item *bulunan*. It has two adjectival readings having the meaning *something found* and *existing* respectively. Among this two readings one can not resolve the ambiguity, as long as he/she does not have any idea about the discourse. Contextual information is not sufficient and the ambiguity should be left pending to the higher level analysis.

[1] Upper-case letters in the morphological break-downs represent some specific classes of vowels, e.g., A stands for low-round vowels *e* and *a*, H stands for high vowels *ı,i,u* and *ü*, and D = {*d,t*}.

[2] Although, the final category is adjective the use of possessive (and/or case, number) suffixes indicate nominal usage, as any adjective in Turkish can be used as a noun.

[3] The correct choices of tags are marked with +.

	işten	**Gloss**	**POS**
1.	iş+DAn	N(iş)+ABL	N+
	döner		
1.	döner	N(döner)	N
2.	dön+Ar	V(dön)+AOR+3SG	V+
3.	dön+Ar	V(dön)+VtoAdj(er)	ADJ
	dönmez		
1.	dön+mA+z	V(dön)+NEG+AOR+3SG	V+
2.	dön+mAz	V(dön)+VtoAdj(mez)	ADJ
	evimizin		
1.	ev+HmHz+nHn	N(ev)+1PL–POSS+GEN	N+
	yakınında		
1.	yakın+sH+nDA	ADJ(yakın)+3SG–POSS+LOC	N^2+
2.	yakın+Hn+DA	ADJ(yakın)+2SG–POSS+LOC	N
	bulunan		
1.	bul+Hn+yAn	V(bul)+PASS+VtoADJ(yan)	ADJ
2.	bulun+yAn	V(bulun)+VtoADJ(yan)	ADJ+
	derin		
1.	deri+Hn	N(deri)+2SG–POSS	N
2.	derin	ADJ(derin)	ADJ+
3.	der+yHn	V(der)+IMP+2PL	V
4.	de+Ar+Hn	V(de)+VtoADJ(er)+2SG–POSS	N
5.	de+Ar+nHn	V(de)+VtoADJ(er)+GEN	N
	gölde		
1.	göl+DA	N(göl)+LOC	N+
	yüzerek		
1.	yüz+yArAk	V(yüz)+VtoADV(yerek)	ADV+
	gevşemek		
1.	gevşe+mAk	V(gevşe)+INF	V+
	en		
1.	en	N(en)	N
2.	en	ADV(en)	ADV+
	büyük		
1.	büyük	ADJ(büyük)	ADJ+
	zevkimdi		
1.	zevk+Hm+yDH	N(zevk)+1SG–POSS+NtoV()+PAST+3SG	V+

Figure 2.1. Morphological analyzer output of the example sentence.

	Gloss	**POS**
işten	N(iş)+ABL	N
döner dönmez	ADV(döner dönmez)	ADV
evimizin	N(ev)+1PL–POSS+GEN	N
yakınında	ADJ(yakın)+3SG–POSS+LOC	N
bulunan	V(bul)+PASS+VtoADJ(yan) V(bulun)+VtoADJ(yan)	ADJ ADJ
derin	ADJ(derin)	ADJ
gölde	N(göl)+LOC	N
yüzerek	V(yüz)+VtoADV(yerek)	ADV
gevşemek	V(gevşe)+INF	V
en	ADV(en)	ADV
büyük	ADJ(büyük)	ADJ
zevkimdi	N(zevk)+1SG–POSS+NtoV()+PAST+3SG	V

Figure 2.2. Tagged form of the example sentence.

The tagger should essentially reduce the possible parses to the minimum, employing various constraint rules, heuristics and usage and other statistical information. A sample output for the example sentence would be as given in Figure 2.2.

2.2 Previous Work

There has been two major paradigms for building POS taggers:

- rule-based approaches,
- statistical approaches.

Early approaches to part-of-speech tagging and disambiguation of prose texts were rule-based ones. After 1980's, statistical methods became more popular. But nowadays, researchers from both camps are trying to improve

the accuracy of their approaches to the maximum extent possible. In the following sections an extensive review of work done in both approaches will be presented.

2.2.1 Rule-based Approaches

The earliest rule-based approach is due to Klein and Simmons [10]. They describe a method directed primarily towards the task of initial categorical tagging, rather than disambiguation. Their primary goal was to avoid the labor of constructing a very large dictionary.

Klein and Simmons's algorithm uses a set of 30 POS categories, and claims an accuracy of 90% in tagging. The algorithm first seeks each word in dictionaries of about 400 function words, and of about 1,500 words which are exceptions to the computational rules used. The program then checks for suffixes and special characters as clues. Finally, context frame tests are applied. These work on scopes bounded by unambiguous words. However, Klein and Simmons impose an explicit limit of three ambiguous words in a row. For each such span of ambiguous words, the pair of unambiguous categories bounding it, is mapped into a list. The list includes all known sequences of tags occuring between the particular bounding tags; all such sequences of the correct length become candidates. The program then matches the candidate sequences against the ambiguities remaining from earlier steps of the algorithm. When only one sequence is possible, disambiguation is successful.

The samples used for calibration and testing were limited. First, Klein and Simmons performed hand analysis of a sample of Golden Book Encyclopedia text. Later, when it was run on several pages from that encyclopedia, it correctly and unambiguously tagged slightly over 90% of the words.

Klein and Simmons asserted that "original fears that sequences of four or more unidentified parts-of-speech would occur with great frequency were not substantiated in fact". This readiness, however, is a consequence of following facts. First, the relatively small set of categories reduces ambiguity. Second, a large sample would contain both low frequency ambiguities and many long spans with a higher probability.

Later, Greene and Rubin [5] developed TAGGIT for tagging the Brown

Corpus. They used 86 POS tags. It is reported that this algorithm correctly tagged approximately 77% of the million words in the Brown Corpus (the tagging was then completed by human post-editors). Although this accuracy is substantially lower than that reported by Klein and Simmons, it should be remembered that Greene and Rubin were the first to attempt so large and varied a sample.

TAGGIT divides the task of category assignment into initial (potentially ambiguous) tagging, and disambiguation. Tagging is carried out as follows; first, the program consults an exception dictionary of about 3,000 words. Among other items, this contains all known closed-class words. It then handles various special cases, such as words with initial "$", contractions, special symbols, and capitalized words. A word's ending is then checked against a suffix list of about 450 strings, that was derived from lexiostatistics of the Brown Corpus. If TAGGIT has not assigned some tag(s) after these several steps, the word is tagged as a noun, a verb and an adjective, i.e., being three way ambiguous, in order that the disambiguation routine may have something to work with.

After tagging, TAGGIT applies a set of 3,300 context frame rules. Each rule, when its context is satisfied, has the effect of deleting one or more candidates from the list of possible tags for one word. If the number of candidates is reduced to one, disambiguation is considered successful subject to human post-editing. Each rule can include a scope of up to two unambiguous words on each side of the ambiguous word to which the rule is being applied. This constraint was determined as follows:

In order to create the original inventory of Context Frame Tests, a 900 sentence subset of the Brown University Corpus was tagged, and ambiguities were resolved manually. Then the program was run and it produced and sorted all possible Context Frame Rules which would have been necessary to perform this disambiguation automatically. The rules generated were able to handle up to three consecutive ambiguous words preceded and followed by two non-ambiguous words. However, upon examination of these rules, it was found that a sequence of two or three ambiguities rarely occured more than once in a given context. Consequently, a decision was made to examine only one ambiguity at a time with up to two unambiguously tagged words on either side.

From 1989 to 1992, a group of researchers from the Research Unit for Computational Linguistics at the University of Helsinki participated to an ESPRIT II project to make an operational parser for running English text mainly for information retrieval purposes. Karlsson [8] proposed a parsing framework, known as Constraint Grammar. In this formalism, for each input word morphological and syntactic descriptions are encoded with tags, and all possible readings of them provided as alternatives by a morphological analyzer, called ENGTWOL.

One of the most important steps of Constraint Grammar formalism was context-dependent morphological disambiguation. For this purpose, Voutilainen [15] wrote a grammar for morphological disambiguation, called ENGCG. The task of this grammar is to discard all and only the contextually illegitimate alternative morphological readings. The disambiguator employs an unordered set of linguistic constraints on the linear order of ambiguity-forming morphological readings. This grammar contains 1,100 constraints based on descriptive grammars and studies of various corpora. This rule-based approach has given encouraging results. After the application of disambiguation, of all words, 93-97% becomes unambiguous. There is also an optionally applicable heuristic grammar of 200 constraints that resolves about half of the remaining ambiguities 96-97% reliably, with 96-98% precision.

Among those rule-based part-of-speech taggers, the one built by Brill [1] has the advantage of learning tagging rules automatically. As it will be explored in the next section, research in trainable part-of-speech taggers has also used stochastic methods. While these taggers obtain high accuracy, linguistic information is captured indirectly, typically in tens of thousands of lexical and contextual probabilities. In 1992, Brill applied transformation-based error-driven learning to part-of-speech tagging, and obtained performance comparable to that of stochastic taggers. In this work, the tagger is trained with the following process: First, text is tagged with an initial annotator, where each word is assigned with the most likely tag. Once text is passed through the annotator, it is then compared to the correct version, i.e., its manually tagged counterpart, and transformations, that can be applied to the output of the initial state annotator to make it better resemble the truth , can then be learned.

During this process, one must specify the following: (1) the initial state annotator, (2) the space of transformations the learner is allowed to examine, and (3) the scoring function for comparing the corpus to the truth.

In the first version, there were transformation templates of the following example forms:

> Change tag **a** to tag **b** when:
> 1. The preceding (following) word is tagged **z**.
> 2. The preceding (following) word is tagged **z** and the word two before (after) is tagged **w**.

where **a**, **b**, **z** and **w** are variables over the set of parts-of-speech. To learn a transformation, the learner applies every possible transformation, counts the number of tagging errors after that transformation is applied, and chooses that transformation resulting in the greatest error reduction. Learning stops when no transformations can be found whose application reduces errors beyond some prespecified threshold. Once an ordered list of transformations is learned, new text can be tagged by first applying the initial annotator to it and then applying each of the learned transformations, in order.

Later in 1994, Brill extended this learning paradigm to capture relationships between words by adding contextual transformations that could make reference to the words as well as part-of-speech tags. Some examples of this transformation templates are:

> Change tag **a** to tag **b** when:
> 1. The preceding (following) word is **w**.
> 2. The current word is **w** and the preceding (following) word is **x**.
> 3. The current word is **w** and the preceding (following) word is tagged **z**.

where **w** and **x** are variables over all words in the training corpus, and **z** is a variable over all parts-of-speech.

This tagger has remarkable performance. After training the tagger with the corpus of size 600K, it produces 219 rules and generates 96.9% accuracy in the first scheme. Moreover, after the extension, number of rules increases to 267 and accuracy increases to 97.2%.

2.2.2 Statistical Approaches

Marshall [11] describes the Lancaster-Oslo-Bergen (LOB) Corpus tagging algorithm, later named CLAWS, as similar to TAGGIT program. The tag set used is very similar, but somewhat larger, at about 130 tags. The dictionary used is derived from the tagged Brown Corpus, rather than from the untagged version. It contains 7,000 rather than 3,000 entries, 700 rather than 450 suffixes. CLAWS treats plural, possessive, and hyphenated words as special cases for purposes of initial tagging.

The LOB researchers began by using TAGGIT on parts of the LOB Corpus. They noticed that, while less than 25% of TAGGIT's context frame rules are concerned with only the immediately preceding or succeeding word, these rules were applied in about 80% of all attempts to apply rules. This relative overuse of minimally specified contexts indicated that exploitation of the relationship between successive tags, coupled with a mechanism that would be applied throughout a sequence of ambiguous words, would produce a more accurate and effective method of word disambiguation.

The main innovation of the CLAWS is the use of a matrix of *collocational probabilities*, indicating the relative likelihood of co-occurrence of all ordered pairs of tags. This matrix can be mechanically derived from any pre-tagged corpus. CLAWS used a large portion of the Brown Corpus, with 200,000 words.

The ambiguities contained within a span of ambiguous words define a precise number of complete sets of mappings from words to individual tags. Each such assignment of tags is called a path. Each path is composed of a number of tag collocations, i.e., tags occuring side by side, and each such collocation has a probability which may be obtained from the collocation matrix. One may thus approximate each path's probability by the product of the probabilities of all its collocations. Each path corresponds to a unique assignment of tags to all words within a span. The paths constitute a span network, and the path of maximal probability may be taken to contain the *best* tags.

There are several advantages of this general approach over rule-based ones.

First, spans of unlimited length can be handled. Although earlier researchers have suggested that spans of length over 5 are rare enough to be of little concern, this is not the case. The number of spans of a given length is a function of that length and the corpus size, so long spans may be obtained

merely by examining more text. Second, a precise mathematical definition is possible for the fundamental idea of CLAWS. Whereas earlier efforts were based primarily on ad hoc sets of rules and descriptions, and employed substantial exception dictionaries. This algorithm requires no human intervention for set-up, it is a systematic process.

During the tagging process of the LOB Corpus a program called IDIOMTAG is used as an extension to CLAWS. IDIOMTAG is applied after initial tag assignment and before disambiguation. It was developed as a means of dealing with idiosyncratic word sequences which would otherwise cause difficulty for the automatic tagging. For example, *in order that* is tagged as a single conjunction. Approximately 1% of running text is tagged by IDIOMTAG.

CLAWS has been applied to the entire LOB Corpus with an accuracy of between 96% and 97%. Without the idiom list, the algorithm was 94% accurate on a sample of 15,000 words. Thus, the preprocessing of 1% of all tokens resulted in a 3% change in accuracy; those particular assignments must therefore have had a substantial effect on their context, resulting in changes of two other words for every one explicitly tagged.

However, CLAWS is time- and storage-inefficient in the extreme. Since CLAWS calculates the probability of every path, it operates in time and space proportional to the product of all the degrees of ambiguity of the words in the span. Thus, the time is exponential in the span length.

Later in 1988, DeRose [4] attempted to solve the inefficiency problem of the CLAWS and proposed a new algorithm called VOLSUNGA. The algorithm depends on a similar empirically-derived transitional probability matrix to that of CLAWS, and has a similar definition of optimal path. The tag set is larger than TAGGIT's, though smaller than CLAWS, containing 97 tags. The ultimate assignments of tags are much like of those of CLAWS.

The optimal path is defined to be the one whose component collocations multiply out to the highest probability. The more complex definition applied by CLAWS, using the sum of all the paths at each node of the network, is not used. By this change VOLSUNGA overcomes complexity problem.

VOLSUNGA does not use tag triples and idioms. Because of this, manually constructing special-case lists is not necessary. Application of the algorithm to Brown Corpus resulted with the 96% accuracy, even though idiom tagging

were not used.

A form of Markov model has also been widely used in statistical approaches. In this model it is assumed that a word depends probabilistically on just its part-of-speech category, which in turn depends solely on the categories of the preceding two words. Two types of training have been used with this model. The first makes use of a tagged training corpus. The second method of training does not require a tagged training corpus. In this situation the Baum-Welch algorithm can be used. Under this regime, the model is called a *Hidden Markov Model* (HMM), as state transitions (i.e., part-of-speech categories) are assumed to be unobservable.

In 1988, Church [2] built a tagger using the first training regime. He extracted all possible readings of each word and their usage frequencies from previously tagged Brown Corpus. The lexical probabilities were estimated in the obvious way. For example, the probability that "I" is a pronoun, $Prob(PPSS|"I")$, is estimated as the $freq(PPSS|"I")/freq("I")$. The contextual probability, the probability of observing part-of-speech X given the following two parts-of-speech Y and Z, is estimated by dividing the trigram frequency XYZ by the bigram frequency YZ. Thus, for example, the probability of observing a verb before an article and a noun is estimated to be the ratio of $freq(VB, AT, NN)$ over the $freq(AT, NN)$.

A search is performed in order to find the assignment of part-of-speech tags to words that optimizes the product of the lexical and contextual probabilities. Conceptually, the search enumerates all possible assignments of parts-of-speech to input words. Each sequence is then scored by the product of the lexical probabilities and the contextual probabilities, and the best sequence is selected. In fact, it is not necessary to enumerate all possible assignments because the scoring function can not see more than two words away. In other words, in the process of enumerating part-of-speech sequences, it is possible in some cases to know that some sequence can not possibly compete with another and can therefore be abandoned. Because of this fact, only $O(n)$ paths will be enumerated. Church states that, "The program performance is encouraging. 95-99% correct, depending on the definition of the correct". But he does not provide any definitions.

Cutting et al. [3] built a tagger using an HMM, which permits complete flexibility in the choice of training corpora. Text from any desired domain can

be used, and the tagger can be tailored for use with a particular text database by training on a portion of that database. The HMM model they used is quite a complicated one the details of which are not necessary here. They claim that, they have produced reasonable results training on a few as 3,000 sentences.

Statistical models have the advantage of automatic training. Required parameters for tagging can be extracted automatically, on a sufficiently large previously tagged corpus, whereas rule-based taggers require a large effort for rule crafting. This major drawback of rule-based models seems to be overridden with the employment of new learning mechanisms, like transformation-based error-driven learning proposed by Brill [1].

Chapter 3

Morphosyntactic Ambiguities in Turkish

Turkish is an agglutinative language with word structures formed by productive affixations of derivational and inflectional suffixes to the root words. Extensive use of suffixes results in ambiguous lexical interpretations in many cases [12, 13]. As shown earlier in Table 1.1 almost 80% of each lexical item has more than one interpretation. In this section, the sources of morphosyntactic ambiguity in Turkish is explored.

- Many words have ambiguous readings even though they have the same morphological break-down. These ambiguities are due to different POS of roots. For example the word *yana* has three different readings:[1]

	yana	**Gloss**	**POS**	**English**
1.	yan+yA	V(yan)+OPT+3SG	V	let it burn
2.	yan+yA	N(yan)+3SG+DAT	N	to this side
3.	yana	POSTP(yana)	POSTP	

  The first and the second readings have the same root and derived with the same suffix, but since the root word *yan* has two different readings, one verbal and one nominal, morphological analyzer produces ambiguous output for the same break-down. Moreover, *yana* has a third postpositional reading without any affixation.

  Another example is the word *en*.

[1] Among the possible readings of words produced by the morphological analyzer, ones which are irrelevant to the example case are discarded.

	en	**Gloss**	**POS**	**English**
1.	en	N(en)+3SG+NOM	N	width
2.	en	ADV(en)	ADV	most

It is two way ambiguous without any derivation due to two different parts-of-speech of the root.

- In Turkish, there are many root words which are prefix of another root word. This also creates ambiguous readings under certain circumstances. An example is:

  Of the two root words, *uymak* and *uyumak*, *uy* is a prefix of *uyu* and when the morphological analyzer is fed with the word *uyuyor*, it outputs the following:

	uyuyor	**Gloss**	**POS**	**English**
1.	uy+Hyor	V(uy)+PR-CONT+3SG	V	it suits
2.	uyu+Hyor	V(uyu)+PR-CONT+3SG	V	s/he is sleeping

  There are several other examples of this kind.

  e.g., *hara/haram*

	haram	**Gloss**	**POS**	**English**
1.	hara+Hm	N(hara)+3SG+1SG-POSS+NOM	N	my horse farm
2.	haram	ADJ(haram)+3SG+NOM	ADJ	unlawful

  e.g., *deva/devam*

	devam	**Gloss**	**POS**	**English**
1.	deva+Hm	N(deva)+3SG+1SG-POSS+NOM	N	my cure
2.	devam	N(devam)+3SG+NOM	N	continuation

- Nominal lexical items with nominative, locative or genitive case, have verbal/predicative interpretations. For example, the word *evde* is the locative case of the root word *ev*. And the morphological analyzer produces the following output for it.

	evde	**Gloss**	**POS**	**English**
1.	ev+DA	N(ev)+3SG+LOC	N	at home
2.	ev+DA	N(ev)+3SG+LOC+NtoV()+PR-CONT	V	(smt.) is at home

  For the following sentences:[2]

Ev-de	**şeker**	**kal-ma-mış.**
home+LOC	sugar	exist+NEG+NARR

  (There is no sugar at home.)

[2]Example sentences are given with their morphological break-down for clarity.

Bütün	**kitap-lar-ım**	**ev-de.**
all	book+PLU+1SG-POSS	home+LOC

(All of my books are at home.)

evde has a nominative reading in the first sentence and a predicative reading in the second one.

- There are morphological structure ambiguities due to the interplay between morphemes and phonetic change rules. Following is the output of morphological analyzer for the word *evin*:

	evin	**Gloss**	**POS**	**English**
1.	ev+Hn	N(ev)+3SG+2SG-POSS+NOM	N	your house
2.	ev+nHn	N(ev)+3SG+GEN	N	of the house

Since the suffixes have to harmonize in certain aspects with the word affixed, the consonant "n" is deleted in the surface realization of the second reading of evin, causing it to have same lexical form with the first reading.

Another example is the surface form realization of accusative and either third person singular possessive (3SG-POSS) or third person plural possessive (3PL-POSS) form of nominals.

	eli	**Gloss**	**POS**	**English**
1.	el+sH	N(el)+3SG+3PS-POSS	N	his/her hand
2.	el+yH	N(el)+3SG+ACC	N	hand (accusative)

- Within a word category, e.g., verbs, some of the roots have specific features which are not common to all. For example, certain reflexive verbs may also have passive readings. Consider the following sentences:

Çamaşır-lar	**dün**	**yıka-n-dı.**
cloth+PLU	yesterday	wash+PASS+PAST

(Clothes were washed yesterday.)

Ali	**dün**	**yıka-n-dı.**
Ali	yesterday	wash+REFLEX+PAST

(Ali washed himself yesterday.)

Following is the morphological break-down of *yıkandı*:

	yıkandı	Gloss	POS	English
1.	yıka+Hn+DH	V(yıka)+PASS+PAST+3SG	V	got washed
2.	yıka+n+DH	V(yıka)+REFLEX+PAST+3SG	V	s/he had a bath

From the same verbal root *yıka* two different break-downs are produced. Passive reading of *yıkandı* is used in the first sentence and the reflexive reading is used in the second sentence.

- Some lexicalized word formations can also be re-derived from the original root and this is another source of ambiguity. The word *mutlu* has two parse with the same meaning, but different morphological break-down.

	mutlu	Gloss	POS	English
1.	mut+lH	N(mut)+NtoADJ(li)+3SG+NOM	ADJ	happy
2.	mutlu	ADJ(mutlu)+3SG+NOM	ADJ	happy

mutlu has a lexicalized adjectival reading where it is considered as a root form as seen in the second reading. However, the same surface form is also derived from the nominal root word *mut*, meaning happiness, with the suffix *+li*, and this form also has the same meaning.

- Plural forms may display an additional ambiguity due to drop of a second plural marker. Consider the example word *evleri*.

	evleri	Gloss	POS	English
1.	ev+lAr+sH	N(ev)+3PL+3PS-POSS	N	his/her houses
2.	ev+lArH	N(ev)+3SG+3PL-POSS	N	their house
3.	ev+lArH	N(ev)+3PL+3PL-POSS	N	their houses
4.	ev+lAr+yH	N(ev)+3PL+ACC	N	houses (accusative)

In the first and the second reading there is only one level of plurality, where either the owner or the ownee is plural. However, the third reading contains a hidden suffix, where both of them are plural. Since it is not possible to detect which one is plural from the surface form, three ambiguous readings are generated.

Considering all these cases, it is apparent that the higher level analysis of Turkish prose text will suffer from this considerable amount of ambiguity. On the other hand as mentioned in the introduction, available local context might be sufficient to resolve some of these ambiguities. For example, if we can trace the sentential positions of nominal forms in a given sentence, their predicative readings might be discarded, i.e., within a noun phrase it is obvious that they can not be predicative.

In the next chapter, the answer to the question *"How can we eliminate these ambiguities?"* will be elaborated.

Chapter 4

The Tagging Tool

The tagging tool for Turkish developed in this thesis integrates a number of functionalities with a user interface as shown in Figure 4.1. The user interface is implemented under X-windows, and enables tagger to be used interactively, though user interaction is optional.

4.1 Functionality Provided by the Tool

The tagger uses a morphological analyzer for acquiring all readings of each word in a given Turkish prose text. The morphological analyzer [12] has a full-scale two-level description which has been implemented using the PC-KIMMO environment and it is based on a root word lexicon of about 23,000 root words. The phonetic rules of contemporary Turkish have been encoded using 22 two-level rules while the morphotactics of the agglutinative word structures have been encoded as finite-state machines for verbal, nominal paradigms and other categories.

The morphological analyzer returns all legitimate morphological break-downs of each word. This output is usually ambiguous due to the reasons explained in the previous section. So the main purpose of the tagger is to assign unique grammatical roles to each word by performing a certain amount of ambiguity resolution. For this purpose tagger utilizes following sources of information:

1. description of multi-word and idiomatic construct patterns,

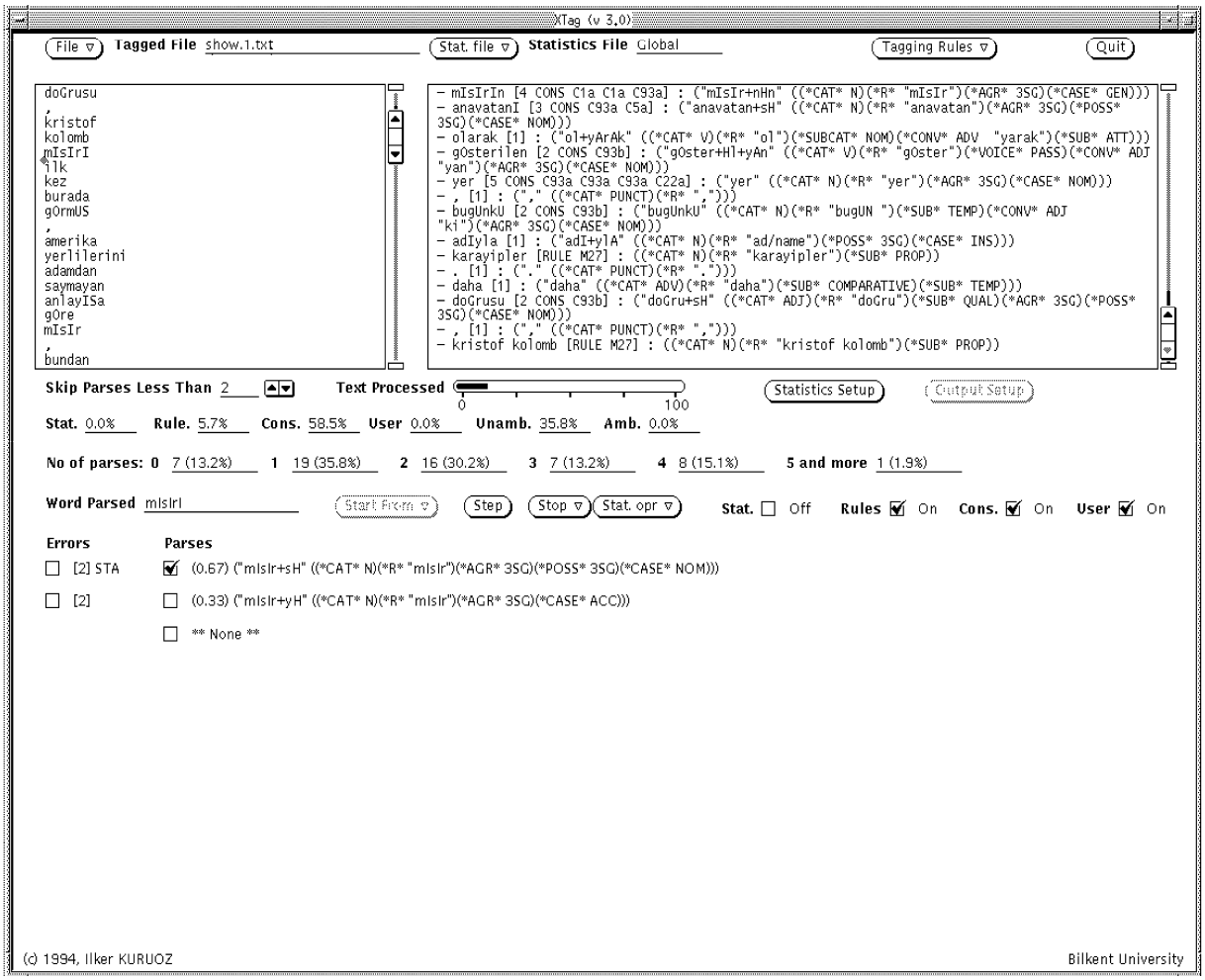

Figure 4.1. The user interface of tagging tool

2. a set of constraint rules and heuristics to eliminate illegitimate readings,

3. user assistance if all above fails.[1]

The first and the second sources of information is processed by a rule-based subsystem.

4.2 Rule-based Disambiguation

Multi-word and idiomatic construct recognition and constraint-based morphological disambiguation are implemented as a rule-based subsystem in which one can write rules of the following form:

$$\mathtt{C}_1\mathtt{:A}_1\mathtt{;}\ \ \mathtt{C}_2\mathtt{:A}_2\mathtt{;}\ \ \ldots\ \ \ \mathtt{C}_n\mathtt{:A}_n\mathtt{.}$$

where each $\mathtt{C}_i$ is a set of constraints on a lexical form, and the corresponding $\mathtt{A}_i$ is an action to be executed on the set of parses associated with that lexical form, *only when all the conditions are satisfied.*

Conditions refer to any available morphological or positional information associated with a lexical form such as:

- Absolute or relative lexical position (e.g., sentence initial or final, or 1 after the current word, etc.)
- root and final POS category,
- derivation type,
- case, agreement (number and person), and certain semantic markers, for nominal forms,
- aspect and tense, subcategorization requirements, verbal voice, modality,and sense for verbal forms
- subcategorization requirements for postpositions.

[1] They are all optional sources. If user does not want to assist, ambiguities are left pending.

Conditions may refer to absolute feature values or variables (as in Prolog, denoted by the prefix _ in the following examples) which are then used to link conditions. *All occurrences of a variable have to unify for the match to be considered successful.* This feature is powerful and necessary for a language like Turkish with agglutinative nature where one can not limit the tag set and has to use the morphological information. Using this we can specify rather general ways long distance feature constraints in complex NPs, PPs and VPs. This is a feature of our system that differentiates it from others.

The actions are of the following types:

- Null action: Nothing is done on the matching parse.
- **Delete**: Removes *the matching parse* if more than one parse for the lexical form are still in the set associated with the lexical form.
- **Output**: Removes *all but the matching parse* from the set effectively tagging the lexical form with the matching parse.
- **Compose**: Composes a new parse from various matching parses, for multi-word constructs.

These rules are ordered, and applied in the given order and actions licensed by any matching rule are applied. One rule formalism is used to encode both multi-word constructs and constraints.

4.3 The Multi-word Construct Processor

As mentioned before, tagging text on lexical item basis may generate spurious or incorrect results when multiple lexical items act as single syntactic or semantic entity. For example, in the following sentence:

Şirin	**mi**	**şirin**	**bir**	**köpek**
pretty	ques[2]	pretty	a	dog
koş-a	**koş-a**	**gel-di.**		
run+AOR	run+AOR	come+PAST		

(A very cute dog came running.)

The fragment *"şirin mi şirin"* constitutes a duplicated emphatic adjective in which there is an embedded question suffix *"mi"* (written separately in Turkish),[3] and the fragment *"koşa koşa"* is a duplicated verbal construction which has the grammatical role of manner adverb in the sentence, though both of the constituent forms are verbal constructions. The purpose of the multi-word construct processor is to detect and tag such constructs in addition to various other semantically coalesced forms such as proper nouns, etc.

4.3.1 The Scope of Multi-word Construct Recognition

Following list is a set of multi-word constructs for Turkish that we handle in our tagger. This list is not meant to be comprehensive, they are the ones we have encountered during the design of a parser for Turkish, obviously new construct specifications can be easily added. It is conceivable that such a functionality can be used in almost any language.

1. duplicated optative and 3SG verbal forms functioning as manner adverb, e.g., *koşa koşa* (running as in "he came running"),
2. aorist verbal forms with root duplications and sense negation functioning as temporal adverbs, e.g., *yapar yapmaz* (as soon as (one) does (something)),
3. duplicated verbal and derived adverbial forms with the same verbal root acting as temporal adverbs, e.g., *gitti gideli* (ever since (one) went),
4. duplicated compound nominal form constructions that act as adjectives, e.g., *güzeller güzeli* (very beautiful),
5. adjective or noun duplications that act as manner adverbs, e.g., *hızlı hızlı* (in a rapid manner), *ev ev* (house by house),
6. emphatic adjectival forms involving the question suffix, e.g., *güzel mi güzel* (very beautiful),
7. word sequences with specific usage whose semantics is not compositional, e.g., *yanı sıra (in addition to), hiç olmazsa* (in any case),

[2] *"mi"* is a question particle which is written separately in Turkish.

[3] If, however, the adjective *"şirin"* was not repeated, then we would have a question formation.

8. proper nouns, e.g., *Jimmy Carter*, *Topkapı Sarayı* (Topkapı Palace),

9. idiomatic forms which are never used singularly, e.g., *gürül gürül*,

10. other idiomatic forms, such as *ipe sapa gelmez* (worthless) which is only used as an adjective.

11. compound verb formations which are formed by a lexically adjacent, direct or oblique object and a verb, which for the purposes of parsing may be considered as single lexical item, such *saygı durmak* (to pay respect), *kafayı yemek* (literally *to eat the head* – to get mentally deranged), etc. The rare cases where some other lexical item intervenes between the object and the verb, have to be dealt at the syntactic level.

4.3.2 Multi-word Construct Specifications

In our tagger, multi-word constructs are specified using the previously defined rule format. However, among those actions only **Compose** is available.

The main idea is to apply multi-word specifications on each word to find a matching pattern. If any matching pattern is found, the involved words are discarded and a new composite lexical entry as specified in the compose action created.

Rule ordering is important for specifications and they are applied in the given order. This property is vital for recognition of patterns which are superset[4] of some other rules. Proper nouns with more than one constituents are good examples of this case. e.g., there are several combinations of usage for the proper noun *"Mustafa Kemal Atatürk"* like *"Mustafa Kemal"*, *"Kemal Atatürk"* and *"Atatürk"*. If they are not specified in the order from the longest one to shortest one, it may not be possible to recognize the longer usages, since a shorter one can be matched before the longer specification applied.[5] Assume *"Mustafa Kemal"* is specified before *"Mustafa Kemal Atatürk"*, in a given text if we encounter the word sequence .. *Mustafa Kemal Atatürk Samsun'a* .. the first two words will be matched by the smaller specification and coalesced into

[4]A rule is a superset of another one, if available feature-value pairs satisfying the constraints of the rule also satisfies the other one, but the reverse does not hold.

[5]The specification of *"Mustafa Kemal Atatürk"* is a superset of the specification of *"Mustafa Kemal"*.

a single lexical item and the tagger will miss to match the longer one *"Mustafa Kemal Atatürk"*.

Example Specifications

Here we present some examples of multi-word construct specifications.[6]

First specification example:

```
(C1) Lex = _W1, Root = _R1, Cat = V, Aspect = AOR,
     Agr = 3SG, Sense = POS:
(A1) Null;
(C2) Lex = _W2, Root = _R1, Cat = V, Aspect = AOR,
     Agr = 3SG, Sense = NEG:
(A2) Compose = ((*CAT* ADV)(*R* "_W1 _W2 (_R1)")(*SUB* TEMP)).
```

This rule would match any adjacent verbal lexical forms with the same root, both with the aorist aspect, and 3SG agreement, e.g., *yapar yapmaz*. The first verb must have a positive and the second one must have a negative sense. When such two adjacent lexical items found, a composite lexical form with an temporal adverb part-of-speech is then generated. The original verbal root may be recovered from the root of the composed form for any subcategorization checks at the syntactic level.

For the following sentence,

Ali	**iş-i-ni**	**tamamlar**
Ali	duty+3SG-POSS+ACC	complete+AOR
tamamlamaz	**git-ti.**	
complete+NEG+AOR	go+PAST	

(Ali went as soon as he completed his task.)

The output of morphological analyzer is given in Figure 4.2.

[6]The output of the morphological analyzer is actually a *feature-value* list in the standard LISP format.

```
("ali"
     ("ali'" ((*CAT* N)(*R* "ali")(*SUB* PROP)
              (*AGR* 3SG)(*CASE* NOM)))
)

("işini
     ("iş+sH+nH" ((*CAT* N)(*R* "iş")(*AGR* 3SG)
                  (*POSS* 3SG)(*CASE* ACC)))
     ("iş+Hn+yH" ((*CAT* N)(*R* "iş")(*AGR* 3SG)
                  (*POSS* 2SG)(*CASE* ACC)))
)

("tamamlar"
     ("tamam+lAr" ((*CAT* N)(*R* "tamam")(*ROLE* ADJ)
                   (*AGR* 3PL)(*CASE* NOM)))
     ("tamam+lAr" ((*CAT* N)(*R* "tamam")(*ROLE* ADJ)
                   (*AGR* 3PL)(*CASE* NOM)(*CONV* V "")
                   (*ASPECT* PR-CONT)(*AGR* 3SG)))
     ("tamamla+Hr" ((*CAT* V)(*R* "tamamla")
                    (*SENSE* POS)(*ASPECT* AOR)
                    (*AGR* 3SG)))
     ("tamamla+Hr" ((*CAT* V)(*R* "tamamla")
                    (*CONV* ADJ "ir")))
)

("tamamlamaz"
     ("tamamla+mA+z" ((*CAT* V)(*R* "tamamla")
                      (*SENSE* NEG)(*ASPECT* AOR)
                      (*AGR* 3SG)))
)

("gitti"
     ("gid+DH" ((*CAT* V)(*R* "git")(*ASPECT* PAST)
                (*AGR* 3SG)))
)
```

Figure 4.2. Output of morphological analyzer for *"Ali işini tamamlar tamamlamaz gitti"*.

For the consecutive words *"tamamlar tamamlamaz"*, when the rule is applied to the fourth reading of *"tamamlar"* and the available single reading of *"tamamlamaz"*, we see that variable references to their roots, i.e., _R1, they unify, since both word have the same root. They both have aorist aspect, third person singular agreement and the first one has a positive sense and the second one has a negative sense. Therefore, the compose action is applied, both words are dropped and a new lexical item with temporal adverb part-of-speech is generated.

```
("tamamlar tamamlamaz"
     ((*CAT* ADV)(*R* "tamamlar tamamlamaz (tamamla)")
      (*SUB* TEMP))
)
```

Note that variable references to surface lexical forms of each word are utilized as the output generated.

The next example is for recognition of emphatic adjectival forms with a question suffix in between, e.g., *güzel mi güzel.*

```
(C1) Lex = _W1, Root = _R1, Cat = ADJ:
(A1) Null;
(C2) Lex = _W2, Root = mi , Cat = QUES:
(A2) Null;
(C3) Lex = _W3, Root = _R1, Cat = ADJ:
(A3) Compose = ((*CAT* ADJ)(*R* "_W1 _W2 _W3 (_R1)")).
```

This rule would match any consecutive three words, where the first and the third have the same root and adjectival readings and there is a question suffix in between. If such a combination is found they are coalesced into a single lexical form with adjectival part-of-speech.

This multi-word construct recognition facility is very efficient for the recognition of proper nouns. Following rule is written for recognition of the proper noun *"Mustafa Kemal Atatürk"*.

```
(C1) Lex = Mustafa :
```

```
(A1) Null;
(C2) Lex = Kemal :
(A2) Null;
(C3) Lex = Atatürk :
(A3) Compose = ((*CAT* N)(*R* "Mustafa Kemal Atatürk")
                (*SUB* PROP)$).
```

In this rule we are only concerned with the surface form of each word. In a given text if there are three adjacent words with the given lexical surface form they are combined to make a single lexical item. the $ sign at the end of compose action implies applicability of inheritance, i.e., if the text contains a word sequence like ... *Mustafa Kemal Atatürk'ün evi* ... it is apparent that Atatürk has a genitive case, hence when the sequence is matched the output generated should indicate that the new lexical item has the genitive case, and the output will be:

```
("Mustafa Kemal Atatürk"
     ((*CAT* N)(*R* "Mustafa Kemal Atatürk")(*SUB* PROP)
      (*CASE* GEN))
)
```

This inheritance property is available only for the last word in the sequence.

4.4 Using Constraints for Morphological Ambiguity Resolution

Morphological analysis does not have access to syntactic context, so when the morphological structure of a lexical form has several distinct analyses, it is not possible to disambiguate such cases except maybe by using root usage frequencies. For disambiguation one may have to use to usage information provided by sentential position and the local morphosyntactic context.

In our tagger, constraint rules are specified by using the previously defined rule format, in a way very similar to specification of multi-word constructs. Use of variables, operators and actions are same except that the compose action does not make sense here.

The task of morphological disambiguator is to discard all and only the contextually illegitimate alternative morphological readings. It employs an ordered set of linguistic constraints on the linear order of ambiguity forming morphological readings. Constraint rules are applied on each word, and requires recognition of a sequence of forms with certain morphological features, sentential positions (i.e., sentence beginning or final) on it and its neighbors. Constraint rules contain some very general linguistic rules, a set of heuristics and some biased preferences on some words depending on the corpus we use.

Since there is a rule order, rule crafting requires attention to rule ordering, but this ordering enables us to handle exceptions and impose our biased preferences on the readings of some words.

So far, about 250 constraint rules have been written, and they disambiguate 98% to 99% of a given text automatically with 97% to 99% accuracy. Detailed information on the results of the experiments will be given in the next chapter.

In the next section, several examples of constraint rules will be presented.

4.4.1 Example Constraint Specifications

The following constraint is used to select the postpositional reading of certain word when it is preceded by a yet unresolved nominal form with a certain case.

```
(C1) LP = 0, FinalCat != V , Case = _C : (A1) Output;
(C2) LP = 1, Cat = POSTP, Subcat = _C : (A2) Output.
```

For example, for the following sentence,

Ahmet'ten	**önce**	**Ali**	**git-ti.**
Ahmet+ABL	before	Ali	go+PAST

(Before Ahmet, Ali went.)

the morphological analyzer outputs the break-downs given in Figure 4.3:

If the rule is applied to the word sequence *"Ahmet'ten önce"*, the postpositional reading of *"önce"* is chosen and the others are discarded. Since, the

```
("ahmet'ten"
     ("ahmet'+DAn" ((*CAT* N)(*R* "ahmet")(*SUB* PROP)
                    (*CASE* ABL)))
)

("önce"
     ("önce" ((*CAT* N)(*R* "önce")(*AGR* 3SG)(*CASE* NOM)))
     ("önce" ((*CAT* N)(*R* "önce")(*AGR* 3SG)(*CASE* NOM)
              (*CONV* V "")(*ASPECT* PR-CONT)(*AGR* 3SG)))
     ("önce" ((*CAT* ADV)(*R* "önce")(*SUB* TEMP)))
     ("önce" ((*CAT* POSTP)(*R* "önce")
              (*SUBCAT* ABL)))
)

("ali"
     ("ali'" ((*CAT* N)(*R* "ali")(*SUB* PROP)(*CASE* NOM)))
     ("ali'" ((*CAT* N)(*R* "ali")(*SUB* PROP)(*CASE* NOM)
              (*CONV* V "")(*ASPECT* PR-CONT)(*AGR* 3SG)))
)

("gitti"
     ("gid+DH" ((*CAT* V)(*R* "git")(*ASPECT* PAST)
                (*AGR* 3SG)))
)
```

Figure 4.3. Output of morphological analyzer for *"Ahmet'ten önce Ali gitti"*.

only requirement is that the case of the nominal form agrees with the subcategorization requirement of the following postposition. (LP = 0 refers to current word, LP = 1 refers to next word.)

This one constraint disambiguates almost all of the postpositions and their arguments, the exceptions being nominal words which semantically convey the information provided by the case (such as words indicating direction, which may be used as if they have a dative case), e.g., *yukarı doğru.* Following is the morphological analyzer's output.

```
("yukarı"
     ("yukarı" ((*CAT* N)(*R* "yukarı")
                (*SEMCASE* DAT)(*AGR* 3SG)
                (*CASE* NOM)))
     ("yukarı" ((*CAT* N)(*R* "yukarı")(*SEMCASE* DAT)
                (*AGR* 3SG)(*CASE* NOM)(*CONV* V "")
                (*ASPECT* PR-CONT)(*AGR* 3SG)))
     ("yukarı" ((*CAT* ADV)(*R* "yukarı")(*SUB* DIR)))
     ("yukarı" ((*CAT* POSTP)(*R* "yukarı")(*SUBCAT* ABL)))
)

("doğru"
     ("doğru" ((*CAT* ADJ)(*R* "doğru")(*SUB* QUAL)
               (*AGR* 3SG)(*CASE* NOM)))
     ("doğru" ((*CAT* ADJ)(*R* "doğru")(*SUB* QUAL)
               (*AGR* 3SG)(*CASE* NOM)(*CONV* V "")
               (*ASPECT* PR-CONT)(*AGR* 3SG)))
     ("doğru" ((*CAT* POSTP)(*R* "doğru")
               (*SUBCAT* DAT)))
)
```

Following rule handles this cases.

```
(C1) LP = 0, FinalCat != V, SemCase = _C : (A1) Output;
(C2) LP = 1, Cat = POSTP, Subcat = _C: (A2) Output.
```

Some of the postpositions have exceptional cases of its own. For example, postpositional readings of *"sonra"* and *"önce"* do not require any agreement between its sub-categorization requirement and the previous nominal's case if the nominal indicates a temporal unit.

```
(C1) LP = 0, FinalCat = N, Sub = TEMP-UNIT, Case = NOM:
(A1) Output;
(C2) LP = 1, Cat = POSTP, R = önce :
(A2) Output.
```

Example:

Ali	**on**	**gün**	**önce**	**gel-di.**
Ali	ten	day	ago	come+PAST

(Ali came ten days ago.)

In this sentence, *"önce"* has a postpositional reading even though its sub-categorization requirement does not agree with the case of *"gün"*. Consider the following morphological break-down for the fragment *"gün önce"*.

```
("gün"
     ("gün" ((*CAT* N)(*R* "gün")
             (*SUB* TEMP-UNIT)(*AGR* 3SG)
             (*CASE* NOM)))
     ("gün" ((*CAT* N)(*R* "gün")(*SUB* TEMP-UNIT)
             (*AGR* 3SG)(*CASE* NOM)(*CONV* V "")
             (*ASPECT* PR-CONT)(*AGR* 3SG)))
)

("önce"
     ("önce" ((*CAT* N)(*R* "önce")(*AGR* 3SG)
              (*CASE* NOM)))
     ("önce" ((*CAT* N)(*R* "önce")(*AGR* 3SG)
              (*CASE* NOM)(*CONV* V "")(*ASPECT* PR-CONT)
              (*AGR* 3SG)))
```

```
        ("önce" ((*CAT* ADV)(*R* "önce")(*SUB* TEMP)))
        ("önce" ((*CAT* POSTP)(*R* "önce")
                 (*SUBCAT* ABL)))
)
```

The first two constraint specifications given earlier disambiguate almost all postpositions, but they have to be complemented with the rules for handling exceptional cases, which are word specific as in the case of *"önce"*, i.e., they can not be generalized to a group of words. If none these rules can assert that the word is a postposition, the postpositional reading is deleted with the following rule.[7]

```
(C1) LP = 0, Cat = POSTP, R != ile :  (A1) Delete;
```

Recognition of noun phrases makes the disambiguation of their components easier. For this purpose, we have implemented a set of rules to disambiguate words which are constituents of noun phrases.

In the simplest case, a Turkish NP can be a possessive or non-possessive noun, a possessive or non-possessive adjective, a proper noun, a pronoun etc. But, it is somewhat hard to use this single word noun phrases for disambiguation purposes. So the recognition of noun phrases consisting of two or more components might be useful. In Turkish, noun phrases can be divided into two categories which are nominal compounds and adjectival compounds.

Nominal compounds are classified into definite and indefinite nominal compounds. Moreover, definite nominal compounds can be further classified as genitive-possessive compounds and possessive-compounds, since they have different syntactic properties.

In Turkish, a noun phrase with genitive suffix can modify a noun with possessive suffix to form a genitive-possessive construction, we can give examples as *benim evim* (my house), *çocuğun kitabı* (the child's book). The genitive suffix indicates that the noun which it is attached is possessor of some other noun. The possessive suffix indicates that the noun to which it is attached is possessed by other noun. Such examples are handled with the following rule.

[7] *"ile"* is a special postposition and hard to disambiguate. It might be better to leave it ambiguous.

```
(C1) LP = 0, FinalCat = N, Case = GEN, Agr = _A1 :
(A1) Output;
(C2) LP = 1, FinalCat = N, Poss = _A1 :
(A2) Output.
```

Following is the morphological break-down for *"çocuğun kitabı"*.

```
("çocuğun"
     ("çocuk+Hn" ((*CAT* N)(*R* "çocuk")(*AGR* 3SG)
                  (*POSS* 2SG)(*CASE* NOM)))
     ("çocuk+Hn" ((*CAT* N)(*R* "çocuk")(*AGR* 3SG)
                  (*POSS* 2SG)(*CASE* NOM)(*CONV* V "")
                  (*ASPECT* PR-CONT)(*AGR* 3SG)))
     ("çocuk+nHn" ((*CAT* N)(*R* "çocuk")
                   (*AGR* 3SG)(*CASE* GEN)))
     ("çocuk+nHn" ((*CAT* N)(*R* "çocuk")(*AGR* 3SG)
                   (*CASE* GEN)(*CONV* V "")
                   (*ASPECT* PR-CONT)(*AGR* 3SG)))
)

("kitabı"
     ("kitab+sH" ((*CAT* N)(*R* "kitap")(*AGR* 3SG)
                  (*POSS* 3SG)(*CASE* NOM)))
     ("kitab+sH" ((*CAT* N)(*R* "kitap")(*AGR* 3SG)
                  (*POSS* 3SG)(*CASE* NOM)(*CONV* V "")
                  (*ASPECT* PR-CONT)(*AGR* 3SG)))
     ("kitab+yH" ((*CAT* N)(*R* "kitap")(*AGR* 3SG)
                  (*CASE* ACC)))
)
```

Other definite nominal compounds that do not fit to the syntax of the genitive-possessive constructions are called as possessive compounds. Some examples are *kitap kapağı* (book cover), at arabası (horse cart). The following rule handles these NPs.

```
(C1) LP = 0, FinalCat = N, Case = NOM, Agr = 3SG :
(A1) Output;
(C2) LP = 1, FinalCat = N, Poss = 3SG :
(A2) Output.
```

Components of indefinite nominal compounds like *çelik kapı* (steel door), *altın bilezik* (golden bracelet), due to their uninflected morphological structure, do not have ambiguous readings except for the predicative readings, and we can resolve them using certain heuristics.

An adjective modifying a noun, precedes the noun to form an adjectival compound. Simple adjectival compounds are composed of two words like *kırmızı kalem* where the modifier is an adjective and the modified is a noun. But there are other adjectival compounds where the modifier adjective is derived form another form, like *bahçedeki*, where the final reading is again an adjective. Consider the following morphological break-down of *bahçedeki ağaç* (the tree in the garden).

```
("bahçedeki"
     ("bahçe+DA+ki" ((*CAT* N)(*R* "bahçe")(*AGR* 3SG)
                     (*CASE* LOC)(*CONV* ADJ "ki")
                     (*AGR* 3SG)(*CASE* NOM)))
     ("bahçe+DA+ki" ((*CAT* N)(*R* "bahçe")(*AGR* 3SG)
                     (*CASE* LOC)(*CONV* ADJ "ki")
                     (*AGR* 3SG)(*CASE* NOM)(*CONV* V "")
                     (*ASPECT* PR-CONT)(*AGR* 3SG)))
)

("ağaç"
     ("ağaç" ((*CAT* N)(*R* "ağaç")(*AGR* 3SG)
              (*CASE* NOM)))
     ("ağaç" ((*CAT* N)(*R* "ağaç")(*AGR* 3SG)(*CASE* NOM)
              (*CONV* V "")(*ASPECT* PR-CONT)(*AGR* 3SG)))
)
```

Such compound nouns are disambiguated with the following rule:

```
(C1) LP = 0, FinalCat = ADJ : (A1) Output;
(C2) LP = 1, FinalCat = N : (A2) Null.
```

Indeed, NPs can have arbitrary length by intervening modifiers in Turkish. So we must be able handle these cases also, for this purpose, we have implemented some other specifications for the disambiguation of NPs like *smt's ADJ smt*, *smt's N smt*, e.g, *evin yeşil kapısı* (green door of the house), *evin demir kapısı* (iron door of the house). On the other hand, in practice we have not encountered very complex noun phrases of length six or seven.

```
(C1) LP = 0, FinalCat = N, Case = GEN, Agr = _A1 :
(A1) Output;
(C2) LP = 1, FinalCat = ADJ :
(A2) Output;
(C3) LP = 2, FinalCat = N, Poss = _A1 :
(A3) Output.

(C1) LP = 0, FinalCat = N, Case = GEN, Agr = _A1 :
(A1) Output;
(C2) LP = 1, FinalCat = N :
(A2) Output;
(C3) LP = 2, FinalCat = N, Poss = _A1 :
(A3) Output.
```

Since our purpose is not to analyse the syntactic structure of noun phrases, the recursive nature of NPs, i.e., two or more NPs combined together to form another NP, helps us to disambiguate very long forms by disambiguting their smaller components with the previously defined rules.

These rules are selected examples from the set of rules we have specified for disambiguating components of noun phrases. The complete set contains a number of other rules to handle some exceptions, and proves a reliable performance.

Our rule specification contains various heuristics besides these linguistic generalizations. For example, the following rule deletes the sentence final adjectival readings derived from verbs, effectively preferring the verbal reading. This heuristic relies on the property that Turkish is an SOV language.

```
(C1) LP = 0, Cat = V, FinalCat = ADJ, SP = END :
(A1) Delete.
```

In the sentence

Mektup	**zaman-ın-da**	**ulaş-ır.**
mail	time+3SG-POSS+LOC	arrive+AOR

(Mail arrives in time.)

the word *ulaşır* has a verbal POS, but the morphological analyzer produces an ambiguous output including an adjectival POS;

```
("ulaşır"
     ("ulaş+Hr" ((*CAT* V)(*R* "ulaş")(*ASPECT* AOR)
                 (*AGR* 3SG)))
     ("ulaş+Hr" ((*CAT* V)(*R* "ulaş")
                 (*CONV* ADJ "ir")))
)
```

and the second reading is discarded by this rule.

Another heuristic is to discard second person singular possessive forms, since they can be observed usually in the dialogs. If the text does not contain any dialog this rule plays an efficient role.

```
(C1) LP = 0, FinalCat = ?, Poss = 2SG : (A1) Delete.
```

The question mark for the final category value means any feature value is valid, i.e., don't care. Consider the following sentence.

Ahmet	**ev-in-de**	**uyu-yor**
Ahmet	home+3SG-POSS+LOC	sleep+PR-CONT

(Ahmet is sleeping at his home.)

Among those readings of *"evin"*, we can discard the second person singular possessions.

```
("evinde"
     ("ev+sH+nDA" ((*CAT* N)(*R* "ev")(*AGR* 3SG)
                   (*POSS* 3SG)(*CASE* LOC)))
     ("ev+sH+nDA" ((*CAT* N)(*R* "ev")(*AGR* 3SG)
                   (*POSS* 3SG)(*CASE* LOC)(*CONV* V "")
                   (*ASPECT* PR-CONT)(*AGR* 3SG)))
     ("ev+Hn+DA" ((*CAT* N)(*R* "ev")(*AGR* 3SG)
                  (*POSS* 2SG)(*CASE* LOC)))
     ("ev+Hn+DA" ((*CAT* N)(*R* "ev")(*AGR* 3SG)
                  (*POSS* 2SG)(*CASE* LOC)(*CONV* V "")
                  (*ASPECT* PR-CONT)(*AGR* 3SG)))
)
```

Bearing the same idea in mind, i.e., there is no dialog in the text, we can discard imperative and optative readings of words. Following two rule handle these cases.

```
(C1) LP = 0 , Finalcat = V , Aspect = OPT: (A1) Delete.
```

```
(C1) LP = 0 , Finalcat = V , Aspect = IMP: (A1) Delete.
```

Consider following examples.

Bu	**Ahmet'in**	**kaz-ı.**
this	Ahmet+GEN	duck+3SG-POSS

(This is Ahmet's duck.)

In this sentence *"kazı"* has the following break-down:

```
("kazı"
     ("kaz+sH" ((*CAT* N)(*R* "kaz")(*AGR* 3SG)
                (*POSS* 3SG)(*CASE* NOM)))
     ("kaz+sH" ((*CAT* N)(*R* "kaz")(*AGR* 3SG)
                (*POSS* 3SG)(*CASE* NOM)(*CONV* V "")
```

```
                (*ASPECT* PR-CONT)(*AGR* 3SG)))
        ("kaz+yH" ((*CAT* N)(*R* "kaz")(*AGR* 3SG)(*CASE* ACC)))
        ("kazı" ((*CAT* N)(*R* "kazı")(*AGR* 3SG)(*CASE* NOM)))
        ("kazı" ((*CAT* N)(*R* "kazı")(*AGR* 3SG)(*CASE* NOM)
                 (*CONV* V "")(*ASPECT* PR-CONT)(*AGR* 3SG)))
        ("kazı" ((*CAT* V)(*R* "kazı")
                 (*ASPECT* IMP)(*AGR* 2SG)))
)
```

Final imperative reading can be safely deleted.

Ahmet	**soru-yu**	**çöz-me-ye**	**çalış-ıyor.**
Ahmet	question+ACC	solve+VtoN+DAT	try+PR-CONT

(Ahmet is trying to solve the question.)

"çözmeye" has two possible readings.

```
("çözmeye"
     ("çöz+mA+yA" ((*CAT* V)(*R* "çöz")(*SENSE* NEG)
                   (*ASPECT* OPT)(*AGR* 3SG)))
     ("çöz+mA+yA" ((*CAT* V)(*R* "çöz")(*CONV* N "ma")
                   (*AGR* 3SG)(*CASE* DAT)))
)
```

And the optative reading is obviously not possible.

4.4.2 Rule Crafting

Some selected examples from our current constraint rules are given in the previous section. It is probably noticable that they contain a set of linguistic generalizations, some exception handling rules and a set of heuristics.

All these rules have been specified after experimenting with a number of texts. The important topic here is the ordering of these specifications, since the rules are applied on each word in the given order. We preferred to give the

generalizations first, with rules handling their exceptions. Then the heuristics and our biased-preferences on some words, like prefering connective reading of *"ama"*(but) since it is very rare to meet its nominal reading (blind).

Apparently, many rules have side effects to other ones, and there is no automatic way of perfect ordering. Therefore, we have tried some different orderings for rules which interact by means of side effects to reach out a reasonable ordering.

4.4.3 Limitations of Constraint-based Disambiguation

Throughout the tests we have faced some language specific and some formalism specific problems.

One of the major problems with Turkish is the semantic ambiguities due to agglutinative nature of the language. For example, the word *kazanlar* has two nominal readings with same POS, yet different morphological breakdown.

	kazanlar	**Gloss**	**POS**	**English**
1.	kazan+lar	N(kazan)+3PL+NOM	N	boilers
2.	kaz+[y]an+lar	V(kaz)+VtoN(yan) +3PL+NOM	N	ones who are digging

Even though the second one is derived from a verbal root they both have the same final category and inflectional feature values. In English each word can have several readings with the same category, but their semantic ambiguity is not supplied by the morphological analyzer. In this case, however, choosing one of these readings as correct means resolution of a semantic ambiguity, although it is not one of the tasks of tagger.

One solution to this problem is to change the semantics of *output* action from remove *all but the matching parse* to remove *all but the matching parses*. This plurality means leaving the ambiguity pending, and moreover, increasing the number of words left ambiguous. In this version of tagger we preferred to resolve such ambiguities by writing some rules to dictate our biased preferences on the readings of some pre-determined words and some other rules to eliminate ambiguities due to such derivations.

Many sentences contain sub-ordinate clauses combined with a coordinating conjunctions like *ve* (and). Some of the ambiguities can not be resolved with the available formalism. For example, in the following sentence fragment:

Vazolar süslenmiş ve güzel masaların üzerinde duruyor.

This is tough to disambiguate the word *süslenmiş*. Whether it is a verb or an adjective which is a part of a noun phrase can not be decided easily. One possibility may be to write arbitrarily long rules to catch whether it is a part of noun phrase or not, but the length of the specification can not be predictable since there can be several other intervening words, and even noun phrases to which it can be attached, thus leading a wrong disambiguation. Another solution might be to add nondeterminism to constraint matching mechanism to check arbitrary distance away words in specified aspects to make decisions, but this may increase the complexity of rule matching mechanism upto the cost of syntactic parsing, which should be avoided.

Application of constraint specifications in the given linear order have a number of drawbacks, beside its advantages. One of the major one is the early disambiguation of certain lexical items. In the following sentence,

Ordu	**üs-ler-i**	**bombala-n-dı.**
army	base+PLU+3SG-POSS	bomb+PASS+PAST

(Army bases have been bombed.)

recognition and disambiguation of the noun phrase *"ordu üsleri"* (army bases) leads to a correct disambiguation within this discourse. However, if we convert the previous sentence to following form:

Ordu	**üs-ler-i**	**bombala-dı.**
army	base+PLU+ACC	bomb+PAST

(Army bombed the bases.)

disambiguation of the fragment *"ordu üsleri"* as the noun phrase *army bases* leads to an incorrect interpretation. Next sentence also contains an example of this case.

Aslında	**benim**	**devam**	**et-mek**	**gibi**
in fact	my	attendance	do+INF	as
bir	**niyet-im**	**yok.**		
a	intention+1SG-POSS	exist+NEG		

(In fact, I do not have any intention to continue.)

For the following piece of sentence *"..benim devam etmek.."* morphological analyzer produces the output given in Figure 4.4. As seen, *"benim devam"* will be disambiguated by one of the noun phrase recognizing rules as a noun phrase (my cure), therefore the tagger will miss the verb group *"devam etmek"*,or will not have any chance to evaluate this possibility. With the current formalism there seems to be no good solution to this problem.

Another major problem with the formalism is that it can not handle the word-order freeness of Turkish. The place of a constituent in a sentence may be changed according to various considerations like focus, topicalization etc. However, all constraints are applied in the given order on the linear order of the input text, so misplacement of constituents may decrease the performance of the tagger in two respects: The first and the most common one is that ambiguity can not be resolved, and the second one is the tagger may choose a wrong interpretation. Luckily, a straight forward prose text not containing dialogs ususally contains sentences which have the standard SOV order.

As repeated in the above cases, this rule formalism has certain limitations.

- One of the major problem is to find a reasonable ordering where the side effects of each rule to other rules are minimized.
- Another problem rises from the application of constraint rules in the given order leading to disambiguation of lexical items promptly when constraints are satisfied, and blocking application of other rules to assert different interpretations.
- This formalism can not capture the word-order freeness yet.

One solution to these above problems might be to use a different inference mechanism where disambiguation decisions do not depend solely on the actions of constraint rules, but on a scoring they made, i.e., application of constraint

```
.
.
("benim"
     ("ben+Hm" ((*CAT* N)(*R* "ben")(*AGR* 3SG)(*POSS* 1SG)
                (*CASE* NOM)))
     ("ben+Hm" ((*CAT* N)(*R* "ben")(*AGR* 3SG)(*POSS* 1SG)
                (*CASE* NOM)(*CONV* V "")(*ASPECT* PR-CONT)
                (*AGR* 3SG)))
     ("ben+yHm" ((*CAT* N)(*R* "ben")(*AGR* 3SG)
                 (*CASE* NOM) (*CONV* V "")
                 (*ASPECT* PR-CONT)(*AGR* 1SG)))
     ("ben+yHm" ((*CAT* PN)(*R* "ben")(*AGR* 1SG)
                 (*CASE* NOM)(*CONV* V "")(*ASPECT* PR-CONT)
                 (*AGR* 1SG)))
     ("benim" ((*CAT* PN)(*R* "ben")(*AGR* 1SG)
               (*CASE* GEN)))
)

("devam"
     ("deva+Hm" ((*CAT* N)(*R* "deva")(*AGR* 3SG)
                 (*POSS* 1SG)(*CASE* NOM)))
     ("deva+Hm" ((*CAT* N)(*R* "deva")(*AGR* 3SG)
                 (*POSS* 1SG)(*CASE* NOM)(*CONV* V "")
                 (*ASPECT* PR-CONT)(*AGR* 3SG)))
     ("devam" ((*CAT* N)(*R* "devam")(*AGR* 3SG)
               (*CASE* NOM)))
     ("devam" ((*CAT* N)(*R* "devam")(*AGR* 3SG)
               (*CASE* NOM)(*CONV* V "")(*ASPECT* PR-CONT)
               (*AGR* 3SG)))
)

("etmek"
     ("ed+mAk" ((*CAT* V)(*R* "et")(*CONV* INF "mak")
                (*CASE* NOM)))
     ("ed+mAk" ((*CAT* V)(*R* "et")(*CONV* INF "mak")
                (*CASE* NOM)(*CONV* V "")(*ASPECT* PR-CONT)
                (*AGR* 3SG)))
)
.
.
```

Figure 4.4. Output of morphological analyzer for *"..benim devam etmek.."*.

rules assigns scores to each readings of a word and the final decision is given on the overall scoring obtained. This modification will have two consequences: It eases the rule crafting, since the rule ordering will not be important. Moreover, each rule will have a chance to evaluate possible readings of lexical items, as none of them is disambiguated before the last rule is applied.

4.5 Text-based Statistical Disambiguation

A common practice in statistical language analysis is to generate statistics from a tagged, heterogeneous corpus like Brown and LOB,[8] and apply these statistics in the analysis of the new texts. If the new text is different from the statistical model of the source corpus, the analyzer is likely to perform less satisfactorily.

A new possibility is to use the analyzed corpus itself both as a source of generalizations and an object of the analyzer based on these generalizations. For this purpose, the tagger compiles root usage statistics from the fully disambiguated part of the text, and if any of the previous techniques can not resolve the ambiguity, optionally the tagger can use these statistics to select proper analysis according to some user specified thresholds. However, we have not implemented this yet.

[8]So far, there is no such a big tagged corpus for Turkish

Chapter 5

Experiments with the Tagger

Throughout the development of tagger we made various tests on different texts, and developed a database of approximately 250 constraint and hundreds of multi-word construct specifications. Some of these constraints are very general (e.g. the disambiguation of postpositions) while some are geared towards recognition of noun phrases of various sorts and the rest apply certain syntactic heuristics and our biased-preferences. The multi-word construct specifications contain the examples given in chapter 4 and lots of proper noun specifications.

We have performed some preliminary experiments with small texts to assess the effectiveness of our tagger, and to fine-tune the constraint specifications along with their orderings. Although the texts that we have experimented with are rather small, the results are encouraging in the sense that our approach is effective in disambiguating morphological structures and hence POS with 97-98% accuracy and with minimal (1.0%) user intervention[14].

At a final reasonable state of our specifications, we tested the performance of tagger with three small texts, which are articles from different newspapers, and an additional larger text, which is a document on Anatolian archeology with about 7000 words. Table 5.1 presents statistics of morphosyntactic ambiguity distribution in the texts, and it is observable that almost 70 to 80% of the sample texts contain ambiguous words.

Table 5.2 presents morphological disambiguation and tagging results. It is evident that the tagger provided a reliable result, and tagged the texts between 98.4% to 99.1% accuracy with very little user intervention. Among those correctly tagged words almost 96.7% to 98.5% of them are tagged automatically

Table 5.1. Statistics on texts tagged.

Text	Words	Morphological Parse Distribution					
		0	1	2	3	4	≥ 5
1	468	7.3%	28.7%	41.1%	11.1%	7.1%	4.7%
2	533	3.8%	24.8%	38.1%	19.1%	9.2%	5.0%
3	573	1.0%	30.2%	37.3%	13.1%	11.1%	7.3%
4	7004	3.9%	17.2%	41.5%	15.6%	11.7%	10.1%

Note: Words with zero parses are the ones which are not in the lexicon of the morphological analyzer, and they are mostly proper nouns. If their specification is available, they are tagged as proper nouns by the multi-word construct processor.

Table 5.2. Tagging and disambiguation results.

Text size	Correctly Tagged	Automatically Tagged	Disambiguation by		
			Multi-word rules	Constraints	User
468	99.1%	98.5%	10.1%	67.7%	0.6%
533	98.9%	97.8%	7.5%	74.5%	1.1%
573	98.8%	98.5%	3.1%	74.4%	0.3%
7004	98.4%	96.7%	4.2%	75.9%	1.7%

Note: Disambiguated by user means the tagger couldn't resolve the ambiguity.

by the tagger and the remaining 0.3% to 1.7% of the words are left ambiguous to be resolved by user. This user interaction is optional as stated before, and we turned this option off for these tests to see whether the remaining ambiguous words still contain the correct readings, or the correct readings are discarded by some constraints. The results are very encouraging in the sense that, even though 0.9 to 1.6% of the texts are tagged with a wrong reading, all of the remaining ambiguous words still contain the legitimate reading.

Currently, the speed of the tagger is limited by essentially that of the morphological analyzer, but the morphological analyzer has been ported to the XEROX TWOL system developed by Karttunen and Beesley [9]. This system which can analyze Turkish word forms at about 1000 forms/sec on Sun SparcStation 10's. We intend to integrate this to our tagger soon, improving its speed performance considerably. The constraint checking mechanism does not create a bottleneck with this amount of constraints. However, if the constraint size increases in the multiples of 10's, a more efficient constraint evaluation

mechanism will be required.

5.1 Impact of Morphological Disambiguation on Parsing Performance

We have tested the impact of morphological disambiguation on the performance of an LFG parser developed for Turkish [6, 7]. The input text fed to the parser was disambiguated using the tool developed and the results were compared to the case when the parser had to consider all possible morphological ambiguities. For a set of 80 sentences considered we obtained the results, shown in Table 5.3.

Table 5.3. Impact of disambiguation on parsing performance.

	No disambiguation		With disambiguation		Ratios	
Avg. Length of sentences	Avg. parses	Avg. time (sec)	Avg. parses	Avg. time (sec)	parses	speed-up
5.7	5.78	29.11	3.30	11.91	1.97	2.38

Note: The ratios are the averages of the sentence by sentence ratios.

It can be seen that, morphological disambiguation enables almost a factor of two reduction in the average number of parses generated and over a factor of two speed-up in time, which is encouraging in the sense that the higher level analysis of Turkish text will benefit from the functionality output of the tagger.

Chapter 6

Conclusions and Future Work

In this thesis, we have presented the design and implementation of a part-of-speech tagger for Turkish text along with various issues that come up in disambiguating among morphological parses of Turkish words. This is the first effort in tagging Turkish text including morphological disambiguation. In languages like Turkish or Finnish, with heavily inflected agglutinative morphology, morphological disambiguation is a crucial process in tagging since the structure of many lexical forms are morphologically ambiguous.

In the literature, we can observe two major paradigms for building part-of-speech (POS) taggers, rule-based approaches and statistical approaches. In this work, we prefered to use a rule based approach where one can write a set of constraint rules to discard all and only the contextually illegitimate alternative morphological readings. This functionality is complemented with a rule-based multi-word and idiomatic construct recognizer to detect multiple lexical items that act as single syntactic or semantic entity and to coalesce them into a single lexical form with a unique POS. The tool also provides additional functionalities, like statistics compilation, for fine-tuning of the morphological analyzer and the tagger itself.

Our constraint database contains almost 250 constraint specifications. Some of these constraints are very general as the disambiguation of postpositions, while some are geared towards recognition of noun phrases of various sorts and the rest apply certain syntactic heuristics and our biased-preferences. The multi-word construct recognition database contains hundreds of specifications where some of them are for the recognition of specific word patterns, like *yapar yapmaz, gelir gelmez* and *koşa koşa, zıplaya zıplaya*, and the rest is for

the recognition of lots of proper nouns, like *Mustafa Kemal Atatürk*.

We have performed several experiments to assess disambiguation performance of our tagger and noted that the use of constraints is very effective in morphological disambiguation. In these experiment the tagger proved 98.4 to 99.1% accuracy with very minimal user intervention, i.e., 96.7 to 98.5% of the texts are tagged automatically and remaining 0.3 to 1.7% of the texts are left ambiguous.

Furthermore, we have performed another experiment to assess the impact of morphological disambiguation to higher level analysis of texts. For this purpose, a set of 80 sentences have been disambiguated by the tagger and an LFG parser for Turkish is fed with the disambiguated sentences. The LFG parser , on the average, generated 50% less ambiguous parses and parsed almost 2.5 times faster.

The current constraint based disambiguation formalism, beside its reliable performance, has certain limitations.

- Constraint rules are applied in the given linear order and rule crafting requires heavy effort for reasonable ordering.
- Immediate application of the actions of constraint rules block other rules to evaluate different possibilities.
- This formalism can not capture the word-order freeness.

One solution to these problems might be to use a different inference mechanism where disambiguation decisions do not depend solely on the immediate application of the actions of constraint rules, instead each constraint rule vote for their preferences on matching words and the final global vote tally determines the assignments.

While experimenting with different texts, we have observed that there are quite many words which are not recognized by the morphological analyzer. Many of these words are proper nouns and if they are in our multi-word construct specifications they are classified. On the other hand, if we miss to recognize these unknown words performance of the tagger degrades. Hence, we need a mechanism to recognize unknown words by analyzing affixations of words.

Automatic rule acquisition is one of the open ended research topics in rule-based disambiguation, and this is the major weakness of rule-based methods against statistical ones. First attempt for automatic rule acquisition came from Brill [1]. In his work, Brill uses a new learning paradigm called transformation-based error-driven learning and gets reasonable performance. We also need to find a mechanism for automatic rule acqusition which will make the life easier.

Bibliography

[1] E. Brill. A simple rule-based part-of-speech tagger. In *Proceedings of the Third Conference on Applied Computational Linguistics*, Trento, Italy, 1992.

[2] K. W. Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing (ACL)*, pages 136–143, 1988.

[3] D. Cutting, J. Kupiec, J. Pedersen, and P. Sibun. A practical part-of-speech tagger. Technical report, Xerox Palo Alto Research Center, 1993.

[4] S. J. DeRose. Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14:31–39, 1988.

[5] B. B. Greene and G. M. Rubin. Automated grammatical tagging of English, 1971.

[6] Z. Güngördü. A Lexical-Functional Grammar for Turkish. Master's thesis, Department of Computer Engineering and Information Sciences, Bilkent University, Ankara, Turkey, July 1993.

[7] Z. Güngördü and K. Oflazer. Parsing Turkish using the lexical-functional grammar formalism. In *Proceedings of COLING-94, the 15th International Conference on Computational Linguistics*, Kyoto,Japan, 1994. To appear.

[8] F. Karlsson. Constraint grammar as a framework for parsing running text. In *Proceedings of COLING-90, the 13th International Conference on Computational Linguistics*, volume 3, pages 168–173, Helsinki, Finland, 1990.

[9] L. Karttunen and K. R. Beesley. Two-level rule compiler. Technical Report, XEROX Palo Alto Research Center, 1992.

[10] S. Klein and R. F. Simmons. Computational approach to grammatical coding of English words. *JACM*, 10:334–47, 1963.

[11] I. Marshall. Choice of grammatical word-class without global syntactic analysis: Tagging words in the LOB corpus. *Computers in Humanities*, 17:139–150, 1983.

[12] K. Oflazer. Two-level description of Turkish morphology. In *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics*, April 1993. A full version appears in *Literary and Linguistic Computing*, Vol.9 No.2, 1994.

[13] K. Oflazer and H. C. Bozşahin. Turkish Natural Language Processing Initiative: An overview. In *Proceedings of Third Turkish Symposium on Artificial Intelligence and Neural Networks*. Middle East Technical University, 1994.

[14] K. Oflazer and İ. Kuruöz. A tool for tagging Turkish text. In *Proceedings of Third Turkish Symposium on Artificial Intelligence and Neural Networks*. Middle East Technical University, 1994.

[15] A. Voutilainen. *Three studies of grammar-based surface parsing of unrestricted English text.* PhD thesis, Department of General Linguistics, University of Helsinki, Helsinki,Finland, May 1994.

Appendix A

Sample Tagged Output

A.1 Sample Text

(Milliyet 7.12.92 Olaylar ve İnsanlar Hasan Pulur)

Kadınlar çok şey istiyor... Kadınlar ne istiyor? Çok şey istiyorlar! Yalnız "çok şey istiyorlar!" dedik diye, "bu kadınlar da ipin ucunu kaçırdı!" gibi bir anlam çıkarılmamalı, eşitlik istiyorlar o kadar...

Kadınlar, 1926 tarihli Medeni Kanun'un, 1990'lı dünyanın gereklerine uydurulmasını istiyorlar... Elbette, 1926 tarihinde, Medeni Kanun, o yılların toplumsal ve kültürel anlayışının sonucu olarak kadın erkek eşitliğine dayalıydı, lakin o günden bugüne köprülerin altından, kadın haklarından, kadın-erkek eşitliğinden yana çok sular geçtiği için, Medeni Kanun'un da, diğer ülkelerin kanunları gibi değiştirilmesi gerekiyor...

Peki, kadınlar ne istiyorlar? İstanbul Üniversitesi Kadın Sorunları Araştırma ve Uygulama Merkezi, Medeni Kanun'da neler istediklerini gerekçeleriyle açıkladı...

Yürürlükteki Medeni Kanun'a göre, ailenin reisi kocadır, ev seçimini o yapar, nerede oturulacağına o karar verir, karısını ve çocuklarını o uygun bir biçimde geçindirir... Kadınların hazırladığı yeni tasarıda, eşler arasında eşitlik esas kabul edildiği için erkeğin reisliği kaldırılmış, oturulacak ev için, iki tarafın anlaşması gerekiyor, anlaşamıyorlarsa, hakime gidecekler... Şimdi erkekler diyecekler ki: "Aile reisliği gitti, ev seçme hakkımız da kalktı, peki, bu aileyi

kim geçindirecek?" Kadınlar ev geçindirmede de eşitlik istiyorlar... Şöyle diyorlar: "Eşlerden her biri evliliğin sorumluluğuna ve aile birliğinin ihtiyaçlarını karşılamasına güçleri oranındakatkıda bulunacaklardır." Şimdi davudi sesli erkek itirazlarını duyuyoruz: "Kadın, hangi gücü oranında ailenin ihtiyacını karşılayacak?" Kadın, dışarıda çalışıp para kazanmıyor ya! Peki, evde çalışan kadının aileye katkısı yok mudur? Hem de ne katkı?

Bir de, soyadı meselesi var! Erkek isterse, karısının, kadın da isterse, kocasının soyadını taşıyacak, ya da bekarlık soyadlarını da kullanabilecekler... Kadınlar kusura bakmasınlar ama, bu biraz ayrıntı, karı-koca ayrı ayrı soyadları taşıyacaklar, biraz garip değil mi, şekilcilik değil mi? Diyelim taşıdılar, ne olacak, temel sorunları çözülecek mi? Ama bir kadın isterse, erkek de uygun görürse, evlendikten sonra kızlık soyadını da kocasının soyadıyla birlikte taşıyabilmeli...

Şimdi gelelim en önemli maddeye... Siz ne derseniz deyin, mal canın yongasıdır... Yürürlükteki kanuna göre, bizde "mal ayrılığı" vardır. Yani kadının malı kadınındır, erkeğin malı erkeğin... Aslında ilk bakışta "mal ayrılığı" kadın-erkek eşitliğine uygun görülebilir. Fakat, Türkiye'deki uygulamada ev kadınlarının hakkı yenmektedir. Kadın kuruluşları "mal birliği" isteyerek şöyle demektedirler: "Mal ayrılığı, görünüşte, kadın erkek eşitliğine uygun bir rejimdir. Ancak uygulamada, özellikle ev kadını diye tanımlanan insanların durumunu ağırlaştırmaktadır. Milyonlarca kadın tarlada çalışarak veya evde en ağır işleri görerek yarattıkları artı değere sahip olamamaktadır. Evlilik dönemi elde edilen taşınmaz mallar, genellikle kocanın adına tapuya kaydolmakta ve gelirler kocanın banka hesabına geçirilmektedir. Evliliğin boşanma veya ölüm ile sona ermesi halinde, kadın ortada kalmaktadır. Kadının çabası her zaman gözle görülen bir kazanç veya gelir şeklinde ortaya çıkmayabilir. O nedenle, bugünkü düzen, sosyal adalet ve eşitlik ilkesine aykırıdır. Ayrıca, boşanma ve miras hukukunda yapılan son değişiklikler, kadın haklarının korunmasını daha da zorunlu kılmaktadır. Kaldı ki, eğer evlilik birliği esnasında alınan mallar kadının üzerine tapulanmışsa, o takdirde de, boşanma halinde erkek mağdur olabilmektedir. Bunun dışında, mal ayrılığı, hileli iflaslarda veya geri ödenmemesi durumlarında alacaklıları güç durumda bırakabilmektedir. Eşlere mal rejimi sözleşmesi yapma hakkı da tanınmıştır. Yeni kanun rejimi, eşlerin mal rejimi sözleşmesi ile kanunda belirtilen diğer rejimlerden birini seçemedikleri takdirde geçerlidir. Evlenmeden önce sahip olunan mallar, mal ayrılığı esasına tabidir. Evlenmeden sonra edinilen mallar için ortak katılım, kanuni rejim olarak kabul edilmiştir."

Kadınlar daha çok şey istiyor... Biz içlerinden bir-ikisini seçtik, diğerlerini öğrenmek istiyorsanız ve açtıkları imza kampanyasını desteklemek istiyorsanız, aşağıdaki telefon numarasına ve adrese başvurabilirsiniz. İlle de kadın olmanız şart değil... Biliyoruz, başta aile reisliği, çok şeylerden vazgeçmek erkekler için kolay değil... Değil ama, oturup anlatmak, uzlaşmak da var... İlle de vazgeçmek değil!

A.2 Tagged Output

The following is the tagged output of the previous sample text.[1]

```
- ( [1] :  ("(" ((*CAT* PUNCT)(*R* "(")))
- milliyet [2 CONS] :
    ("milliyet" ((*CAT* N)(*R* "milliyet")(*AGR* 3SG)(*CASE* NOM)))
- 7 [1] :
    ("7" ((*CAT* NUM)(*R* "7")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- 12 [1] :
    ("12" ((*CAT* NUM)(*R* "12")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- 92 [1] :
    ("92" ((*CAT* NUM)(*R* "92")))
- olaylar [3 CONS] :
    ("olay+lAr" ((*CAT* N)(*R* "olay")(*AGR* 3PL)(*CASE* NOM)))
- ve [1] :
    ("ve" ((*CAT* CON)(*R* "ve")))
- insanlar [3 CONS] :
    ("insan+lAr" ((*CAT* N)(*R* "insan")(*AGR* 3PL)(*CASE* NOM)))
- hasan pulur [RULE] :
    ((*CAT* N)(*R* "hasan pulur")(*SUB* PROP))
- ) [1] :  (")" ((*CAT* PUNCT)(*R* ")")))
- kadInlar [3 CONS] :
    ("kadIn+lAr" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3PL)(*CASE* NOM)))
- Cok [3 CONS] :
    ("Cok" ((*CAT* ADJ)(*R* "Cok")(*SUB* QTY-U)(*AGR* 3SG)(*CASE* NOM)))
- Sey [2 CONS] :
    ("Sey" ((*CAT* N)(*R* "Sey")(*AGR* 3SG)(*CASE* NOM)))
- istiyor [1] :
    ("iste+Hyor" ((*CAT* V)(*R* "iste")(*ASPECT* PR-CONT)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- kadInlar [3 CONS] :
    ("kadIn+lAr" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3PL)(*CASE* NOM)))
```

[1] Among those words, the ones tagged with an illegitimate POS are marked with "X", and the ones left ambiguous are marked with "#".

```
- ne [4 CONS] :
    ("neY" ((*CAT* ADV)(*R* "ne")(*SUB* QUES)))
- istiyor [1] :
    ("iste+Hyor" ((*CAT* V)(*R* "iste")(*ASPECT* PR-CONT)(*AGR* 3SG)))
- ?  [1] :  ("?" ((*CAT* PUNCT)(*R* "?")))
- Cok [3 CONS] :
    ("Cok" ((*CAT* ADJ)(*R* "Cok")(*SUB* QTY-U)(*AGR* 3SG)(*CASE* NOM)))
- Sey [2 CONS] :
    ("Sey" ((*CAT* N)(*R* "Sey")(*AGR* 3SG)(*CASE* NOM)))
- istiyorlar [1] :
    ("iste+Hyor+lAr" ((*CAT* V)(*R* "iste")(*ASPECT* PR-CONT)(*AGR* 3PL)))
- !  [1] :  ("!" ((*CAT* PUNCT)(*R* "!")))
- yalnIz [4 CONS] :
    ("yalnIz" ((*CAT* CON)(*R* "yalnIz")))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- Cok [3 CONS] :
    ("Cok" ((*CAT* ADJ)(*R* "Cok")(*SUB* QTY-U)(*AGR* 3SG)(*CASE* NOM)))
- Sey [2 CONS] :
    ("Sey" ((*CAT* N)(*R* "Sey")(*AGR* 3SG)(*CASE* NOM)))
- istiyorlar [1] :
    ("iste+Hyor+lAr" ((*CAT* V)(*R* "iste")(*ASPECT* PR-CONT)(*AGR* 3PL)))
- !  [1] :  ("!" ((*CAT* PUNCT)(*R* "!")))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- dedik [1] :
    ("de+DH+k" ((*CAT* V)(*R* "de")(*ASPECT* PAST)(*AGR* 1PL)))
- diye [2 CONS] :
    ("diye" ((*CAT* POSTP)(*R* "diye")))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- bu [3 CONS] :
    ("bu" ((*CAT* ADJ)(*R* "bu")(*AGR* 3SG)(*SUB* DEMO)))
- kadInlar [3 CONS] :
    ("kadIn+lAr" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3PL)(*CASE* NOM)))
- da [1] :
    ("da" ((*CAT* CON)(*R* "de")))
- ipin [4 CONS] :
    ("ip+nHn" ((*CAT* N)(*R* "ip")(*AGR* 3SG)(*CASE* GEN)))
- ucunu [2 CONS] :
    ("uC+sH+nH" ((*CAT* N)(*R* "uC")(*AGR* 3SG)(*POSS* 3SG)(*CASE* ACC)))
- kaCIrdI [3 CONS] :
    ("kaJ+Hr+yDH" ((*CAT* V)(*R* "kaC")(*ASPECT* AOR)(*TENSE* PAST)(*AGR* 3SG)))
- !  [1] :  ("!" ((*CAT* PUNCT)(*R* "!")))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- gibi [1] :
    ("gibi" ((*CAT* POSTP)(*R* "gibi")(*SUBCAT* NOM)))
- bir [2 CONS] :
    ("bir" ((*CAT* ADJ)(*R* "bir")(*SUB* NUM)(*VALUE* 1)(*AGR* 3SG)(*CASE* NOM)))
- anlam [2 CONS] :
    ("anlam" ((*CAT* N)(*R* "anlam")(*AGR* 3SG)(*CASE* NOM)))
- CIkarIlmamalI [1] :
    ("CIK+Ar+Hl+mA+mAlH" ((*CAT* V)(*R* "CIk")(*VOICE* CAUS)(*VOICE* PASS)
                          (*SENSE* NEG)(*ASPECT* NECES)(*AGR* 3SG)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
```

```
- eSitlik [2 CONS] :
    ("eSit+lHk" ((*CAT* ADJ)(*R* "eSit")(*SUB* QUAL)(*CONV* N "lik")(*AGR* 3SG)(*CASE* NOM)))
- istiyorlar [1] :
    ("iste+Hyor+lAr" ((*CAT* V)(*R* "iste")(*ASPECT* PR-CONT)(*AGR* 3PL)))
- o [4 CONS] :
    ("o" ((*CAT* PN)(*R* "o")(*AGR* 3SG)(*CASE* NOM)))
- kadar [3 CONS] :
    ("kadar" ((*CAT* N)(*R* "kadar")(*AGR* 3SG)(*CASE* NOM)(*CONV* V "")
              (*ASPECT* PR-CONT)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- kadInlar [3 CONS] :
    ("kadIn+lAr" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3PL)(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- 1926 [1] :
    ("1926" ((*CAT* NUM)(*R* "1926")))
- tarihli [2 CONS] :
    ("tarih+lH" ((*CAT* N)(*R* "tarih")(*AGR* 3SG)(*CONV* ADJ "li")(*SUB* QUAL)
                 (*AGR* 3SG)(*CASE* NOM)))
- medeni kanun'un [RULE] :
    ((*CAT* N)(*R* "medeni kanun")(*SUB* PROP)(*AGR* 3SG)(*CASE* GEN))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- 1990'lI [1] :
    ("1990'lI" ((*CAT* NUM)(*R* "1990'lI")))
- dUnyanIn [4 CONS] :
    ("dUnya+nHn" ((*CAT* N)(*R* "dUnya")(*AGR* 3SG)(*CASE* GEN)))
- gereklerine [4 CONS] :
    ("gerek+lAr+sH+nA" ((*CAT* N)(*R* "gerek")(*AGR* 3PL)(*POSS* 3SG)(*CASE* DAT)))
- uydurulmasInI [1] :
    ("uy+DHr+Hl+mA+sH+nH" ((*CAT* V)(*R* "uy")(*VOICE* CAUS)(*VOICE* PASS)
                           (*CONV* N "ma")(*AGR* 3SG)(*POSS* 3SG)(*CASE* ACC)))
- istiyorlar [1] :
    ("iste+Hyor+lAr" ((*CAT* V)(*R* "iste")(*ASPECT* PR-CONT)(*AGR* 3PL)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- elbette [1] :
    ("elbette" ((*CAT* ADV)(*R* " elbette")(*SUB* SENT)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- 1926 [1] :
    ("1926" ((*CAT* NUM)(*R* "1926")))
- tarihinde [6 CONS] :
    ("tarih+sH+nDA" ((*CAT* N)(*R* "tarih")(*AGR* 3SG)(*POSS* 3SG)(*CASE* LOC)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- medeni kanun [RULE] :
    ((*CAT* N)(*R* "medeni kanun")(*SUB* PROP)(*AGR* 3SG)(*CASE* NOM))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- o [4 CONS] :
    ("o" ((*CAT* ADJ)(*R* "o")(*AGR* 3SG)(*SUB* DEMO)))
- yIllarIn [4 CONS] :
    ("yIl+lAr+nHn" ((*CAT* N)(*R* "yIl")(*SUB* TEMP-UNIT)(*AGR* 3PL)(*CASE* GEN)))
- toplumsal [2 CONS] :
```

```
     ("toplumsal" ((*CAT* ADJ)(*R* "toplumsal")(*AGR* 3SG)(*CASE* NOM)))
- ve [1] :
     ("ve" ((*CAT* CON)(*R* "ve")))
- kUltUrel [2 CONS] :
     ("kUltUrel" ((*CAT* ADJ)(*R* "kUltUrel")(*AGR* 3SG)(*CASE* NOM)))
- anlayISInIn [8 CONS] :
     ("anlayIS+sH+nHn" ((*CAT* N)(*R* "anlayIS")(*AGR* 3SG)(*POSS* 3SG)(*CASE* GEN)))
- sonucu [3 CONS] :
     ("sonuC+sH" ((*CAT* N)(*R* "sonuC")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- olarak [1] :
     ("ol+yArAk" ((*CAT* V)(*R* "ol")(*SUBCAT* NOM)(*CONV* ADV "yarak")(*SUB* ATT)))
- kadIn [2 CONS] :
     ("kadIn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- erkek [2 CONS] :
     ("erkek" ((*CAT* N)(*R* "erkek")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- eSitliGine [2 CONS] :
     ("eSit+lHk+sH+nA" ((*CAT* ADJ)(*R* "eSit")(*SUB* QUAL)(*CONV* N "lik")(*AGR* 3SG)
                        (*POSS* 3SG)(*CASE* DAT)))
- dayalIydI [1] :
     ("dayalI+yDH" ((*CAT* ADJ)(*R* "dayalI")(*AGR* 3SG)(*CASE* NOM)(*CONV* V "")
                    (*ASPECT* PAST)(*AGR* 3SG)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- lakin [1] :
     ("lakin" ((*CAT* CON)(*R* "lakin")))
- o [4 CONS] :
     ("o" ((*CAT* ADJ)(*R* "o")(*AGR* 3SG)(*SUB* DEMO)))
- gUnden [1] :
     ("gUn+DAn" ((*CAT* N)(*R* "gUn")(*SUB* TEMP-UNIT)(*AGR* 3SG)(*CASE* ABL)))
- bugUne [1] :
     ("bugUn+yA" ((*CAT* N)(*R* "bugUn")(*SUB* TEMP)(*AGR* 3SG)(*CASE* DAT)))
- kOprUlerin [4 CONS] :
     ("kOprU+lAr+nHn" ((*CAT* N)(*R* "kOprU")(*AGR* 3PL)(*CASE* GEN)))
- altIndan [4 CONS] :
     ("alt+sH+nDAn" ((*CAT* N)(*R* "alt")(*ROLE* ADJ)(*AGR* 3SG)(*POSS* 3SG)(*CASE* ABL)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- kadIn [2 CONS] :
     ("kadIn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- haklarIndan [2 CONS] :
     ("haklar+sH+nDAn" ((*CAT* N)(*R* "hak")(*AGR* PL)(*POSS* 3SG)(*CASE* ABL)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- kadIn [2 CONS] :
     ("kadIn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- - [1] :  ("-" ((*CAT* PUNCT)(*R* "-")))
- erkek [2 CONS] :
     ("erkek" ((*CAT* N)(*R* "erkek")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- eSitliGinden [2 CONS] :
     ("eSit+lHk+sH+nDAn" ((*CAT* ADJ)(*R* "eSit")(*SUB* QUAL)(*CONV* N "lik")
                          (*AGR* 3SG)(*POSS* 3SG)(*CASE* ABL)))
- yana [3 CONS] :
     ("yana" ((*CAT* POSTP)(*R* "yana")(*SUBCAT* ABL)))
- Cok [3 CONS] :
     ("Cok" ((*CAT* ADJ)(*R* "Cok")(*SUB* QTY-U)(*AGR* 3SG)(*CASE* NOM)))
- sular [5 CONS] :
```

```
     ("sula+Hr" ((*CAT* V)(*R* "sula")(*CONV* ADJ "ir")))
- geCtiGi [2 CONS] :
     ("geJ+DHk+sH" ((*CAT* V)(*R* "geC")(*CONV* ADJ "dik")(*POSS* 3SG)(*CASE* NOM)))
- iCin [6 CONS] :
     ("iJ+nHn" ((*CAT* N)(*R* "iC")(*ROLE* ADJ)(*SUB* SPATIAL)(*AGR* 3SG)(*CASE* GEN)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- medeni kanun'un [RULE] :
     ((*CAT* N)(*R* "medeni kanun")(*SUB* PROP)(*CASE* GEN))
- da [1] :
     ("da" ((*CAT* CON)(*R* "de")))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- diGer [2 CONS] :
     ("diGer" ((*CAT* ADJ)(*R* "diGer")(*AGR* 3SG)(*CASE* NOM)))
- Ulkelerin [4 CONS] :
     ("Ulke+lAr+nHn" ((*CAT* N)(*R* "Ulke")(*AGR* 3PL)(*CASE* GEN)))
- kanunlarI [7 CONS] :
     ("kanun+lArH" ((*CAT* N)(*R* "kanun")(*AGR* 3SG)(*POSS* 3PL)(*CASE* NOM)))
- gibi [1] :
     ("gibi" ((*CAT* POSTP)(*R* "gibi")(*SUBCAT* NOM)))
- deGiStirilmesi [2 CONS] :
     ("deGiS+DHr+Hl+mA+sH" ((*CAT* V)(*R* "deGiS")(*VOICE* CAUS)(*VOICE* PASS)
                            (*CONV* N "ma")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- gerekiyor [1] :
     ("gereK+Hyor" ((*CAT* V)(*R* "gerek")(*ASPECT* PR-CONT)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- peki [1] :
     ("peki" ((*CAT* ADV)(*R* "peki")(*SUB* YANIT)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- kadInlar [3 CONS] :
     ("kadIn+lAr" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3PL)(*CASE* NOM)))
- ne [4 CONS] :
     ("neY" ((*CAT* ADV)(*R* "ne")(*SUB* QUES)))
- istiyorlar [1] :
     ("iste+Hyor+lAr" ((*CAT* V)(*R* "iste")(*ASPECT* PR-CONT)(*AGR* 3PL)))
- ?  [1] :  ("?" ((*CAT* PUNCT)(*R* "?")))
- istanbul Universitesi [RULE] :
     ((*CAT* N)(*R* "istanbul Universitesi")(*SUB* PROP)(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM))
- kadIn [2 CONS] :
     ("kadIn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- sorunlarI [7 CONS] :
     ("sorun+lAr+sH" ((*CAT* N)(*R* "sorun")(*AGR* 3PL)(*POSS* 3SG)(*CASE* NOM)))
- araStIrma [3 CONS] :
     ("araStIr+mA" ((*CAT* V)(*R* "araStIr")(*CONV* N "ma")(*AGR* 3SG)(*CASE* NOM)))
- ve [1] :
     ("ve" ((*CAT* CON)(*R* "ve")))
- uygulama [5 CONS] :
     ("uygulama" ((*CAT* N)(*R* "uygulama")(*AGR* 3SG)(*CASE* NOM)))
- merkezi [5 CONS] :
     ("merkez+sH" ((*CAT* N)(*R* "merkez")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- medeni kanun'da [RULE] :
```

```
     ((*CAT* N)(*R* "medeni kanun")(*SUB* PROP)(*AGR* 3SG)(*CASE* LOC))
- neler [3 CONS] :
     ("neY+lAr" ((*CAT* ADJ)(*R* "ne")(*SUB* QUES)(*AGR* 3PL)(*CASE* NOM)))
- istediklerini [4 CONS] :
     ("iste+DHk+lAr+sH+nH" ((*CAT* V)(*R* "iste")(*CONV* ADJ "dik")(*AGR* 3PL)
                            (*POSS* 3SG)(*CASE* ACC)))
- gerekCeleriyle [3 CONS] :
     ("gerekCe+lAr+sH+ylA" ((*CAT* N)(*R* "gerekCe")(*AGR* 3PL)(*POSS* 3SG)(*CASE* INS)))
- aCIkladI [1] :
     ("aCIkla+DH" ((*CAT* V)(*R* "aCIkla")(*ASPECT* PAST)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- yUrUrlUkteki [2 CONS] :
     ("yUrUrlUk+DA+ki" ((*CAT* N)(*R* "yUrUrlUk")(*AGR* 3SG)(*CASE* LOC)
                        (*CONV* ADJ "ki")(*AGR* 3SG)(*CASE* NOM)))
- medeni kanun'a [RULE] :
     ((*CAT* N)(*R* "medeni kanun")(*SUB* PROP)(*AGR* 3SG)(*CASE* DAT))
- gOre [2 CONS] :
     ("gOre" ((*CAT* POSTP)(*R* "gOre")(*SUBCAT* DAT)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- ailenin [4 CONS] :
     ("aile+nHn" ((*CAT* N)(*R* "aile")(*AGR* 3SG)(*CASE* GEN)))
- reisi [3 CONS] :
     ("reis+sH" ((*CAT* N)(*R* "reis")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- kocadIr [2 CONS] :
     ("koca+DHr" ((*CAT* N)(*R* "koca")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)
                  (*CONV* V "")(*ASPECT* PR-CONT)(*AGR* 3SG)(*MISC* COPU)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- ev [2 CONS] :
     ("ev" ((*CAT* N)(*R* "ev")(*AGR* 3SG)(*CASE* NOM)))
- seCimini [2 CONS] :
     ("seCim+sH+nH" ((*CAT* N)(*R* "seCim")(*AGR* 3SG)(*POSS* 3SG)(*CASE* ACC)))
- o [4 CONS] :
     ("o" ((*CAT* PN)(*R* "o")(*AGR* 3SG)(*CASE* NOM)))
- yapar [2 CONS] :
     ("yap+Ar" ((*CAT* V)(*R* "yap")(*CONV* ADJ "ir")))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- nerede [2 CONS] :
     ("nere+DA" ((*CAT* PN)(*R* "nere")(*SUB *QUES)(*AGR* 3SG)(*CASE* LOC)))
- oturulacaGIna [2 CONS] :
     ("otur+Hl+yAcAk+sH+nA" ((*CAT* V)(*R* "otur")(*VOICE* PASS)(*CONV* ADJ "yacak")
                             (*AGR* 3SG)(*POSS* 3SG)(*CASE* DAT)))
X o [4 CONS] :
     ("o" ((*CAT* ADJ)(*R* "o")(*AGR* 3SG)(*SUB* DEMO)))
- karar [7 CONS] :
     ("karar" ((*CAT* N)(*R* "karar")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- verir [2 CONS] :
     ("ver+Hr" ((*CAT* V)(*R* "ver")(*CONV* ADJ "ir")))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- karIsInI [1] :
     ("karI+sH+nH" ((*CAT* N)(*R* "karI")(*AGR* 3SG)(*POSS* 3SG)(*CASE* ACC)))
- ve [1] :
```

```
     ("ve" ((*CAT* CON)(*R* "ve")))
- CocuklarInI [4 CONS] :
     ("Cocuk+lAr+sH+nH" ((*CAT* N)(*R* "Cocuk")(*AGR* 3PL)(*POSS* 3SG)(*CASE* ACC)))
- o [4 CONS] :
     ("o" ((*CAT* PN)(*R* "o")(*AGR* 3SG)(*CASE* NOM)))
- uygun [4 CONS] :
     ("uygun" ((*CAT* ADJ)(*R* "uygun")(*AGR* 3SG)(*CASE* NOM)))
- bir [2 CONS] :
     ("bir" ((*CAT* ADJ)(*R* "bir")(*SUB* NUM)(*VALUE* 1)(*AGR* 3SG)(*CASE* NOM)))
- biCimde [2 CONS] :
     ("biCim+DA" ((*CAT* N)(*R* "biCim")(*AGR* 3SG)(*CASE* LOC)))
- geCindirir [2 CONS] :
     ("geCin+DHr+Hr" ((*CAT* V)(*R* "geCin")(*VOICE* CAUS)(*ASPECT* AOR)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- kadInlarIn [4 CONS] :
     ("kadIn+lAr+nHn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3PL)(*CASE* GEN)))
- hazIrladIGI [2 CONS] :
     ("hazIrla+DHk+sH" ((*CAT* V)(*R* "hazIrla")(*CONV* ADJ "dik")(*POSS* 3SG)(*CASE* NOM)))
X yeni [5 CONS] :
     ("yen+sH" ((*CAT* N)(*R* "yen")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- tasarIda [2 CONS] :
     ("tasarI+DA" ((*CAT* N)(*R* "tasarI")(*AGR* 3SG)(*CASE* LOC)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
X eSler [5 CONS] :
     ("eSle+Hr" ((*CAT* V)(*R* "eSle")(*CONV* ADJ "ir")))
- arasInda [3 CONS] :
     ("ara+sH+nDA" ((*CAT* N)(*R* "ara")(*ROLE* ADJ)(*SUB* TEMP-POINT)(*SUB* SPATIAL)
                    (*AGR* 3SG)(*POSS* 3SG)(*CASE* LOC)))
- eSitlik [2 CONS] :
     ("eSit+lHk" ((*CAT* ADJ)(*R* "eSit")(*SUB* QUAL)(*CONV* N "lik")(*AGR* 3SG)(*CASE* NOM)))
- esas [2 CONS] :
     ("esas" ((*CAT* N)(*R* "esas")(*AGR* 3SG)(*CASE* NOM)))
- kabul edildiGi [RULE] :
     ((*CAT* V)(*R* "hak et")(*VOICE* PASS)(*POSS* 3SG)(*CASE* NOM))
- iCin [6 CONS] :
     ("iCin" ((*CAT* POSTP)(*R* "iCin")(*SUBCAT* NOM)))
- erkeGin [4 CONS] :
     ("erkek+nHn" ((*CAT* N)(*R* "erkek")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* GEN)))
- reisliGi [3 CONS] :
     ("reislik+sH" ((*CAT* N)(*R* "reislik")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- kaldIrIlmIS [3 CONS] :
     ("kaldIr+Hl+mHS" ((*CAT* V)(*R* "kaldIr")(*VOICE* PASS)(*CONV* ADJ "mis")
                       (*AGR* 3SG)(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- oturulacak [3 CONS] :
     ("otur+Hl+yAcAk" ((*CAT* V)(*R* "otur")(*VOICE* PASS)(*CONV* ADJ "yacak")
                       (*AGR* 3SG)(*CASE* NOM)))
- ev [2 CONS] :
     ("ev" ((*CAT* N)(*R* "ev")(*AGR* 3SG)(*CASE* NOM)))
- iCin [6 CONS] :
     ("iCin" ((*CAT* POSTP)(*R* "iCin")(*SUBCAT* NOM)))
```

```
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- iki [2 CONS] :
     ("iki" ((*CAT* ADJ)(*R* "iki")(*SUB* NUM)(*VALUE* 2)(*AGR* 3SG)(*CASE* NOM)))
- tarafIn [4 CONS] :
     ("taraf+nHn" ((*CAT* N)(*R* "taraf")(*AGR* 3SG)(*CASE* GEN)))
- anlaSmasI [4 CONS] :
     ("anlaSma+sH" ((*CAT* N)(*R* "anlaSma")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- gerekiyor [1] :
     ("gereK+Hyor" ((*CAT* V)(*R* "gerek")(*ASPECT* PR-CONT)(*AGR* 3SG)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- anlaSamIyorlarsa [1] :
     ("anlaS+yAmA+Hyor+lAr+ysA" ((*CAT* V)(*R* "anlaS") (*SENSE* NEGC)
                                 (*ASPECT* PR-CONT)(*AGR* 3PL)(*TENSE* COND)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- hakime [1] :
     ("hakim+yA" ((*CAT* N)(*R* "hakim")(*AGR* 3SG)(*CASE* DAT)))
- gidecekler [4 CONS] :
     ("gid+yAcAk+lAr" ((*CAT* V)(*R* "git")(*ASPECT* FUT)(*AGR* 3PL)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- Simdi [1] :
     ("Simdi" ((*CAT* ADV)(*R* "Simdi")(*SUB* TEMP)))
- erkekler [3 CONS] :
     ("erkek+lAr" ((*CAT* N)(*R* "erkek")(*ROLE* ADJ)(*AGR* 3PL)(*CASE* NOM)))
X diyecekler [4 CONS] :
     ("diyecek+lAr" ((*CAT* V)(*R* "de")(*CONV* ADJ "yacak")(*AGR* 3PL)(*CASE* NOM)))
- ki [1] :
     ("ki" ((*CAT* CON)(*R* "ki")))
- :  [1] :  (":" ((*CAT* PUNCT)(*R* ":")))
- " [1] :  ("""" ((*CAT* PUNCT)(*R* """")))
- aile [2 CONS] :
     ("aile" ((*CAT* N)(*R* "aile")(*AGR* 3SG)(*CASE* NOM)))
- reisliGi [3 CONS] :
     ("reislik+sH" ((*CAT* N)(*R* "reislik")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- gitti [1] :
     ("gid+DH" ((*CAT* V)(*R* "git")(*ASPECT* PAST)(*AGR* 3SG)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- ev [2 CONS] :
     ("ev" ((*CAT* N)(*R* "ev")(*AGR* 3SG)(*CASE* NOM)))
- seCme [3 CONS] :
     ("seJ+mA" ((*CAT* V)(*R* "seC")(*CONV* N "ma")(*AGR* 3SG)(*CASE* NOM)))
- hakkImIz [3 CONS] :
     ("hakkImIz" ((*CAT* N)(*R* "hak")(*POSS* 1 PL)(*CASE* NOM)))
- da [1] :
     ("da" ((*CAT* CON)(*R* "de")))
- kalktI [1] :
     ("kalK+DH" ((*CAT* V)(*R* "kalk")(*ASPECT* PAST)(*AGR* 3SG)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- peki [1] :
     ("peki" ((*CAT* ADV)(*R* "peki")(*SUB* YANIT)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- bu [3 CONS] :
```

```
     ("bu" ((*CAT* ADJ)(*R* "bu")(*AGR* 3SG)(*SUB* DEMO)))
- aileyi [1] :
     ("aile+yH" ((*CAT* N)(*R* "aile")(*AGR* 3SG)(*CASE* ACC)))
- kim [2 CONS] :
     ("kim" ((*CAT* PN)(*R* "kim")(*SUB* QUES)(*AGR* 3SG)(*CASE* NOM)))
- geCindirecek [3 CONS] :
     ("geCin+DHr+yAcAk" ((*CAT* V)(*R* "geCin")(*VOICE* CAUS)(*ASPECT* FUT)(*AGR* 3SG)))
- ?  [1] :  ("?" ((*CAT* PUNCT)(*R* "?")))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- kadInlar [3 CONS] :
     ("kadIn+lAr" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3PL)(*CASE* NOM)))
- ev [2 CONS] :
     ("ev" ((*CAT* N)(*R* "ev")(*AGR* 3SG)(*CASE* NOM)))
- geCindirmede [3 CONS] :
     ("geCin+DHr+mA+DA" ((*CAT* V)(*R* "geCin")(*VOICE* CAUS)(*CONV* N "ma")
                         (*AGR* 3SG)(*CASE* LOC)))
- de [2 CONS] :
     ("de" ((*CAT* CON)(*R* "de")))
- eSitlik [2 CONS] :
     ("eSit+lHk" ((*CAT* ADJ)(*R* "eSit")(*SUB* QUAL)(*CONV* N "lik")(*AGR* 3SG)(*CASE* NOM)))
- istiyorlar [1] :
     ("iste+Hyor+lAr" ((*CAT* V)(*R* "iste")(*ASPECT* PR-CONT)(*AGR* 3PL)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- SOyle [4 CONS] :
     ("SOyle" ((*CAT* ADV)(*R* "SOyle")))
- diyorlar [1] :
     ("de+Hyor+lAr" ((*CAT* V)(*R* "de")(*ASPECT* PR-CONT)(*AGR* 3PL)))
- :  [1] :  (":" ((*CAT* PUNCT)(*R* ":")))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- eSlerden [1] :
     ("eS+lAr+DAn" ((*CAT* N)(*R* "eS")(*AGR* 3PL)(*CASE* ABL)))
- her [1] :
     ("her" ((*CAT* ADJ)(*R* "her")(*SUB* ADJ-ONLY)))
- biri [5 CONS] :
     ("bir+sH" ((*CAT* ADJ)(*R* "bir")(*SUB* NUM)(*VALUE* 1)(*AGR* 3SG)
                (*POSS* 3SG)(*CASE* NOM)))
- evliliGin [12 CONS] :
     ("ev+lH+lHk+nHn" ((*CAT* N)(*R* "ev")(*AGR* 3SG)(*CONV* ADJ "li")(*SUB* QUAL)
                       (*CONV* N "lik")(*AGR* 3SG)(*CASE* GEN)))
- sorumluluGuna [2 CONS] :
     ("sorumluluk+sH+nA" ((*CAT* N)(*R* "sorumluluk")(*AGR* 3SG)(*POSS* 3SG)(*CASE* DAT)))
- ve [1] :
     ("ve" ((*CAT* CON)(*R* "ve")))
- aile [2 CONS] :
     ("aile" ((*CAT* N)(*R* "aile")(*AGR* 3SG)(*CASE* NOM)))
- birliGinin [8 CONS] :
     ("birlik+sH+nHn" ((*CAT* N)(*R* "birlik")(*AGR* 3SG)(*POSS* 3SG)(*CASE* GEN)))
- ihtiyaClarInI [4 CONS] :
     ("ihtiyaC+lAr+sH+nH" ((*CAT* N)(*R* "ihtiyaC")(*AGR* 3PL)(*POSS* 3SG)(*CASE* ACC)))
- karSIlamasIna [1] :
     ("karSIla+mA+sH+nA" ((*CAT* V)(*R* "karSIla")(*CONV* N "ma")(*AGR* 3SG)
```

```
                        (*POSS* 3SG)(*CASE* DAT)))
- gUCleri [7 CONS] :
    ("gUC+lArH" ((*CAT* N)(*R* "gUC")(*AGR* 3PL)(*POSS* 3PL)(*CASE* NOM)))
- oranInda [4 CONS] :
    ("oran+sH+nDA" ((*CAT* N)(*R* "oran")(*AGR* 3SG)(*POSS* 3SG)(*CASE* LOC)))
- katkIda [2 CONS] :
    ("katkI+DA" ((*CAT* N)(*R* "katkI")(*AGR* 3SG)(*CASE* LOC)))
- bulunacaklardIr [8 CONS] :
    ("bulun+yAcAk+lAr+DHr" ((*CAT* V)(*R* "bulun")(*SUBCAT* LOC)(*CONV* ADJ "yacak")
                            (*AGR* 3PL)(*CASE* NOM)(*CONV* V "")(*ASPECT* PR-CONT)
                            (*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- Simdi [1] :
    ("Simdi" ((*CAT* ADV)(*R* "Simdi")(*SUB* TEMP)))
- davudi [2 CONS] :
    ("davudi" ((*CAT* ADJ)(*R* "davudi")(*AGR* 3SG)(*CASE* NOM)))
- sesli [2 CONS] :
    ("ses+lH" ((*CAT* N)(*R* "ses")(*AGR* 3SG)(*CONV* ADJ "li")(*SUB* QUAL)
               (*AGR* 3SG)(*CASE* NOM)))
- erkek [2 CONS] :
    ("erkek" ((*CAT* N)(*R* "erkek")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- itirazlarInI [4 CONS] :
    ("itiraz+lAr+sH+nH" ((*CAT* N)(*R* "itiraz")(*AGR* 3PL)(*POSS* 3SG)(*CASE* ACC)))
- duyuyoruz [1] :
    ("duy+Hyor+yHz" ((*CAT* V)(*R* "duy")(*ASPECT* PR-CONT)(*AGR* 1PL)))
- :  [1] :  (":" ((*CAT* PUNCT)(*R* ":")))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- kadIn [2 CONS] :
    ("kadIn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- hangi [2 CONS] :
    ("hangi" ((*CAT* ADJ)(*R* "hangi")(*SUB* QUES)(*AGR* 3SG)(*CASE* NOM)))
X gUcU [3 CONS] :
    ("gUC+yH" ((*CAT* N)(*R* "gUC")(*AGR* 3SG)(*CASE* ACC)))
- oranInda [4 CONS] :
    ("oran+sH+nDA" ((*CAT* N)(*R* "oran")(*AGR* 3SG)(*POSS* 3SG)(*CASE* LOC)))
- ailenin [4 CONS] :
    ("aile+nHn" ((*CAT* N)(*R* "aile")(*AGR* 3SG)(*CASE* GEN)))
- ihtiyacInI [2 CONS] :
    ("ihtiyaC+sH+nH" ((*CAT* N)(*R* "ihtiyaC")(*AGR* 3SG)(*POSS* 3SG)(*CASE* ACC)))
- karSIlayacak [3 CONS] :
    ("karSIla+yAcAk" ((*CAT* V)(*R* "karSIla")(*ASPECT* FUT)(*AGR* 3SG)))
- ?  [1] :  ("?" ((*CAT* PUNCT)(*R* "?")))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- kadIn [2 CONS] :
    ("kadIn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- dISarIda [2 CONS] :
    ("dISarI+DA" ((*CAT* N)(*R* "dISarI")(*SEMCASE* DAT)(*AGR* 3SG)(*CASE* LOC)))
- CalISIp [1] :
    ("CalIS+yHp" ((*CAT* V)(*R* "CalIS")(*CONV* ADV "yip")(*SUB* TEMP)))
- para [2 CONS] :
```

```
     ("para" ((*CAT* N)(*R* "para")(*AGR* 3SG)(*CASE* NOM)))
- kazanmIyor [1] :
     ("kazan+mA+Hyor" ((*CAT* V)(*R* "kazan")(*SENSE* NEG)(*ASPECT* PR-CONT)(*AGR* 3SG)))
- ya [2 CONS] :
     ("ya" ((*CAT* EXC)(*R* "ya")))
- !  [1] :  ("!" ((*CAT* PUNCT)(*R* "!")))
- peki [1] :
     ("peki" ((*CAT* ADV)(*R* "peki")(*SUB* YANIT)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- evde [2 CONS] :
     ("ev+DA" ((*CAT* N)(*R* "ev")(*AGR* 3SG)(*CASE* LOC)))
- CalISan [2 CONS] :
     ("CalIS+yAn" ((*CAT* V)(*R* "CalIS")(*CONV* ADJ "yan")(*AGR* 3SG)(*CASE* NOM)))
- kadInIn [4 CONS] :
     ("kadIn+nHn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* GEN)))
- aileye [1] :
     ("aile+yA" ((*CAT* N)(*R* "aile")(*AGR* 3SG)(*CASE* DAT)))
- katkIsI [2 CONS] :
     ("katkI+sH" ((*CAT* N)(*R* "katkI")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- yok [3 CONS] :
     ("yok" ((*CAT* N)(*R* "yok")(*AGR* 3SG)(*CASE* NOM)))
- mudur [1] :
     ("mu+dHr" ((*CAT* QUES)(*R* "mi")(*MISC* COPU)))
- ?  [1] :  ("?" ((*CAT* PUNCT)(*R* "?")))
- hem [2 CONS] :
     ("hem" ((*CAT* CON)(*R* "hem")))
- de [2 CONS] :
     ("de" ((*CAT* CON)(*R* "de")))
- ne [4 CONS] :
     ("neY" ((*CAT* ADJ)(*R* "ne")(*SUB* QUES)(*AGR* 3SG)(*CASE* NOM)))
- katkI [2 CONS] :
     ("katkI" ((*CAT* N)(*R* "katkI")(*AGR* 3SG)(*CASE* NOM)))
- ?  [1] :  ("?" ((*CAT* PUNCT)(*R* "?")))
- bir [2 CONS] :
     ("bir" ((*CAT* ADJ)(*R* "bir")(*SUB* NUM)(*VALUE* 1)(*AGR* 3SG)(*CASE* NOM)))
- de [2 CONS] :
     ("de" ((*CAT* CON)(*R* "de")))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- soyadI [4 CONS] :
     ("soyaD+sH" ((*CAT* N)(*R* "soyadI")(*AGR* 3SG)(*CASE* NOM)))
- meselesi [2 CONS] :
     ("mesele+sH" ((*CAT* N)(*R* "mesele")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- var [3 CONS] :
     ("var" ((*CAT* ADJ)(*R* "var")(*AGR* 3SG)(*CASE* NOM)(*CONV* V "")
     (*ASPECT* PR-CONT)(*AGR* 3SG)))
- !  [1] :  ("!" ((*CAT* PUNCT)(*R* "!")))
- erkek [2 CONS] :
     ("erkek" ((*CAT* N)(*R* "erkek")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- isterse [1] :
     ("iste+Hr+ysA" ((*CAT* V)(*R* "iste")(*ASPECT* AOR)(*TENSE* COND)(*AGR* 3SG)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- karIsInIn [2 CONS] :
     ("karI+sH+nHn" ((*CAT* N)(*R* "karI")(*AGR* 3SG)(*POSS* 3SG)(*CASE* GEN)))
```

```
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- kadIn [2 CONS] :
    ("kadIn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- da [1] :
    ("da" ((*CAT* CON)(*R* "de")))
- isterse [1] :
    ("iste+Hr+ysA" ((*CAT* V)(*R* "iste")(*ASPECT* AOR)(*TENSE* COND)(*AGR* 3SG)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- kocasInIn [2 CONS] :
    ("koca+sH+nHn" ((*CAT* N)(*R* "koca")(*ROLE* ADJ)(*AGR* 3SG)(*POSS* 3SG)(*CASE* GEN)))
- soyadInI [3 CONS] :
    ("soyaD+sH+nH" ((*CAT* N)(*R* "soyadI")(*AGR* 3SG)(*POSS* 3SG)(*CASE* ACC)))
X taSIyacak [3 CONS] :
    ("taSI+yAcAk" ((*CAT* V)(*R* "taSI")(*SUBCAT* ACC)(*CONV* ADJ "yacak")
                   (*AGR* 3SG)(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- ya da [RULE] :
    ((*CAT* CON)(*R* "ya da"))
- bekarlIk [2 CONS] :
    ("bekarlIk" ((*CAT* N)(*R* "bekarlIk")(*AGR* 3SG)(*CASE* NOM)))
- soyadlarInI [4 CONS] :
    ("soyaD+lAr+sH+nH" ((*CAT* N)(*R* "soyadI")(*AGR* 3PL)(*POSS* 3SG)(*CASE* ACC)))
- da [1] :
    ("da" ((*CAT* CON)(*R* "de")))
- kullanabilecekler [4 CONS] :
    ("kullan+yAbil+yAcAk+lAr" ((*CAT* V)(*R* "kullan")(*COMP* "yabil")(*ASPECT* FUT)(*AGR* 3PL)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- kadInlar [3 CONS] :
    ("kadIn+lAr" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3PL)(*CASE* NOM)))
- kusura [1] :
    ("kusur+yA" ((*CAT* N)(*R* "kusur")(*AGR* 3SG)(*CASE* DAT)))
- bakmasInlar [1] :
    ("baK+mA+ZHnlAr" ((*CAT* V)(*R* "bak")(*SENSE* NEG)(*ASPECT* IMP)(*AGR* 3PL)))
- ama [3 CONS] :
    ("ama" ((*CAT* CON)(*R* "ama")))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- bu [3 CONS] :
    ("bu" ((*CAT* PN)(*R* "bu")(*AGR* 3SG)(*CASE* NOM)))
- biraz [3 CONS] :
    ("biraz" ((*CAT* ADJ)(*R* "biraz")(*SUB* QTY-U)(*AGR* 3SG)(*CASE* NOM)))
- ayrIntI [2 CONS] :
    ("ayrIntI" ((*CAT* N)(*R* "ayrIntI")(*AGR* 3SG)(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- karI [6 CONS] :
    ("karI" ((*CAT* N)(*R* "karI")(*AGR* 3SG)(*CASE* NOM)))
- - [1] :  ("-" ((*CAT* PUNCT)(*R* "-")))
- koca [3 CONS] :
    ("koca" ((*CAT* N)(*R* "koca")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- ayrI ayrI [RULE] :
    ((*R* "ayrI ayrI")(*CAT* ADV))
- soyadlarI [7 CONS] :
```

```
     ("soyaD+lArH" ((*CAT* N)(*R* "soyadI")(*AGR* 3PL)(*POSS* 3PL)(*CASE* NOM)))
- taSIyacaklar [4 CONS] :
     ("taSI+yAcAk+lAr" ((*CAT* V)(*R* "taSI")(*SUBCAT* ACC)(*CONV* ADJ "yacak")
                        (*AGR* 3PL)(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- biraz [3 CONS] :
     ("biraz" ((*CAT* ADJ)(*R* "biraz")(*SUB* QTY-U)(*AGR* 3SG)(*CASE* NOM)))
- garip [2 CONS] :
     ("garib" ((*CAT* ADJ)(*R* "garip")(*AGR* 3SG)(*CASE* NOM)))
- deGil [3 CONS] :
     ("deGil" ((*CAT* N)(*R* "deGil")(*AGR* 3SG)(*CASE* NOM)))
- mi [1] :
     ("mi" ((*CAT* QUES)(*R* "mi")))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- Sekilcilik [2 CONS] :
     ("Sekilcilik" ((*CAT* N)(*R* "Sekilcilik")(*AGR* 3SG)(*CASE* NOM)))
- deGil [3 CONS] :
     ("deGil" ((*CAT* N)(*R* "deGil")(*AGR* 3SG)(*CASE* NOM)))
- mi [1] :
     ("mi" ((*CAT* QUES)(*R* "mi")))
- ?  [1] :  ("?" ((*CAT* PUNCT)(*R* "?")))
- diyelim [1] :
     ("diye+lHm" ((*CAT* V)(*R* "de")(*ASPECT* OPT)(*AGR* 1PL)))
- taSIdIlar [1] :
     ("taSI+DH+lAr" ((*CAT* V)(*R* "taSI")(*SUBCAT* ACC)(*ASPECT* PAST)(*AGR* 3PL)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- ne [4 CONS] :
     ("ne" ((*CAT* CON)(*R* "ne")))
- olacak [3 CONS] :
     ("ol+yAcAk" ((*CAT* V)(*R* "ol")(*SUBCAT* NOM)(*CONV* ADJ "yacak")(*AGR* 3SG)(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- temel [2 CONS] :
     ("temel" ((*CAT* N)(*R* "temel")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- sorunlarI [7 CONS] :
     ("sorun+lAr+sH" ((*CAT* N)(*R* "sorun")(*AGR* 3PL)(*POSS* 3SG)(*CASE* NOM)))
- COzUlecek [3 CONS] :
     ("COz+Hl+yAcAk" ((*CAT* V)(*R* "COz")(*VOICE* PASS)(*CONV* ADJ "yacak")
                      (*AGR* 3SG)(*CASE* NOM)))
- mi [1] :
     ("mi" ((*CAT* QUES)(*R* "mi")))
- ?  [1] :  ("?" ((*CAT* PUNCT)(*R* "?")))
- ama [3 CONS] :
     ("ama" ((*CAT* CON)(*R* "ama")))
- bir [2 CONS] :
     ("bir" ((*CAT* ADJ)(*R* "bir")(*SUB* NUM)(*VALUE* 1)(*AGR* 3SG)(*CASE* NOM)))
- kadIn [2 CONS] :
     ("kadIn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- isterse [1] :
     ("iste+Hr+ysA" ((*CAT* V)(*R* "iste")(*ASPECT* AOR)(*TENSE* COND)(*AGR* 3SG)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- erkek [2 CONS] :
     ("erkek" ((*CAT* N)(*R* "erkek")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- de [2 CONS] :
```

```
     ("de" ((*CAT* CON)(*R* "de")))
- uygun [4 CONS] :
     ("uygun" ((*CAT* ADJ)(*R* "uygun")(*AGR* 3SG)(*CASE* NOM)))
- gOrUrse [1] :
     ("gOr+Hr+ysA" ((*CAT* V)(*R* "gOr")(*SUBCAT* ACC)(*ASPECT* AOR)(*TENSE* COND)(*AGR* 3SG)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- evlendikten [1] :
     ("evlen+DHktAn" ((*CAT* V)(*R* "evlen")(*CONV* ADJ "dik")(*CASE* ABL)))
- sonra [4 CONS] :
     ("sonra" ((*CAT* POSTP)(*R* "sonra")(*SUBCAT* ABL)))
- kIzlIk [2 CONS] :
     ("kIzlIk" ((*CAT* N)(*R* "kIzlIk")(*AGR* 3SG)(*CASE* NOM)))
- soyadInI [3 CONS] :
     ("soyaD+sH+nH" ((*CAT* N)(*R* "soyadI")(*AGR* 3SG)(*POSS* 3SG)(*CASE* ACC)))
- da [1] :
     ("da" ((*CAT* CON)(*R* "de")))
- kocasInIn [2 CONS] :
     ("koca+sH+nHn" ((*CAT* N)(*R* "koca")(*ROLE* ADJ)(*AGR* 3SG)(*POSS* 3SG)(*CASE* GEN)))
- soyadIyla [2 CONS] :
     ("soyaD+sH+ylA" ((*CAT* N)(*R* "soyadI")(*AGR* 3SG)(*POSS* 3SG)(*CASE* INS)))
- birlikte [6 CONS] :
     ("birlikte" ((*CAT* POSTP)(*R* "birlikte")(*SUBCAT* INS)))
- taSIyabilmeli [1] :
     ("taSI+yAbil+mAlH" ((*CAT* V)(*R* "taSI")(*SUBCAT* ACC)(*COMP* "yabil")
                         (*ASPECT* NECES)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- Simdi [1] :
     ("Simdi" ((*CAT* ADV)(*R* "Simdi")(*SUB* TEMP)))
- gelelim [3 CONS] :
     ("gele+lH+Hm" ((*CAT* N)(*R* "gele")(*AGR* 3SG)(*CONV* ADJ "li")(*SUB* QUAL)
                    (*AGR* 3SG)(*POSS* 1SG)(*CASE* NOM)))
- en [3 CONS] :
     ("en" ((*CAT* N)(*R* "en")(*AGR* 3SG)(*CASE* NOM)))
- Onemli [2 CONS] :
     ("Onem+lH" ((*CAT* N)(*R* "Onem")(*AGR* 3SG)(*CONV* ADJ "li")(*SUB* QUAL)
                 (*AGR* 3SG)(*CASE* NOM)))
- maddeye [1] :
     ("madde+yA" ((*CAT* N)(*R* "madde")(*AGR* 3SG)(*CASE* DAT)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- siz [2 CONS] :
     ("siz" ((*CAT* PN)(*R* "siz")(*AGR* 2PL)(*CASE* NOM)))
- ne [4 CONS] :
     ("neY" ((*CAT* ADV)(*R* "ne")(*SUB* QUES)))
- derseniz [2 CONS] :
     ("der+ZA+nHz" ((*CAT* V)(*R* "der")(*ASPECT* COND)(*AGR* 2PL)))
- deyin [1] :
     ("de+yHn" ((*CAT* V)(*R* "de")(*ASPECT* IMP)(*AGR* 2PL)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- mal [2 CONS] :
```

```
     ("mal" ((*CAT* N)(*R* "mal")(*AGR* 3SG)(*CASE* NOM)))
- canIn [4 CONS] :
     ("can+nHn" ((*CAT* N)(*R* "can")(*AGR* 3SG)(*CASE* GEN)))
- yongasIdIr [2 CONS] :
     ("yonga+sH+DHr" ((*CAT* N)(*R* "yonga")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)
                      (*CONV* V "")(*ASPECT* PR-CONT)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- yUrUrlUkteki [2 CONS] :
     ("yUrUrlUk+DA+ki" ((*CAT* N)(*R* "yUrUrlUk")(*AGR* 3SG)(*CASE* LOC)
                        (*CONV* ADJ "ki")(*AGR* 3SG)(*CASE* NOM)))
- kanuna [1] :
     ("kanun+yA" ((*CAT* N)(*R* "kanun")(*AGR* 3SG)(*CASE* DAT)))
- gOre [2 CONS] :
     ("gOre" ((*CAT* POSTP)(*R* "gOre")(*SUBCAT* DAT)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- bizde [2 CONS] :
     ("biz+DA" ((*CAT* PN)(*R* "biz")(*AGR* 1PL)(*CASE* LOC)))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- mal [2 CONS] :
     ("mal" ((*CAT* N)(*R* "mal")(*AGR* 3SG)(*CASE* NOM)))
- ayrIlIGI [6 CONS] :
     ("ayrIlIk+sH" ((*CAT* N)(*R* "ayrIlIk")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- vardIr [3 CONS] :
     ("var+DHr" ((*CAT* ADJ)(*R* "var")(*AGR* 3SG)(*CASE* NOM)(*CONV* V "")
                 (*ASPECT* PR-CONT)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- yani [1] :
     ("yani" ((*CAT* CON)(*R* "yani")))
- kadInIn [4 CONS] :
     ("kadIn+nHn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* GEN)))
- malI [3 CONS] :
     ("mal+sH" ((*CAT* N)(*R* "mal")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- kadInIndIr [4 CONS] :
     ("kadIn+nHn+DHr" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* GEN)
                       (*CONV* V "")(*ASPECT* PR-CONT)(*AGR* 3SG)(*MISC* COPU)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- erkeGin [4 CONS] :
     ("erkek+nHn" ((*CAT* N)(*R* "erkek")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* GEN)))
- malI [3 CONS] :
     ("mal+sH" ((*CAT* N)(*R* "mal")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- erkeGin [4 CONS] :
     ("erkek+nHn" ((*CAT* N)(*R* "erkek")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* GEN)
                   (*CONV* V "")(*ASPECT* PR-CONT)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- aslInda [11 CONS] :
     ("aslInda" ((*CAT* ADV)(*R* "aslInda")(*SUB* SENT)))
- ilk [3 CONS] :
     ("ilK" ((*CAT* ADJ)(*R* "ilk")(*AGR* 3SG)(*CASE* NOM)))
```

```
- bakISta [2 CONS] :
    ("baK+yHS+DA" ((*CAT* V)(*R* "bak")(*CONV* N "yis")(*AGR* 3SG)(*CASE* LOC)))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- mal [2 CONS] :
    ("mal" ((*CAT* N)(*R* "mal")(*AGR* 3SG)(*CASE* NOM)))
- ayrIlIGI [6 CONS] :
    ("ayrIlIk+sH" ((*CAT* N)(*R* "ayrIlIk")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- kadIn [2 CONS] :
    ("kadIn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- - [1] :  ("-" ((*CAT* PUNCT)(*R* "-")))
- erkek [2 CONS] :
    ("erkek" ((*CAT* N)(*R* "erkek")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- eSitliGine [2 CONS] :
    ("eSit+lHk+sH+nA" ((*CAT* ADJ)(*R* "eSit")(*SUB* QUAL)(*CONV* N "lik")
                       (*AGR* 3SG)(*POSS* 3SG)(*CASE* DAT)))
- uygun [4 CONS] :
    ("uygun" ((*CAT* ADJ)(*R* "uygun")(*AGR* 3SG)(*CASE* NOM)))
- gOrUlebilir [2 CONS] :
    ("gOr+Hl+yAbil+Hr" ((*CAT* V)(*R* "gOr")(*SUBCAT* ACC)(*VOICE* PASS)
                        (*COMP* "yabil")(*ASPECT* AOR)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- fakat [1] :
    ("fakat" ((*CAT* CON)(*R* "fakat")))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- tUrkiye'deki [2 CONS] :
    ("tUrkiye'+DA+ki" ((*CAT* N)(*R* "tUrkiye")(*SUB* PROP)(*AGR* 3SG)(*CASE* LOC)
                       (*CONV* ADJ "ki")(*AGR* 3SG)(*CASE* NOM)))
- uygulamada [5 CONS] :
    ("uygulama+DA" ((*CAT* N)(*R* "uygulama")(*AGR* 3SG)(*CASE* LOC)))
- ev [2 CONS] :
    ("ev" ((*CAT* N)(*R* "ev")(*AGR* 3SG)(*CASE* NOM)))
- kadInlarInIn [8 CONS] :
    ("kadIn+lAr+sH+nHn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3PL)(*POSS* 3SG)(*CASE* GEN)))
- hakkI [3 CONS] :
    ("hakkI" ((*CAT* N)(*R* "hak")(*POSS* 3SG)(*CASE* NOM)))
- yenmektedir [2 CONS] :
    ("yen+mAktA+DHr" ((*CAT* V)(*R* "yen")(*ASPECT* PR-CONT)(*MISC* COPU)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- kadIn [2 CONS] :
    ("kadIn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- kuruluSlarI [14 CONS] :
    ("kuruluS+lAr+sH" ((*CAT* N)(*R* "kuruluS")(*AGR* 3PL)(*POSS* 3SG)(*CASE* NOM)))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- mal [2 CONS] :
    ("mal" ((*CAT* N)(*R* "mal")(*AGR* 3SG)(*CASE* NOM)))
- birliGi [6 CONS] :
    ("birlik+sH" ((*CAT* N)(*R* "birlik")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- isteyerek [1] :
    ("iste+yArAk" ((*CAT* V)(*R* "iste")(*CONV* ADV "yarak")(*SUB* ATT)))
- SOyle [4 CONS] :
    ("SOyle" ((*CAT* ADV)(*R* "SOyle")))
```

```
- demektedirler [1] :
     ("de+mAktA+DHr+lAr" ((*CAT* V)(*R* "de")(*ASPECT* PR-CONT)(*MISC* COPU)(*AGR* 3PL)))
- :  [1] :  (":" ((*CAT* PUNCT)(*R* ":")))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- mal [2 CONS] :
     ("mal" ((*CAT* N)(*R* "mal")(*AGR* 3SG)(*CASE* NOM)))
- ayrIlIGI [6 CONS] :
     ("ayrIlIk+sH" ((*CAT* N)(*R* "ayrIlIk")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- gOrUnUSte [2 CONS] :
     ("gOrUn+yHS+DA" ((*CAT* V)(*R* "gOrUn")(*CONV* N "yis")(*AGR* 3SG)(*CASE* LOC)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- kadIn [2 CONS] :
     ("kadIn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- erkek [2 CONS] :
     ("erkek" ((*CAT* N)(*R* "erkek")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- eSitliGine [2 CONS] :
     ("eSit+lHk+sH+nA" ((*CAT* ADJ)(*R* "eSit")(*SUB* QUAL)(*CONV* N "lik")
                        (*AGR* 3SG)(*POSS* 3SG)(*CASE* DAT)))
- uygun [4 CONS] :
     ("uygun" ((*CAT* ADJ)(*R* "uygun")(*AGR* 3SG)(*CASE* NOM)))
- bir [2 CONS] :
     ("bir" ((*CAT* ADJ)(*R* "bir")(*SUB* NUM)(*VALUE* 1)(*AGR* 3SG)(*CASE* NOM)))
- rejimdir [4 CONS] :
     ("reji+Hm+DHr" ((*CAT* N)(*R* "reji")(*AGR* 3SG)(*POSS* 1SG)(*CASE* NOM)
                     (*CONV* V "")(*ASPECT* PR-CONT)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- ancak [2 CONS] :
     ("ancak" ((*CAT* CON)(*R* "ancak")))
# uygulamada [5 AMBIGUOUS] :
     ("uygulama+DA" ((*CAT* N)(*R* "uygulama")(*AGR* 3SG)(*CASE* LOC)))
     ("uygula+mA+DA" ((*CAT* V)(*R* "uygula")(*CONV* N "ma")(*AGR* 3SG)(*CASE* LOC)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- Ozellikle [3 CONS] :
     ("Ozellikle" ((*CAT* ADV)(*R* " Ozellikle")(*SUB* SENT)))
- ev [2 CONS] :
     ("ev" ((*CAT* N)(*R* "ev")(*AGR* 3SG)(*CASE* NOM)))
- kadInI [3 CONS] :
     ("kadIn+sH" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- diye [2 CONS] :
     ("diye" ((*CAT* POSTP)(*R* "diye")))
- tanImlanan [2 CONS] :
     ("tanImla+n+yAn" ((*CAT* V)(*R* "tanImla")(*VOICE* PASS)(*CONV* ADJ "yan")
                       (*AGR* 3SG)(*CASE* NOM)))
- insanlarIn [4 CONS] :
     ("insan+lAr+nHn" ((*CAT* N)(*R* "insan")(*AGR* 3PL)(*CASE* GEN)))
- durumunu [2 CONS] :
     ("durum+sH+nH" ((*CAT* N)(*R* "durum")(*AGR* 3SG)(*POSS* 3SG)(*CASE* ACC)))
- aGIrlaStIrmaktadIr [1] :
     ("aGIr+lAS+DHr+mAktA+DHr" ((*CAT* ADJ)(*R* "aGIr")(*SUB* QUAL)(*CONV* V "las")
                                (*VOICE* CAUS)(*ASPECT* PR-CONT)(*MISC* COPU)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- milyonlarca [2 CONS] :
```

```
     ("milyonlarca" ((*CAT* ADJ)(*R* "milyonlarca")(*SUB* QTY-U)(*AGR* 3SG)(*CASE* NOM)))
- kadIn [2 CONS] :
     ("kadIn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- tarlada [2 CONS] :
     ("tarla+DA" ((*CAT* N)(*R* "tarla")(*AGR* 3SG)(*CASE* LOC)))
- CalISarak [1] :
     ("CalIS+yArAk" ((*CAT* V)(*R* "CalIS")(*CONV* ADV "yarak")(*SUB* ATT)))
- veya [1] :
     ("veya" ((*CAT* CON)(*R* "veya")))
- evde [2 CONS] :
     ("ev+DA" ((*CAT* N)(*R* "ev")(*AGR* 3SG)(*CASE* LOC)))
- en [3 CONS] :
     ("en" ((*CAT* ADV)(*R* "en")(*SUB* SUPERLATIVE)))
- aGIr [4 CONS] :
     ("aG+Hr" ((*CAT* V)(*R* "aG")(*CONV* ADJ "ir")))
- iSleri [7 CONS] :
     ("iS+lAr+yH" ((*CAT* N)(*R* "iS")(*AGR* 3PL)(*CASE* ACC)))
- gOrerek [1] :
     ("gOr+yArAk" ((*CAT* V)(*R* "gOr")(*SUBCAT* ACC)(*CONV* ADV "yarak")(*SUB* ATT)))
- yarattIklarI [12 CONS] :
     ("yara+t+DHk+lAr+sH" ((*CAT* V)(*R* "yara")(*VOICE* CAUS)(*CONV* ADJ "dik")
                           (*AGR* 3PL)(*POSS* 3SG)(*CASE* NOM)))
- artI [2 CONS] :
     ("artI" ((*CAT* N)(*R* "artI")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- deGere [1] :
     ("deGer+yA" ((*CAT* N)(*R* "deGer")(*AGR* 3SG)(*CASE* DAT)))
- sahip [2 CONS] :
     ("sahib" ((*CAT* N)(*R* "sahip")(*AGR* 3SG)(*CASE* NOM)))
- olamamaktadIr [1] :
     ("ol+yAmA+mAktA+DHr" ((*CAT* V)(*R* "ol")(*SUBCAT* NOM)(*SENSE* NEGC)
                           (*ASPECT* PR-CONT)(*MISC* COPU)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- evlilik [6 CONS] :
     ("ev+lH+lHk" ((*CAT* N)(*R* "ev")(*AGR* 3SG)(*CONV* ADJ "li")(*SUB* QUAL)
                   (*CONV* N "lik")(*AGR* 3SG)(*CASE* NOM)))
- dOnemi [3 CONS] :
     ("dOnem+sH" ((*CAT* N)(*R* "dOnem")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- elde [2 CONS] :
     ("el+DA" ((*CAT* N)(*R* "el")(*AGR* 3SG)(*CASE* LOC)))
- edilen [2 CONS] :
     ("ed+Hl+yAn" ((*CAT* V)(*R* "et")(*VOICE* PASS)(*CONV* ADJ "yan")(*AGR* 3SG)(*CASE* NOM)))
- taSInmaz [3 CONS] :
     ("taSInmaz" ((*CAT* N)(*R* "taSInmaz")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- mallar [3 CONS] :
     ("mal+lAr" ((*CAT* N)(*R* "mal")(*AGR* 3PL)(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- genellikle [3 CONS] :
     ("genellikle" ((*CAT* CON)(*R* "genellikle")))
- kocanIn [5 CONS] :
     ("koca+nHn" ((*CAT* N)(*R* "koca")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* GEN)))
- adIna [2 CONS] :
     ("adI+nA" ((*CAT* N)(*R* "ad/name")(*POSS* 3SG)(*CASE* DAT)))
- tapuya [1] :
```

```
     ("tapu+yA" ((*CAT* N)(*R* "tapu")(*AGR* 3SG)(*CASE* DAT)))
- kaydolmakta [2 CONS] :
     ("kaydol+mAk+DA" ((*CAT* V)(*R* "kaydol")(*CONV* INF "mak")(*CASE* LOC)))
- ve [1] :
     ("ve" ((*CAT* CON)(*R* "ve")))
- gelirler [4 CONS] :
     ("gelir+lAr" ((*CAT* N)(*R* "gelir")(*AGR* 3PL)(*CASE* NOM)))
- kocanIn [5 CONS] :
     ("koca+nHn" ((*CAT* N)(*R* "koca")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* GEN)))
- banka [3 CONS] :
     ("banka" ((*CAT* N)(*R* "banka")(*AGR* 3SG)(*CASE* NOM)))
- hesabIna [2 CONS] :
     ("hesab+sH+nA" ((*CAT* N)(*R* "hesap")(*AGR* 3SG)(*POSS* 3SG)(*CASE* DAT)))
- geCirilmektedir [1] :
     ("geJ+Hr+Hl+mAktA+DHr" ((*CAT* V)(*R* "geC")(*VOICE* CAUS)(*VOICE* PASS)
                            (*ASPECT* PR-CONT)(*MISC* COPU)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
# evliliGin [12 AMBIGUOUS] :
     ("ev+lH+lHk+nHn" ((*CAT* N)(*R* "ev")(*AGR* 3SG)(*CONV* ADJ "li")(*SUB* QUAL)
                       (*CONV* N "lik")(*AGR* 3SG)(*CASE* GEN)))
     ("evlilik+nHn" ((*CAT* N)(*R* "evlilik")(*CASE* GEN)))
     ("evli+lHk+nHn" ((*CAT* ADJ)(*R* "evli")(*CONV* N "lik")(*AGR* 3SG)(*CASE* GEN)))
- boSanma [3 CONS] :
     ("boSa+n+mA" ((*CAT* V)(*R* "boSa")(*VOICE* PASS)(*CONV* N "ma")(*AGR* 3SG)(*CASE* NOM)))
- veya [1] :
     ("veya" ((*CAT* CON)(*R* "veya")))
- OlUm [4 CONS] :
     ("OlU+Hm" ((*CAT* N)(*R* "OlU")(*ROLE* ADJ)(*AGR* 3SG)(*POSS* 1SG)(*CASE* NOM)))
- ile [3 CONS] :
     ("il+yA" ((*CAT* N)(*R* "il")(*AGR* 3SG)(*CASE* DAT)))
- sona [2 CONS] :
     ("son+yA" ((*CAT* ADJ)(*R* "son")(*ROLE* ADJ-ONLY)(*AGR* 3SG)(*CASE* DAT)))
- ermesi [2 CONS] :
     ("er+mA+sH" ((*CAT* V)(*R* "er")(*CONV* N "ma")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- halinde [5 CONS] :
     ("halinde" ((*CAT* POSTP)(*R* "halinde")(*SUBCAT* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- kadIn [2 CONS] :
     ("kadIn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- ortada [2 CONS] :
     ("orta+DA" ((*CAT* N)(*R* "orta")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* LOC)))
- kalmaktadIr [1] :
     ("kal+mAktA+DHr" ((*CAT* V)(*R* "kal")(*ASPECT* PR-CONT)(*MISC* COPU)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- kadInIn [4 CONS] :
     ("kadIn+nHn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* GEN)))
- CabasI [2 CONS] :
     ("Caba+sH" ((*CAT* N)(*R* "Caba")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- her [1] :
     ("her" ((*CAT* ADJ)(*R* "her")(*SUB* ADJ-ONLY)))
- zaman [2 CONS] :
     ("zaman" ((*CAT* N)(*R* "zaman")(*SUB* TEMP-POINT)(*AGR* 3SG)(*CASE* NOM)))
- gOzle [2 CONS] :
```

```
     ("gOz+ylA" ((*CAT* N)(*R* "gOz")(*AGR* 3SG)(*CASE* INS)))
- gOrUlen [2 CONS] :
     ("gOr+Hl+yAn" ((*CAT* V)(*R* "gOr")(*SUBCAT* ACC)(*VOICE* PASS)(*CONV* ADJ "yan")
                    (*AGR* 3SG)(*CASE* NOM)))
- bir [2 CONS] :
     ("bir" ((*CAT* ADJ)(*R* "bir")(*SUB* NUM)(*VALUE* 1)(*AGR* 3SG)(*CASE* NOM)))
- kazanC [2 CONS] :
     ("kazanC" ((*CAT* N)(*R* "kazanC")(*AGR* 3SG)(*CASE* NOM)))
- veya [1] :
     ("veya" ((*CAT* CON)(*R* "veya")))
- gelir [4 CONS] :
     ("gelir" ((*CAT* N)(*R* "gelir")(*AGR* 3SG)(*CASE* NOM)))
- Seklinde [4 CONS] :
     ("Sek$il+sH+nDA" ((*CAT* N)(*R* "Sekil")(*AGR* 3SG)(*POSS* 3SG)(*CASE* LOC)))
- ortaya [1] :
     ("orta+yA" ((*CAT* N)(*R* "orta")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* DAT)))
- CIkmayabilir [2 CONS] :
     ("CIK+mA+yAbil+Hr" ((*CAT* V)(*R* "CIk")(*SENSE* NEG)(*COMP* "yabil")
                         (*ASPECT* AOR)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- o [4 CONS] :
     ("o" ((*CAT* ADJ)(*R* "o")(*AGR* 3SG)(*SUB* DEMO)))
- nedenle [1] :
     ("neden+ylA" ((*CAT* N)(*R* "neden")(*AGR* 3SG)(*CASE* INS)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- bugUnkU [2 CONS] :
     ("bugUnkU" ((*CAT* N)(*R* "bugUN ")(*SUB* TEMP)(*CONV* ADJ "ki")(*AGR* 3SG)(*CASE* NOM)))
- dUzen [5 CONS] :
     ("dUzen" ((*CAT* N)(*R* "dUzen")(*AGR* 3SG)(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- sosyal [2 CONS] :
     ("sosy- adalet [2 CONS] :
     ("adalet" ((*CAT* N)(*R* "adalet")(*AGR* 3SG)(*CASE* NOM)))
- ve [1] :
     ("ve" ((*CAT* CON)(*R* "ve")))
- eSitlik [2 CONS] :
     ("eSit+lHk" ((*CAT* ADJ)(*R* "eSit")(*SUB* QUAL)(*CONV* N "lik")(*AGR* 3SG)(*CASE* NOM)))
- ilkesine [1] :
     ("ilke+sH+nA" ((*CAT* N)(*R* "ilke")(*AGR* 3SG)(*POSS* 3SG)(*CASE* DAT)))
- aykIrIdIr [2 CONS] :
     ("aykIrI+DHr" ((*CAT* ADJ)(*R* "aykIrI")(*AGR* 3SG)(*CASE* NOM)(*CONV* V "")
                    (*ASPECT* PR-CONT)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- ayrIca [1] :
     ("ayrIca" ((*CAT* ADV)(*R* "ayrIca")))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- boSanma [3 CONS] :
     ("boSa+n+mA" ((*CAT* V)(*R* "boSa")(*VOICE* PASS)(*CONV* N "ma")(*AGR* 3SG)(*CASE* NOM)))
- ve [1] :
     ("ve" ((*CAT* CON)(*R* "ve")))
- miras [2 CONS] :
     ("miras" ((*CAT* N)(*R* "miras")(*AGR* 3SG)(*CASE* NOM)))
- hukukunda [4 CONS] :
```

```
     ("hukuK+sH+nDA" ((*CAT* N)(*R* "hukuk")(*AGR* 3SG)(*POSS* 3SG)(*CASE* LOC)))
- yapIlan [3 CONS] :
     ("yap+Hl+yAn" ((*CAT* V)(*R* "yap")(*VOICE* PASS)(*CONV* ADJ "yan")(*AGR* 3SG)(*CASE* NOM)))
- son [4 CONS] :
     ("son" ((*CAT* ADJ)(*R* "son")(*ROLE* ADJ-ONLY)(*AGR* 3SG)(*CASE* NOM)))
- deGiSiklikler [3 CONS] :
     ("deGiSik+lHk+lAr" ((*CAT* ADJ)(*R* "deGiSik")(*SUB* QUAL)(*CONV* N "lik")
                         (*AGR* 3PL)(*CASE* NOM)))
- , [1] :  (",," ((*CAT* PUNCT)(*R* ",")))
- kadIn [2 CONS] :
     ("kadIn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- haklarInIn [4 CONS] :
     ("haklar+sH+nHn" ((*CAT* N)(*R* "hak")(*AGR* PL)(*POSS* 3SG)(*CASE* GEN)))
- korunmasInI [1] :
     ("koru+n+mA+sH+nH" ((*CAT* V)(*R* "koru")(*VOICE* PASS)(*CONV* N "ma")
                         (*AGR* 3SG)(*POSS* 3SG)(*CASE* ACC)))
- daha [1] :
     ("daha" ((*CAT* ADV)(*R* "daha")(*SUB* COMPARATIVE)(*SUB* TEMP)))
- da [1] :
     ("da" ((*CAT* CON)(*R* "de")))
- zorunlu [2 CONS] :
     ("zorunlu" ((*CAT* ADJ)(*R* "zorunlu")(*AGR* 3SG)(*CASE* NOM)))
- kIlmaktadIr [1] :
     ("kIl+mAktA+DHr" ((*CAT* V)(*R* "kIl")(*ASPECT* PR-CONT)(*MISC* COPU)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- " [1] :  ("""" ((*CAT* PUNCT)(*R* """")))
- kaldI [1] :
     ("kal+DH" ((*CAT* V)(*R* "kal")(*ASPECT* PAST)(*AGR* 3SG)))
- ki [1] :
     ("ki" ((*CAT* CON)(*R* "ki")))
- , [1] :  (",," ((*CAT* PUNCT)(*R* ",")))
- eGer [3 CONS] :
     ("eGer" ((*CAT* CON)(*R* "eGer")))
- evlilik [6 CONS] :
     ("ev+lH+lHk" ((*CAT* N)(*R* "ev")(*AGR* 3SG)(*CONV* ADJ "li")(*SUB* QUAL)
                   (*CONV* N "lik")(*AGR* 3SG)(*CASE* NOM)))
- birliGi [6 CONS] :
     ("birlik+sH" ((*CAT* N)(*R* "birlik")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- esnasInda [2 CONS] :
     ("esna+sH+nDA" ((*CAT* N)(*R* "esna")(*AGR* 3SG)(*POSS* 3SG)(*CASE* LOC)))
- alInan [4 CONS] :
     ("al+Hn+yAn" ((*CAT* V)(*R* "al")(*SUBCAT* ACC)(*VOICE* PASS)(*CONV* ADJ "yan")
                   (*AGR* 3SG)(*CASE* NOM)))
- mallar [3 CONS] :
     ("mal+lAr" ((*CAT* N)(*R* "mal")(*AGR* 3PL)(*CASE* NOM)))
- kadInIn [4 CONS] :
     ("kadIn+nHn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* GEN)))
- Uzerine [3 CONS] :
     ("Uzerine" ((*CAT* POSTP)(*R* "Uzerine")(*SUBCAT* GEN)))
- tapulanmISsa [2 CONS] :
     ("tapula+n+mHS+ysA" ((*CAT* V)(*R* "tapula")(*VOICE* PASS)(*ASPECT* NARR)
                          (*TENSE* COND)(*AGR* 3SG)))
- , [1] :  (",," ((*CAT* PUNCT)(*R* ",")))
```

```
- o [4 CONS] :
     ("o" ((*CAT* ADJ)(*R* "o")(*AGR* 3SG)(*SUB* DEMO)))
- takdirde [3 CONS] :
     ("takdir+DA" ((*CAT* N)(*R* "takdir")(*AGR* 3SG)(*CASE* LOC)))
- de [2 CONS] :
     ("de" ((*CAT* CON)(*R* "de")))
- , [1] :  (",," ((*CAT* PUNCT)(*R* ",")))
- boSanma [3 CONS] :
     ("boSa+n+mA" ((*CAT* V)(*R* "boSa")(*VOICE* PASS)(*CONV* N "ma")(*AGR* 3SG)(*CASE* NOM)))
- halinde [5 CONS] :
     ("halinde" ((*CAT* POSTP)(*R* "halinde")(*SUBCAT* NOM)))
- erkek [2 CONS] :
     ("erkek" ((*CAT* N)(*R* "erkek")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- maGdur [2 CONS] :
     ("maGdur" ((*CAT* ADJ)(*R* "maGdur")(*AGR* 3SG)(*CASE* NOM)))
- olabilmektedir [1] :
     ("ol+yAbil+mAktA+DHr" ((*CAT* V)(*R* "ol")(*SUBCAT* NOM)(*COMP* "yabil")
                            (*ASPECT* PR-CONT)(*MISC* COPU)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- bunun [3 CONS] :
     ("bu+nHn" ((*CAT* PN)(*R* "bu")(*AGR* 3SG)(*CASE* GEN)))
- dISInda [4 CONS] :
     ("dIS+sH+nDA" ((*CAT* N)(*R* "dIS")(*ROLE* ADJ)(*SUB* SPATIAL)(*AGR* 3SG)
                    (*POSS* 3SG)(*CASE* LOC)))
- , [1] :  (",," ((*CAT* PUNCT)(*R* ",")))
- mal [2 CONS] :
     ("mal" ((*CAT* N)(*R* "mal")(*AGR* 3SG)(*CASE* NOM)))
- ayrIlIGI [6 CONS] :
     ("ayrIlIk+sH" ((*CAT* N)(*R* "ayrIlIk")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- , [1] :  (",," ((*CAT* PUNCT)(*R* ",")))
- hileli [2 CONS] :
     ("hile+lH" ((*CAT* N)(*R* "hile")(*AGR* 3SG)(*CONV* ADJ "li")(*SUB* QUAL)
                 (*AGR* 3SG)(*CASE* NOM)))
- iflaslarda [2 CONS] :
     ("iflas+lAr+DA" ((*CAT* ADJ)(*R* "iflas")(*AGR* 3PL)(*CASE* LOC)))
- veya [1] :
     ("veya" ((*CAT* CON)(*R* "veya")))
- geri [3 CONS] :
     ("geri" ((*CAT* N)(*R* "geri")(*AGR* 3SG)(*CASE* NOM)))
- Odenmemesi [2 CONS] :
     ("Ode+n+mA+mA+sH" ((*CAT* V)(*R* "Ode")(*VOICE* PASS)(*SENSE* NEG)
                        (*CONV* N "ma")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- durumlarInda [8 CONS] :
     ("durum+lAr+sH+nDA" ((*CAT* N)(*R* "durum")(*AGR* 3PL)(*POSS* 3SG)(*CASE* LOC)))
X alacaklIlarI [7 CONS] :
     ("alacak+lH+lAr+sH" ((*CAT* N)(*R* "alacak")(*AGR* 3SG)(*CONV* ADJ "li")
                          (*SUB* QUAL)(*AGR* 3PL)(*POSS* 3SG)(*CASE* NOM)))
- gUC [2 CONS] :
     ("gUC" ((*CAT* N)(*R* "gUC")(*AGR* 3SG)(*CASE* NOM)))
- durumda [4 CONS] :
     ("durum+DA" ((*CAT* N)(*R* "durum")(*AGR* 3SG)(*CASE* LOC)))
- bIrakabilmektedir [1] :
     ("bIraK+yAbil+mAktA+DHr" ((*CAT* V)(*R* "bIrak")(*COMP* "yabil")
```

```
                              (*ASPECT* PR-CONT)(*MISC* COPU)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- eSlere [1] :
     ("eS+lAr+yA" ((*CAT* N)(*R* "eS")(*AGR* 3PL)(*CASE* DAT)))
- mal [2 CONS] :
     ("mal" ((*CAT* N)(*R* "mal")(*AGR* 3SG)(*CASE* NOM)))
- rejimi [4 CONS] :
     ("rejim+sH" ((*CAT* N)(*R* "rejim")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- sOzleSmesi [2 CONS] :
     ("sOzleSme+sH" ((*CAT* N)(*R* "sOzleSme")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- yapma [3 CONS] :
     ("yap+mA" ((*CAT* V)(*R* "yap")(*CONV* N "ma")(*AGR* 3SG)(*CASE* NOM)))
- hakkI [3 CONS] :
     ("hakkI" ((*CAT* N)(*R* "hak")(*POSS* 3SG)(*CASE* NOM)))
- da [1] :
     ("da" ((*CAT* CON)(*R* "de")))
- tanInmIStIr [6 CONS] :
     ("tanI+n+mHS+DHr" ((*CAT* V)(*R* "tanI")(*VOICE* PASS)(*ASPECT* NARR)(*MISC* COPU)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- yeni [5 CONS] :
     ("yeni" ((*CAT* ADJ)(*R* "yeni")(*AGR* 3SG)(*CASE* NOM)))
- kanun [2 CONS] :
     ("kanun" ((*CAT* N)(*R* "kanun")(*AGR* 3SG)(*CASE* NOM)))
- rejimi [4 CONS] :
     ("rejim+sH" ((*CAT* N)(*R* "rejim")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- eSlerin [4 CONS] :
     ("eS+lAr+nHn" ((*CAT* N)(*R* "eS")(*AGR* 3PL)(*CASE* GEN)))
- mal [2 CONS] :
     ("mal" ((*CAT* N)(*R* "mal")(*AGR* 3SG)(*CASE* NOM)))
- rejimi [4 CONS] :
     ("rejim+sH" ((*CAT* N)(*R* "rejim")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- sOzleSmesi [2 CONS] :
     ("sOzleSme+sH" ((*CAT* N)(*R* "sOzleSme")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- ile [3 CONS] :
     ("ile" ((*CAT* CON)(*R* "ile")))
- kanunda [2 CONS] :
     ("kanun+DA" ((*CAT* N)(*R* "kanun")(*AGR* 3SG)(*CASE* LOC)))
- belirtilen [2 CONS] :
     ("belir+t+Hl+yAn" ((*CAT* V)(*R* "belir")(*VOICE* CAUS)(*VOICE* PASS)
                        (*CONV* ADJ "yan")(*AGR* 3SG)(*CASE* NOM)))
- diGer [2 CONS] :
     ("diGer" ((*CAT* ADJ)(*R* "diGer")(*AGR* 3SG)(*CASE* NOM)))
- rejimlerden [1] :
     ("rejim+lAr+DAn" ((*CAT* N)(*R* "rejim")(*AGR* 3PL)(*CASE* ABL)))
- birini [3 CONS] :
     ("bir+sH+nH" ((*CAT* ADJ)(*R* "bir")(*SUB* NUM)(*VALUE* 1)(*AGR* 3SG)(*POSS* 3SG)(*CASE* ACC)))
- seCemedikleri [6 CONS] :
     ("seJ+yAmA+DHk+lAr+sH" ((*CAT* V)(*R* "seC")(*SENSE* NEGC)(*CONV* ADJ "dik")
                             (*AGR* 3PL)(*POSS* 3SG)(*CASE* NOM)))
- takdirde [3 CONS] :
     ("takdir+DA" ((*CAT* N)(*R* "takdir")(*AGR* 3SG)(*CASE* LOC)))
- geCerlidir [2 CONS] :
```

```
     ("geCer+lH+DHr" ((*CAT* ADJ)(*R* "geCer")(*AGR* 3SG)(*CONV* ADJ "li")(*SUB* QUAL)
                      (*AGR* 3SG)(*CASE* NOM)(*CONV* V "")(*ASPECT* PR-CONT)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- evlenmeden [2 CONS] :
     ("evlen+mA+DAn" ((*CAT* V)(*R* "evlen")(*CONV* N "ma")(*AGR* 3SG)(*CASE* ABL)))
- Once [4 CONS] :
     ("Once" ((*CAT* POSTP)(*R* "Once")(*SUBCAT* ABL)))
- sahip [2 CONS] :
     ("sahib" ((*CAT* N)(*R* "sahip")(*AGR* 3SG)(*CASE* NOM)))
- olunan [2 CONS] :
     ("ol+Hn+yAn" ((*CAT* V)(*R* "ol")(*SUBCAT* NOM)(*VOICE* PASS)
                   (*CONV* ADJ "yan")(*AGR* 3SG)(*CASE* NOM)))
- mallar [3 CONS] :
     ("mal+lAr" ((*CAT* N)(*R* "mal")(*AGR* 3PL)(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- mal [2 CONS] :
     ("mal" ((*CAT* N)(*R* "mal")(*AGR* 3SG)(*CASE* NOM)))
- ayrIlIGI [6 CONS] :
     ("ayrIlIk+sH" ((*CAT* N)(*R* "ayrIlIk")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- esasIna [2 CONS] :
     ("esas+sH+nA" ((*CAT* N)(*R* "esas")(*AGR* 3SG)(*POSS* 3SG)(*CASE* DAT)))
- tabidir [2 CONS] :
     ("tabi+DHr" ((*CAT* ADJ)(*R* "tabi")(*AGR* 3SG)(*CASE* NOM)(*CONV* V "")
                  (*ASPECT* PR-CONT)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- evlenmeden [2 CONS] :
     ("evlen+mA+DAn" ((*CAT* V)(*R* "evlen")(*CONV* N "ma")(*AGR* 3SG)(*CASE* ABL)))
- sonra [4 CONS] :
     ("sonra" ((*CAT* POSTP)(*R* "sonra")(*SUBCAT* ABL)))
- edinilen [2 CONS] :
     ("edin+Hl+yAn" ((*CAT* V)(*R* "edin")(*VOICE* PASS)(*CONV* ADJ "yan")
                     (*AGR* 3SG)(*CASE* NOM)))
- mallar [3 CONS] :
     ("mal+lAr" ((*CAT* N)(*R* "mal")(*AGR* 3PL)(*CASE* NOM)))
- iCin [6 CONS] :
     ("iCin" ((*CAT* POSTP)(*R* "iCin")(*SUBCAT* NOM)))
- ortak [2 CONS] :
     ("ortak" ((*CAT* N)(*R* "ortak")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- katIlIm [4 CONS] :
     ("katIlIm" ((*CAT* N)(*R* "katIlIm")(*AGR* 3SG)(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- kanuni [2 CONS] :
     ("kanuni" ((*CAT* ADJ)(*R* "kanuni")(*AGR* 3SG)(*CASE* NOM)))
- rejim [4 CONS] :
     ("rejim" ((*CAT* N)(*R* "rejim")(*AGR* 3SG)(*CASE* NOM)))
- olarak [1] :
     ("ol+yArAk" ((*CAT* V)(*R* "ol")(*SUBCAT* NOM)(*CONV* ADV "yarak")(*SUB* ATT)))
- kabul edilmiStir [RULE] :
     ((*CAT* V)(*R* "hak et")(*VOICE* PASS)(*AGR* 3SG)(*CASE* NOM)(*ASPECT* PR-CONT))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- " [1] :  (""" ((*CAT* PUNCT)(*R* """)))
- kadInlar [3 CONS] :
     ("kadIn+lAr" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3PL)(*CASE* NOM)))
```

```
- daha [1] :
    ("daha" ((*CAT* ADV)(*R* "daha")(*SUB* COMPARATIVE)(*SUB* TEMP)))
- Cok [3 CONS] :
    ("Cok" ((*CAT* ADJ)(*R* "Cok")(*SUB* QTY-U)(*AGR* 3SG)(*CASE* NOM)))
- Sey [2 CONS] :
    ("Sey" ((*CAT* N)(*R* "Sey")(*AGR* 3SG)(*CASE* NOM)))
- istiyor [1] :
    ("iste+Hyor" ((*CAT* V)(*R* "iste")(*ASPECT* PR-CONT)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- biz [2 CONS] :
    ("biz" ((*CAT* PN)(*R* "biz")(*AGR* 1PL)(*CASE* NOM)))
- iClerinden [4 CONS] :
    ("iJ+lArH+nDAn" ((*CAT* N)(*R* "iC")(*ROLE* ADJ)(*SUB* SPATIAL)
                     (*AGR* 3PL)(*POSS* 3PL)(*CASE* ABL)))
- bir [2 CONS] :
    ("bir" ((*CAT* ADJ)(*R* "bir")(*SUB* NUM)(*VALUE* 1)(*AGR* 3SG)(*CASE* NOM)))
- - [1] :  ("-" ((*CAT* PUNCT)(*R* "-")))
- ikisini [1] :
    ("iki+sH+nH" ((*CAT* ADJ)(*R* "iki")(*SUB* NUM)(*VALUE* 2)(*AGR* 3SG)
                  (*POSS* 3SG)(*CASE* ACC)))
- seCtik [1] :
    ("seJ+DH+k" ((*CAT* V)(*R* "seC")(*ASPECT* PAST)(*AGR* 1PL)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- diGerlerini [4 CONS] :
    ("diGer+lArH+nH" ((*CAT* ADJ)(*R* "diGer")(*AGR* 3PL)(*POSS* 3PL)(*CASE* ACC)))
- OGrenmek [2 CONS] :
    ("OGren+mAk" ((*CAT* V)(*R* "OGren")(*SUBCAT* ACC)(*CONV* INF "mak")(*CASE* NOM)))
- istiyorsanIz [1] :
    ("iste+Hyor+ysA+nHz" ((*CAT* V)(*R* "iste")(*ASPECT* PR-CONT)(*TENSE* COND)(*AGR* 2PL)))
- ve [1] :
    ("ve" ((*CAT* CON)(*R* "ve")))
- aCtIklarI [6 CONS] :
    ("aJ+DHk+lAr+sH" ((*CAT* V)(*R* "aC")(*SUBCAT* ACC)(*CONV* ADJ "dik")
                      (*AGR* 3PL)(*POSS* 3SG)(*CASE* NOM)))
- imza [2 CONS] :
    ("imza" ((*CAT* N)(*R* "imza")(*AGR* 3SG)(*CASE* NOM)))
- kampanyasInI [1] :
    ("kampanya+sH+nH" ((*CAT* N)(*R* "kampanya")(*AGR* 3SG)(*POSS* 3SG)(*CASE* ACC)))
- desteklemek [2 CONS] :
    ("destekle+mAk" ((*CAT* V)(*R* "destekle")(*CONV* INF "mak")(*CASE* NOM)))
- istiyorsanIz [1] :
    ("iste+Hyor+ysA+nHz" ((*CAT* V)(*R* "iste")(*ASPECT* PR-CONT)(*TENSE* COND)(*AGR* 2PL)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- aSaGIdaki [2 CONS] :
    ("aSaGI+DA+ki" ((*CAT* N)(*R* "aSaGI")(*SEMCASE* DAT)(*AGR* 3SG)(*CASE* LOC)
                    (*CONV* ADJ "ki")(*AGR* 3SG)(*CASE* NOM)))
- telefon [2 CONS] :
    ("telefon" ((*CAT* N)(*R* "telefon")(*AGR* 3SG)(*CASE* NOM)))
- numarasIna [1] :
    ("numara+sH+nA" ((*CAT* N)(*R* "numara")(*AGR* 3SG)(*POSS* 3SG)(*CASE* DAT)))
- ve [1] :
```

```
     ("ve" ((*CAT* CON)(*R* "ve")))
- adrese [1] :
     ("adres+yA" ((*CAT* N)(*R* "adres")(*AGR* 3SG)(*CASE* DAT)))
- baSvurabilirsiniz [1] :
     ("baSvur+yAbil+Hr+ZHnHz" ((*CAT* V)(*R* "baSvur")(*COMP* "yabil")(*ASPECT* AOR)(*AGR* 2PL)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- ille [2 CONS] :
     ("ille" ((*CAT* ADV)(*R* "ille")))
- de [2 CONS] :
     ("de" ((*CAT* CON)(*R* "de")))
- kadIn [2 CONS] :
     ("kadIn" ((*CAT* N)(*R* "kadIn")(*ROLE* ADJ)(*AGR* 3SG)(*CASE* NOM)))
- olmanIz [3 CONS] :
     ("ol+mA+HnHz" ((*CAT* V)(*R* "ol")(*SUBCAT* NOM)(*CONV* N "ma")(*AGR* 3SG)
                    (*POSS* 2PL)(*CASE* NOM)))
- Sart [2 CONS] :
     ("Sart" ((*CAT* N)(*R* "Sart")(*AGR* 3SG)(*CASE* NOM)))
- deGil [3 CONS] :
     ("deGil" ((*CAT* N)(*R* "deGil")(*AGR* 3SG)(*CASE* NOM)(*CONV* V "")
               (*ASPECT* PR-CONT)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- biliyoruz [2 CONS] :
     ("bile+Hyor+yHz" ((*CAT* V)(*R* "bile")(*ASPECT* PR-CONT)(*AGR* 1PL)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- baSta [2 CONS] :
     ("baS+DA" ((*CAT* N)(*R* "baS")(*ROLE* MEASURE)(*AGR* 3SG)(*CASE* LOC)))
- aile [2 CONS] :
     ("aile" ((*CAT* N)(*R* "aile")(*AGR* 3SG)(*CASE* NOM)))
- reisliGi [3 CONS] :
     ("reislik+sH" ((*CAT* N)(*R* "reislik")(*AGR* 3SG)(*POSS* 3SG)(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- Cok [3 CONS] :
     ("Cok" ((*CAT* ADJ)(*R* "Cok")(*SUB* QTY-U)(*AGR* 3SG)(*CASE* NOM)))
- Seylerden [1] :
     ("Sey+lAr+DAn" ((*CAT* N)(*R* "Sey")(*AGR* 3PL)(*CASE* ABL)))
- vazgeCmek [2 CONS] :
     ("vazgeJ+mAk" ((*CAT* V)(*R* "vazgeC")(*CONV* INF "mak")(*CASE* NOM)))
- erkekler [3 CONS] :
     ("erkek+lAr" ((*CAT* N)(*R* "erkek")(*ROLE* ADJ)(*AGR* 3PL)(*CASE* NOM)))
- iCin [6 CONS] :
     ("iCin" ((*CAT* POSTP)(*R* "iCin")(*SUBCAT* NOM)))
- kolay [2 CONS] :
     ("kolay" ((*CAT* ADJ)(*R* "kolay")(*AGR* 3SG)(*CASE* NOM)))
- deGil [3 CONS] :
     ("deGil" ((*CAT* N)(*R* "deGil")(*AGR* 3SG)(*CASE* NOM)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- deGil [3 CONS] :
     ("deGil" ((*CAT* N)(*R* "deGil")(*AGR* 3SG)(*CASE* NOM)))
- ama [3 CONS] :
```

```
     ("ama" ((*CAT* CON)(*R* "ama")))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- oturup [1] :
     ("otur+yHp" ((*CAT* V)(*R* "otur")(*CONV* ADV "yip")(*SUB* TEMP)))
- anlatmak [2 CONS] :
     ("anlat+mAk" ((*CAT* V)(*R* "anlat") (*CONV* INF "mak")(*CASE* NOM)))
- , [1] :  ("," ((*CAT* PUNCT)(*R* ",")))
- uzlaSmak [2 CONS] :
     ("uzlaS+mAk" ((*CAT* V)(*R* "uzlaS")(*CONV* INF "mak")(*CASE* NOM)))
- da [1] :
     ("da" ((*CAT* CON)(*R* "de")))
- var [3 CONS] :
     ("var" ((*CAT* ADJ)(*R* "var")(*AGR* 3SG)(*CASE* NOM)(*CONV* V "")
             (*ASPECT* PR-CONT)(*AGR* 3SG)))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- .  [1] :  ("." ((*CAT* PUNCT)(*R* ".")))
- ille [2 CONS] :
     ("ille" ((*CAT* ADV)(*R* "ille")))
- de [2 CONS] :
     ("de" ((*CAT* CON)(*R* "de")))
- vazgeCmek [2 CONS] :
     ("vazgeJ+mAk" ((*CAT* V)(*R* "vazgeC")(*CONV* INF "mak")(*CASE* NOM)))
- deGil [3 CONS] :
     ("deGil" ((*CAT* N)(*R* "deGil")(*AGR* 3SG)(*CASE* NOM)(*CONV* V "")
              (*ASPECT* PR-CONT)(*AGR* 3SG)))
- !  [1] :  ("!" ((*CAT* PUNCT)(*R* "!")))
```

Appendix B

Sample Specifications

B.1 Multi-word Construct Specifications

```
# Duplicated optative and 3SG,  koSa koSa .
Label = R1p;
R=_R1, CAT = V, ASPECT = OPT, AGR = 3SG;
R = _R1, CAT = V, ASPECT = OPT, AGR = 3SG:
Compose = ((*R* "W1 W2")(*CAT* ADV)).


# Duplications with question suffix in between, gUzel mi gUzel.
Label = R2p;
R = _R1, CAT = ADJ, SUB = ?;
R = mi, CAT = QUES;
R = _R1, CAT = ADJ, SUB = ?:
Compose = ((*R* "W1 W2 W3")(*CAT* ADJ)).


# adjective + adjective,  mavi mavi.
Label = R3p;
R = _R1, CAT = ADJ, POSS = ~;
R = _R1, CAT = ADJ, POSS = ~:
Compose = ((*R* "W1 W2")(*CAT* ADV)).
```

```
# noun + noun,  ev ev.
Label = R4pa;
R = _R1, CAT = N, CASE = NOM;
R = _R1, CAT = N, CASE = NOM:
Compose = ((*R* "W1 W2")(*CAT* ADV)).

# noun + be + noun,  ev be ev.
Label = R4pb;
R = _R1, CAT = N, CASE = NOM;
R = be;
R = _R1, CAT = N, CASE = NOM:
Compose = ((*R* "W1 be W2")(*CAT* ADV)).

# aorist verbal construct,  yapar yapmaz.
Label = R5p;
R = _R1, CAT = V, ASPECT = AOR, AGR = 3SG;
R = _R1, CAT = V, ASPECT = AOR, AGR = 3SG, SENSE = NEG:
Compose = ((*R* "W1 W2")(*CAT* ADV)).

# Reflections from the nature, takIr takIr
Label = R6p;
CAT = DUP, R = _R1;
CAT = DUP, R = _R1:
Compose = ((*CAT* ADV)(*R* "W1 W2")).

# gelip gelmemiz, gelip gelmemesi
Label = R7p;
R = _R1, CAT = V, FINALCAT = ADV, CONV = "yip";
R = _R1, CAT = V, SENSE = NEG, FINALCAT=N, CONV="ma", POSS = ?:
Compose = ((*R* "W1 W2")(*CAT* N)).

# donemden doneme
Label = R7p;
R = _R1, CAT = N, AGR = 3SG, CASE = ABL;
R = _R1, CAT = N, AGR = 3SG, CASE = DAT:
Compose = ((*R* "W1 W2")(*CAT* ADV)).
```

```
# kaza yapmak
Label = R1v;
LEX = kaza, CASE = NOM;
R = yap, CAT = V:
Compose = ((*CAT* V)(*R* "kaza yap")$).

# kazan kaldIrmak
Label = R2v;
LEX = kazan, CASE = NOM;
R = kaldIr,  CAT = V:
Compose = ((*CAT* V)(*R* "kazan kaldIr")$).

# uygulamaya koymak
Label = R3v;
LEX = uygulamaya ;
R = koy,  CAT = V:
Compose = ((*CAT* V)(*R* "uygulamaya koy")$).

Label = M1adv;
LEX = Simdiye; LEX = dek:
Compose = ((*CAT* ADV)(*R* "Simdiye dek")).

Label = M1adj;
LEX = proto; LEX = neolitik:
Compose = ((*CAT* ADJ)(*R* "proto neolitik")(*AGR* 3SG)).

Label = M1;
LEX = topkapI; LEX = sarayI:
Compose = ((*CAT* N)(*R* "topkapI sarayI")(*SUB* PROP)$).

Label = M9;
LEX = gazi; LEX = hUsrev; LEX = bey; LEX = camii:
Compose = ((*CAT* N)(*R* "gazi hUsrev bey camii")(*SUB* PROP)$).

Label = M25;
LEX = istanbul; LEX = Universitesi:
Compose = ((*CAT* N)(*R* "istanbul Universitesi")(*SUB* PROP)$).
```

B.2 Constraint Specifications

```
#
# remove 2SG-POSS readings
Label = C1a;
POSS = 2SG, Action = Delete.

Label = C1b;
LP = 1, POSS = 2SG, Action = Delete.

Label = C1c;
LP = 2, POSS = 2SG, Action = Delete.

#
# 12. Ignore optative and imperative readings
Label = C12;
FINALCAT = V, ASPECT = OPT, AGR = ?, Action = Delete.

Label = C13;
FINALCAT = V, ASPECT = IMP, AGR = ?, Action = Delete.

#
# Eliminate N to V conversions, unless they are at the end
Label = C93;
SP = END, CAT = V, FINALCAT = V, CASE = NOM, Action = Delete.

#
# Eliminate ADJ to V conversions, unless they are at the end
Label = C3b;
CAT = ADJ, FINALCAT = V, SP = !END, Action = Delete.

#
# Eliminate V to ADJ conversions at he end
Label = C4;
CAT = V, FINALCAT = ADJ, VOICE = ?, SP = END, Action = Delete.

Label = C4;
LP = 1, CAT = V, FINALCAT=ADJ, VOICE=?, SP=END, Action=Delete.
```

```
#
# handle pronoun "bu"
Label = Cpn;
LP = 0, R = bu, CAT = PN, Action = Output;
LP = 1, FINALCAT = V.

Label = Cpn;
LP = 0, R = bu, CAT = ADJ, Action = Output;
LP = 1, FINALCAT = N.

#
# rules for postpositions
# these handle postposition subCATegorizations
Label = C25a;
CASE = _C, Action = Output;
LP = 1, CAT = POSTP, SUBCAT = _C, Action = Output.

Label = C25c;
LP = 0, SEMCASE = _C, Action = Output;
LP = 1, CAT = POSTP, SUBCAT = _C, Action = Output.


## these handle special cases for sonra and once
Label = C26;
FINALCAT = N, SUB =TEMP-UNIT, CASE = NOM, Action = Output;
LP = 1, CAT = POSTP, R = Once, Action = Output.

# 27.
Label = C27;
FINALCAT = N, SUB =TEMP-UNIT, Action = Output;
LP = 1, CAT = POSTP, R = sonra, Action = Output.

## special cases doGru
Label = C35;
LP = -1, CASE = !DAT;
R = doGru, CAT = ADJ, Action = Output.
```

```
## special cases yana
Label = CyanaPOSTP;
CAT = PN, AGR = 3SG,Action = Output;
LP = 1, R = yana, CAT = POSTP, Action = Output.

## icin prefer POSTP
Label = CicinPOSTP;
LP = 0, LEX = iCin, CAT = POSTP, Action = Output.

## special cases uzere
Label = CuzerePOSTP;
LP = -1, FINALCAT = INF;
LP = 0, LEX = Uzere, CAT = POSTP, Action = Output.

## special cases kadar
Label = CkadarPOSTP;
LP = -1, FINALCAT = INF;
LP = 0, LEX = kadar, CAT = POSTP, Action = Output.

# if all above fails remove POSTP
Label = CDeletUnwantedPostP;
CAT = POSTP, R = !ile, Action = Delete.

#
# Probably an exclamation, if succeeded by "!"
Label = C8-0;
CAT = EXC, Action = Output;
LP = 1, CAT = PUNCT, R = \!.

#
# Remove accusative readings if followed by a nominal
Label = CAcc;
LP = 0, CASE = ACC, Action = Delete;
LP = 1, FINALCAT = N.

# remove derived ADjs if there is an underived one
# e.g. mutlu, mut+lu
Label = C16-0;
```

```
CAT = ADJ, FINALCAT = ADJ;
CAT = !ADJ, FINALCAT = ADJ, Action = Delete.

#
# some rules for handling noun phrases

# smt's ADV  ADJ  smt
Label = C16b;
CASE = GEN, AGR = _A1, Action = Output;
LP = 1, FINALCAT = ADV, Action = Output;
LP = 2, FINALCAT = ADJ, Action = Output;
LP = 3, POSS = _A1, Action = Output.

# smt's ADJ ADJ  smt
Label = C16b;
CASE = GEN, AGR = _A1, Action = Output;
LP = 1, FINALCAT = ADJ, Action = Output;
LP = 2, FINALCAT = ADJ, Action = Output;
LP = 3, POSS = _A1, Action = Output.

# smt's smt's smt
Label = C16b;
CASE = GEN, AGR = _A1, Action = Output;
LP = 1, CASE = GEN, POSS = _A1, AGR = _A2, Action = Output;
LP = 2, POSS = _A2, Action = Output.

# Handle 3PL posses something singular
Label = C16b;
CASE = GEN, AGR = 3PL, Action = Output;
LP = 1, CASE = GEN, POSS = 3SG, AGR = _A2, Action = Output;
LP = 2, POSS = _A2, Action = Output.

# smt's N smt
Label = C17a;
CASE = GEN, AGR = _A1, Action = Output;
LP = 1, FINALCAT = N, Action = Output;
LP = 2, POSS = _A1, Action = Output.
```

```
# e.g. kitap kapaGI
Label = C18a1;
FINALCAT = N, CASE = NOM, AGR = 3SG, POSS = ~, Action = Output;
LP = 1, FINALCAT = N, POSS = 3SG, Action = Output.

# e.g. kitap kapaGI
Label = C18aINF;
FINALCAT = INF, CASE = NOM, AGR = 3SG, Action = Output;
LP = 1, FINALCAT = N, POSS = 3SG, Action = Output.


#
# some heuristics
#

Label = C44;
R = dUn, CAT = N, AGR = 3SG, CASE = NOM, Action = Delete.

Label = C45;
R = bugUn, CAT = N, AGR = 3SG, CASE = NOM, Action = Delete.

Label = C46;
R = yarIn, CAT = N, AGR = 3SG, CASE = NOM, Action = Delete.

Label = C47;
R = geri, CAT = N, AGR = 3SG, CASE = NOM, Action = Delete.

Label = C48;
R = ileri, CAT = N, AGR = 3SG, CASE = NOM, Action = Delete.

Label = C64;
LP = -1, FINALCAT = N, CASE = INS;
R = ilgi, FINALCAT = ADJ, Action = Output.

Label = C65;
LP = -1, FINALCAT = ADJ, CASE = INS;
R = ilgi, FINALCAT = ADJ, Action = Output.
```

```
Label = C66;
LP = -1, LEX = ile;
R = ilgi, FINALCAT = ADJ, Action = Output.

# at the sentence beginnings prefer ones without possesions
# e.g. ", etrafI"

Label = C76;
LP = 0, SP = BEGINNING, POSS = ~;
LP = 0, POSS = ?, Action = Delete.

Label = C77;
LP = -1, CAT = PUNCT;
LP = 0, POSS = ~;
LP = 0, POSS = ?, Action = Delete.

# verbal readings befor a question suffix are preferred
Label = C82;
LP = 0, FINALCAT = V, Action = Output;
LP = 1, CAT = QUES, Action = Output.

# try to eliminate some derivations
Label = H1;
LP = 0, R = sakal;
LP = 0, R = saka, Action = Delete.

Label = H2;
LP = 0, R = konuS;
LP = 0, R = kon, Action = Delete.

Label = H3;
LP = 0, R = hasta;
LP = 0, R = has, Action = Delete.

Label = H26;
LP = 0, R = yalIn;
LP = 0, R = yal, Action = Delete.
```

```
Label = H26;
LP = 0, R = yalIn;
LP = 0, R = yalI, Action = Delete.

Label = H59;
LP = 0, R = gemi;
LP = 0, R = gem, Action = Delete.

Label = H60;
LP = 0, R =kim;
LP = 0, R =kimse, Action = Delete.
```