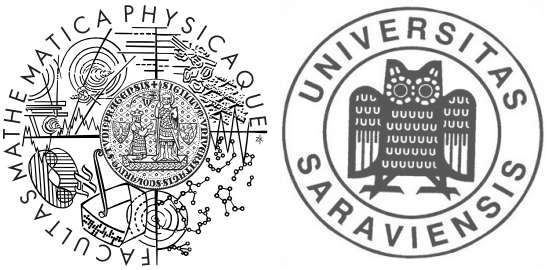
Processing of Turkic Languages



Sibel Ciddi

Faculty of Mathematics and Physics

Charles University in Prague

Thesis Supervisor:

RNDr. Daniel Zeman, Ph.D., Univerzita Karlova

Co-Supervisors:

Prof. Dr. Hans Uszkoreit, Universität des Saarlandes

Dr. Yi Zhang, Universität des Saarlandes

Specialization:

Mathematical Linguistics

A thesis submitted for the degree of

European Masters in Language and Communication Technologies

2012 - 2013

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ..... date ..... signature of the author

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

....., den ..... Unterschrift

Acknowledgements

I am thankful to everyone who has supported me during my journey sharing both my happiness and difficulties. I give my special gratitude to my local coordinators and supervisor in Prague, and the LCT program for giving me the opportunity to study in Europe. Finally, I am thankful to my sister, İdil, without whose support I could not have made it through this far.

Sibel Ciddi
Prague, 2013

Abstract

Title: Processing of Turkic Languages
Author: Sibel Ciddi
Department: Institute of Formal and Applied Linguistics,
Faculty of Mathematics and Physics, Charles University in Prague
Supervisor: RNDr. Daniel Zeman, Ph.D.
Abstract: This thesis presents several methods for the morphological processing of Turkic languages, such as Turkish, which pose a specific set of challenges for natural language processing. In order to alleviate the problems with lack of large language resources, it makes the data sets used for morphological processing and expansion of lexicons publicly available for further use by researchers. Data sparsity, caused by highly productive and agglutinative morphology in Turkish, imposes difficulties in processing of Turkish text, especially for methods using purely statistical natural language processing. Therefore, we evaluated a publicly available rule-based morphological analyzer, TRmorph, based on finite state methods and technologies. In order to enhance the efficiency of this analyzer, we worked on expansion of lexicons, by employing heuristics-based methods for the extraction of named entities and multi-word expressions. Furthermore, as a preprocessing step, we introduced a dictionary-based recognition method for tokenization of multi-word expressions. This method complements tokens in the larger multi-word expression lexicons, and prepares them for further morphological processing. Experiment results point out that the new addition of lexical tokens provide promising coverage increase for the text processed by TRmorph. The dictionary-based recognition method enables tokenization of multi-word expressions; and tokenized multi-word expressions help reducing morphological ambiguity. The proposed method enhances the efficiency of a purely rule-based morphological analyzer with finite-state transducers.
Keywords: morphological analysis, finite-state transducer, finite-state automata, recognition and tokenization of named entities, and multi-word expressions, morphological & lexical ambiguity.

Abstrakt

Název: Zpracování turkických jazyků
Autor: Sibel Ciddi
Katedra: Ústav formální a aplikované lingvistiky,
Matematicko-fyzikální fakulta, Univerzita Karlova v Praze
Vedoucí diplomové práce: RNDr. Daniel Zeman, Ph.D.
Abstrakt: Tato práce představuje a na příkladu turečtiny demonstruje několik metod morfologického zpracování vhodných pro turkické jazyky, jejichž počítačové zpracování přináší sadu specifických problémů. Přínosem práce je také značné rozšíření lexikální databáze a souvisejících dat potřebných pro morfologickou analýzu a syntézu; tato data jsou nyní volně dostupná veřejnosti. S ohledem na vysoce produktivní a aglutinační tureckou morfologii a s ní spojenou řídkost dat byl omezený rozsah slovníku významnou překážkou počítačového zpracování jazyka, zvláště pokud jde o zpracování statistickými metodami. Proto jsme důkladně otestovali a vyhodnotili veřejně dostupný, na konečných převodnících založený morfologický analyzátor TRmorph. Zaměřili jsme se na rozšíření záběru a slovníku tohoto analyzátoru. Za tím účelem jsme navrhli heuristické metody pro získávání pojmenovaných entit a víceslovných výrazů. Další vylepšení spočívá ve slovníkovém rozpoznávání víceslovných výrazů, které je předstupněm k jejich morfologickému zpracování. Výsledky experimentů ukazují, že takové heuristické rozšíření slovníku slibně zvyšuje pokrytí textů, které jsme s pomocí TRmorphu rozebrali. Slovníková metoda nejen umožňuje správnou tokenizaci víceslovných výrazů, ale také tím snižuje (lexikální) morfologickou nejednoznačnost a připravuje analyzovaný text pro použití v aplikacích vyšší úrovně. Námi navržený přístup tak zvyšuje účinnost čistě pravidlového morfologického analyzátoru s konečnými převodníky.
Klíčová slova: morfologická analýza, konečný převodník, konečný automat, rozpoznávání pojmenovaných entit, rozpoznávání víceslovných výrazů, morfologická a lexikální nejednoznačnost.

Contents

Chapter 1

Introduction

There have been many research studies, and special interests groups around morphologically rich languages with efforts to solve natural language processing (NLP) problems, concerning their complex morpho-syntactic and morpho-phonological structures. Many morphologically rich languages are at the same time low-resource languages, i.e. the extent of available annotated data, dictionaries etc. for these languages is far from sufficient. Given these circumstances, they continue to provide a lot of unresolved questions for researchers in the NLP field. Agglutinative languages, such as Finnish, Hungarian and Turkish, indeed have an extremely rich morphological system. In the present thesis, we focus on Turkish; even though there are now some resources for this particular language, Turkish is a representative of a large group of closely-related Turkic languages, which are mostly very resource-poor. We believe that the methods we propose to extend the Turkish resources can be largely reused for other languages in this language family.

The existing research methodologies and various processing tools do attempt to take advantage of recent developments in NLP. However, due to the complex structure of agglutinative languages and the lack of resources, the state-of-the-art implementations of various NLP tools, such as parsers, part-of-speech taggers, morphological analysers, and Named Entity Recognition (NER) systems for these languages still cannot be effectively compared with other NLP tools developed for less complex languages that have access to a lot more resources.

In order to alleviate some of these current issues, this thesis aims to assess currently available NLP tools for the morphological analysis of the Turkish language, and propose new extensions and methods with proven effectiveness in order to complement the missing components, and improve the accuracy and the efficiency of the existing tools. Finally, it aims to provide the researchers and academicians

with publicly available data resources that can be used for further research in similar issues, or for the extension of the methods that will be described in this thesis.

With these goals in mind, the following chapter 2 discusses morphological processing and analysis in a general context and describes the main features of Turkish morphology that sets up the ground for the motivation in pursuing this research. Chapter 3 continues to discuss the previous work and main methods and theories in morphological processing with a focus on morphological analysis of Turkish. Afterwards, it follows up by introducing one of the open-source morphological analyzer tools, *TRmorph: A Turkish morphological analyzer*[1] [2] by Çöltekin (2010)—which sets up the main focus of this thesis. Chapter 4 continues with the evaluation of TRmorph analyzer by assessing coverage and evaluation metrics of data sets that were available at the time of this research.

The following chapters and sections, thereafter discuss three of the main, interrelated issues concerning the morphological analysis of Turkish, thus the TRmorph tool; and describe our proposed methods for improving those issues. In other words, the focus of this thesis can be summarized around these main issues as the following:

i. Morphological processing of out-of-vocabulary (OOV) tokens by extending the fixed-lexicons in TRmorph

ii. Recognition and tokenization of Named Entity (NE) and proper nouns; Recognition and tokenization of Multiword Expressions (MWEs) for further morphological processing

iii. Morphological processing of unknown proper nouns via a finite-state guesser

After the discussion of our proposed methods, it also discusses present issues and challenges about the potential methods for morphological disambiguation and more fine-grained and accurately ranked morphological analysis.

More precisely, chapter 5 provides a discussion of techniques and methods employed for the expansion of fixed-lexicons in TRmorph-Baseline with by providing an extension of existing lexicons and adding several new lexicons. Afterwards, it introduces our method for the recognition and tokenization of multi-word expression units (including Named Entities) that is necessary for further morphological processing. In the last section, it provides a description of the finite-state guesser for OOV proper nouns, and follows up with the summary of current architecture

[1] `http://www.let.rug.nl/~coltekin/trmorph/`
[2] `https://github.com/coltekin/TRmorph`

and work-flow that has been implemented for the new TRmorph analyzer.

Chapter 6 provides an evaluation of TRmorph—utilizing our methods—by evaluating several data sets, including the manually created new data set with a collection of sentences, consisting of examples with multi-word expressions. It further provides a comparison of data sets evaluated with the TRmorph-Baseline, and evaluated with TRmorph utilizing our methods. At last, it discusses the remaining challenges regarding different tagsets used in different morphological analyzers, and our attempt at tagset conversion as a preprocessing step for context-based morphological disambiguation. Finally, the last chapter 7 discusses the conclusions gathered from this thesis study, and describes some of the limitations that remain to be challenging. At the end, it provides a summary of questions and issues that are yet to be resolved for further research.

Chapter 2

Motivation

> Morphology is the study of the way words are built up from smaller meaning bearing units i.e., morphemes. A morpheme is often defined as the minimal meaning-bearing unit in language. So, for example, the word fox consists of a single morpheme (the morpheme fox) while the word cats consists of the morpheme cat and the morpheme –s. As this example suggests, it is often useful to distinguish two broad classes of morphemes: stems and affixes. The exact details of the distinction vary from language to language, but intuitively, the stem is the "main" morpheme of the word, supplying the main meaning, while the affixes add "additional" meanings of various kinds.
> (Jurafsky et al., 2000)

2.1 Morphological Processing

In written text processing, morphological analysis is one of the most important tasks that most NLP tools need as a supplement to their final toolkit because it serves as the basis for the development of more comprehensive natural language processing tasks. For example, NLP applications that need to use a part-of-speech tagger, *(shallow)* parsers, or phrase chunkers, Information Extraction (IE) tools such as Question-Answering systems, or tools that are used in the development of machine translation systems, and text proofing tools such as spell checkers and grammar checkers, Computer Assisted Language Learning (CALL) tools and electronic and/or web-based dictionaries often rely on annotated text. In order for these applications to be designed properly, morphological analyzers serve as a pre-requisite, and often as a pre-processing layer.

Therefore, if we think of such natural language processing applications having

a hierarchical workflow order; morphological analyzers would be placed in one of the initial steps of the workflow that enable the larger NLP tools and systems to proceed to the next steps required for their own set of tasks. If we examine the steps that make up the morphological processing task, then we can divide those steps into their own sub-tasks. In that case, those steps that lead to morphological processing as a whole can be described as *morphological analysis*, and *morphological generation*.

Morphological Analysis

Most commonly, morphological analyzers are implemented as *finite-state transducers*. Their main task is to map a given word form to all its possible morphological tags. For example, the word form *'drinks'* may be mapped to its morphological tags and return as its output: `drink+V+3p+Sg` which would denote that the word form being analyzed is a verb, in third-person, singular. Another potential analysis could also show that the word form *drinks* may be mapped to the morphological tags, showing: `drink+N+3p+Pl`, denoting that *drinks* could also be a noun, in third-person, plural form.

As discussed in a greater detail in Beesley and Karttunen (2003), the morphological analysis task–*which is often called as 'lookup' task as well*–returns as successful only when the word form that leads to such an analysis has previously been described *(often via a set of various rules)* for that language. This process requires matching of the symbols from the input words to verify them against the pre-defined symbols *(and rules)* for that language. Such a process implies that a morphological analysis of a word form becomes successful if and only if the required pre-steps have already been done (e.g. the grammar engineer pre-defines the symbols, and necessary rules in advance). Otherwise, the analyzer returns no output. Therefore, in the implementation of a morphological analyzer, considering the most important thing is the pre-definition of symbols *(and rules)* that results in the analyzed output of the input word form; how the analysis output is generated becomes trivial.

Morphological Generation

Considering how the morphological analysis task is done with the finite-state-transducers, we can describe similar set of processes and steps for the task of morphological generation. This becomes possible once a finite-state-transducer is built; then it can be considered as a bidirectional processor, which can infer both the potential set of all morphological tag sequences for a given word form, and also the word form given a set of morphological tag sequences.

In other words, as we can see from the description of the task of a morphological analyzer as providing *all* potential analyses of a word form–*which was shown by the ambiguous word form example 'drinks'*–then it is assumed that the task of morphological generation is to map the morphological tag sequences to their associated word forms. This implies that the morphological generation task is required to perform the opposite of the tasks done by the morphological analyzer–in a backwards direction–given a set morphological tag sequences, it is expected to match the set of tag sequences to the word form they are associated with. In this case, if a morphological processor is given the set of tags as `drink+N+3p+Pl`, or `drink+V+3p+Sg`; in either case, the expected word form would be *'drinks'*.

As it is the case with morphological analysis task; in the same way, the morphological generation task also returns a successful matching output, *if and only if* the given set of specific morphological tag sequences has already been defined for that language. In other words, for the generator to return a matching output, via a set of specific rules, and definitions, the finite-state-transducer needs to be taught that the English morpheme *-s* can lead both to a plural form of a *regular* noun, and also the third-person, singular, present tense form of a *regular* English verb.

Given these descriptions of morphological analysis and morphological generation tasks, we see that in natural language processing applications and tools, the role of morphological processing cannot be underestimated. This is not exactly because morphological processing is *not* avoidable, due to being used as an initial step in a natural language processing tool; but because the types of output that can be obtained from a morphological processor can also become useful in the further stages of an NLP task. Because morphological processors have the bidirectional capacity to provide both the word forms *(given tag sequences)*, and also the set of morphological tag sequences *(given the word forms)*, the output produced by a processor can be used at different stages of the development of a tool.

However, it is also important to point out that—as we can see from the examples of *regular* English verb forms, and *regular* English nouns—the rule-based nature of morphological processors may also make them susceptible to some of the challenging problems that are commonly observed in morphological processing in general. Such problems may become even more complicated and harder to define depending on language specifications. For example, in Turkish, considering the word *'sular'* (water: verb, noun), looking at its morphological analysis and generation, we should be able to see the following parts of speech and different

derivations based on the same surface form. For *all* of the analyses[1] of *'sular'*:

(1)
```
apply up> sular
sula<v><t_aor><3p>
sula<v><t_aor><3s>
su<n><pl>
su<n><pl><3p>
su<n><pl><3s>
```

For the generation to be obtained from verb form, `sula<v><t_aor><3s>`:

(1-a)
```
apply down> sula<v><t_aor><3s>
sular
Sular
SULAR
```

For the generation to be obtained from noun form, `su<n><pl>`:

(1-b)
```
apply down> su<n><pl>
sular
Sular
SULAR
```

However, even if this example may look like a straightforward case of a morphological analysis and generation; we cannot always assume that it is possible to generate *any* word form given its tag-sequence. Because pattern-based rules may not always be general enough to make language specific assumptions, we may not always be able to generate the corresponding word form of a tag-sequence, if that tag-sequence is based on previous observations, and the knowledge of previous rule generations. This issue can be demonstrated with an example, in Turkish, by looking at the instrumental noun case suffix: *-le, -la*
If we analyze the word, *'şarapla'* (wine+inst. as in *with wine*), we get the following derivations:

(2)
```
apply up> şarapla
şarap<n><ins>
şarap<n><ins><3p>
şarap<n><ins><3s>
```

[1]For the remaining of this thesis, all the morphological analyses are shown in the following convention:
1. Prompt for analysis : "apply up>" or "apply down>"
2. The input: "sular" *(following the prompt)*
3. The output: Lines 2 to 6 *(after the input)*

If we put this analysis in a mini experiment, relying on these derivations obtained from the analysis of *'şarapla'*, we assume that the word stem is *'şarap'*, and the suffix *-la* is the instrumental case. In this experiment, we make a generalization and derive a word form from *'water'* → *'su'* to make the *hypothetical* instrumental case form *'with water'* → *'sula'*. According to our previous assumptions and observations from the word *'şarapla'*, then when we analyze the word form *'sula'*; we should be able to get the same derivations that we observed for the example *şarapla*, in (2):

(2-a) `apply up> sula`
`sula<v><t_imp><2s>`

Wrong assumption. The morphological analysis of the word form *sula* contradicts our previous assumption—which was based on the derivations of the word form *şarapla*. If we want to see the word form of *su-* in instrumental case, then we generate it by su<n><ins>:

(2-b) `apply down> su<n><ins>`
`suyla`
`Suyla`
`SUYLA`

Notice the */y/* between *su-* and *-la*, which was not in our initial assumption[1] based on the previous example of the word form *şarapla*. Therefore, this mini experiment shows that not all tag sequences that are rule-based patterns can be directly applicable to make general assumptions; and that there might be a variety of other factors that might give the final word form of words in a language lexicon. This shows us that it is important to identify language-specific points that may pose challenging problems in morphological processing. As described in Beesley and Karttunen (2003), the two biggest challenges in morphology come out as the morphosyntax (*aka. morpho-tactics*), and morphophonology of a given language.

These problems can be described as 'word formation'–which is made up from morphemes, smaller parts of meaning split within a word–and 'phonological and orthographical alternation'–which is the spelling or sound of a morpheme occurring in a specific context. Karttunen (1991) argues that word formation process comes out as a result of *principles* that impose constraints on the combinations of stems, affixes, and other types of morphemes. On the other hand, the issue of

[1]The -y infix is used only with certain suffix forms when the word stem ends with a vowel as in *su*, and it is followed by such suffixes starting with a consonant, as in *-le, -la* instrumental case. For such nouns, the exception rules must be made in advance.

morphological alternations stem from the fact that a single morpheme can appear in different phonological environments and different word formations without losing its original meaning.

In other words, *word formation* dictates the specific constraints on the order and combination of morphemes within a word. For example, English derivations follow a certain order. We can derive 'playfulness' from the stem 'play-'+full+ness, but not *play-ness-full. As for *alternation*, we observe that certain changes appear as context-dependent sound and spelling alternations, as in the English *-s* suffix to denote plurality of nouns, except for nouns ending with *-s, -ch, -sh, -z*, where the plural form is altered as *-es*, instead of the regular, single *-s* plural suffix. According to these irregular forms in English, for example, 'glass' becomes 'glasses', and 'church' becomes 'churches'.

In order to further examine the issue of word formation and context-dependent phonological & orthographical alternations, the following section describes and discusses how these kinds of potentially problematic morphological issues reflect on Turkish morphology, while giving a general overview on the background and the main aspects of Turkish morphology.

2.2 Turkish Morphology

Dilbilimcileştiremeyebileceklerimizden miydiniz?

As it can be seen from '*Dilbilimcileştiremeyebileceklerimizden_miydiniz*'[1]-[2], the productive inflectional and derivational morphology enables Turkish to have lengthy word formations–that can even be used as a whole sentence. Therefore, we can deduce that more than the order of words in a sentence, it is this complex structure of morphology that gives a sentence both the syntactic and semantic context. Eryiğit et al. (2008) argues that regardless of the SOV order predominantly seen in written texts, Turkish is a flexible word order language, and the order of words may change depending on the context. However, we can assume that from a dependency structure point of view, Turkish is primarily (but not exclusively) head final.

[1]It can be translated as *'Were you one of those whom we would not be able to transform into a linguist?'*, and segmented as "Dilbilim-ci-leş-tir-e-me-yebil-ecek-ler-i-miz-den-mi-ydi-niz" leads to 36 possible analyses.For more, see:`http://en.wikipedia.org/wiki/Longest_word_in_Turkish`

[2]For more details and the derivational analysis, see Appendix A, 101.

Besides the primarily head-final structure of the language, Turkish is also an agglutinative language that derives words and new word forms from existing roots via suffixation. Kerslake and Göksel (2005) describes this word formation process as *'the formation of a new word by attaching an affix to the right of a root.'* In Turkish, suffixation is done by means of derivational and inflectional suffixes. Except for borrowed or foreign words, the use of prefixes is not a part of Turkish morphology. Since the order of suffixation can follow both the derivational and the inflectional suffixes[1] *(as well as certain infixes[2])*, it becomes possible to create these kinds of long word forms. While the example used here only serves to make the point about the productive morphology–and it should be noted that the given example is not a commonly used, frequent word–we can safely assume that this agglutinating nature of the morphology in Turkish makes the word forms less common than other languages' word forms, because the majority of word forms in Turkish appear more unique as a result of unique suffixation. This problem alone causes one of the biggest difficulties for the field of natural language processing, especially for morphological processing with statistical methods as the problem of data sparsity is unavoidable.

2.2.1 Word Formation in Turkish

Among Turkish language linguists and grammarians, there is a general consensus that part-of-speech labels of words may not be clearly defined without a given context because most words are derived from either nominal or verbal root forms initially, and their final surface form may be a different part-of-speech. Therefore, Hengirmen (2005) suggests that it is important to consider the meaning and the context of the word in that given sentence before determining its part-of-speech label.

As one of the main identifiers of the part-of-speech of a word, the types of suffixes determine the types of words in Turkish. In other words, we see that the types of words are distinguished according to the types of suffixes they take–whether they can take derivational suffix, or inflectional, or both. Therefore, Turkish *word formation* can be categorized as the following groups of words:

1. **Simple words**: Group of words whose stems are derived from a single part-of-speech. They take only inflectional suffixes:

    i. masa-**lar** → *table**s***

[1]Note that in most cases, in a complex word, derivational suffix(es) come before the inflectional suffix classes.

[2]The negation suffix -me, -ma is used as an infix.

ii. kedi-**cik**→ *cat-**diminutive***

iii. köpek-**ler-im** → *dog-**s-my***

2. **Complex words**: Group of words whose stems are derived from a different part-of-speech, or still the same part-of-speech, where the meaning of the stem changes [1] due to derivational suffixation *(for example, verb to verb, noun to noun, or verb to noun, or verb to adjective, etc.)*. Following a derivational suffix, they can take inflectional suffixes as well, as it is shown by the following examples:

   i. uyku-**lu** → *sleep-**y** (from noun to adjective)*

   ii. dal**gın** → *(to) drift^DB*[2] → absent-minded
   (from verb to adj, as in from 'to drift' to 'absent-minded').

   iii. kitap-lık-**lar** → *book^DB+plr.* → bookshelves
   (from noun to noun, as in from 'book' to 'bookshelves').

   iv. sev**gi** → *(to) love^DB* → love-*ing*
   (from verb to noun, as in from 'to love' to 'loving').

3. **Compound words**: Group of words that are formed as a combination of two words written together. Sometimes one of the words lose its original meaning, sometimes both lose its original meaning, and sometimes both preserve their original meaning. In all cases, they can take both derivational and inflectional suffixes, such as:

   i. dil+bilim+**ci** → dilbilim**ci** *(language+science^DB: linguist).*

2.2.2 Morphophonology *(sound alternations)* in Turkish

As for the morphophonology of Turkish, as it was briefly shown with the examples in the previous chapter, we see that the following phonological phenomena—which mainly has to do with vowel harmony and consonant changes—give their emphasis to the word stems and cause certain alternations (and certain irregularities) occurring in specific contexts. As it is described in (Kerslake and Göksel, 2005, p. 21), the forms of suffixes are conditioned by the vowels and consonants that

[1] Note that these are the types of words that can be ambiguous if there is an overlap between certain derivational and inflectional suffixes, e.g.: *-ecek, -acak* suffixes can be used both as tense inflection when added onto the verbs, and also it can make verbs into adjectives when the word occurs in a certain context. For example, in *'Şiiri sonuna kadar okuy-acak'* (s/he **will read** the poem till its end) where 'okuy-acak' is a verb, and in *'Şiiri okuy-acak çocuk geldi'* (The child who **will read** the poem has arrived), where 'okuy-acak' is an adjectival.

[2] DB:Derivational Boundary where the derivational morphemes are attached to the stem

precede them.

Initial consonants and vowels in suffixes are conditioned by the consonants and vowels in the preceding syllable. How these constraints are conditioned can be shown by the following vowel chart in 2.1, which shows the categorization of the vowel sounds according to their position in terms of frontness, backness, roundness and unroundness, and highness and lowness:

	FRONT	**BACK**
	Rounded & Unrounded	Rounded & Unrounded
HIGH	ü, i	u, ı
LOW	ö, e	o, a

Table 2.1: Chart of Turkish Vowels

Given this vowel chart, in their work, Kaplan and Kay (1994) describe the process of vowel harmony and how the morphophonological rules are applied to Turkish words[1] with four phonological rules. Considering the following vowels and their corresponding representations mappings:

i. The vowel `/A/` represents `/e/` and `/a/` as front and back and low.

ii. The vowel `/I/` represents `/i/` and `/ı/` as front and back and high.

   a. The group of vowel harmony seen in (i.) applies to the plural suffix `'lAr'`

   b. The group of vowel harmony seen in (ii.) applies to the possessive suffix `'Im'` for *'my', 'your', 'her', 'his'*, etc.

Both of these suffixes can be used together, and the vowel harmony is still preserved via the different/corresponding realizations of vowels. To illustrate these representations with examples:

i. The word form 'arkadaş`lArIm`' for *'my friends'* applies to the vowel harmony seen in (i).
   -The realized surface form of 'arkadaş`lArIm`' appears as 'arkadaş*lar**ı**m*'.

ii. The word form 'asker`lArIm`' for *'my soldiers'* applies to the vowel harmony seen in (ii.)
   -The realized surface form of 'asker`lArIm`' appears as 'asker*ler**i**m*'.

[1]This example concerns only the words having /e/ or /a/ in the last syllable of the stem.

Returning back to the more formal representations of these phonological rules as described by Kaplan and Kay (1994), the vowel harmony variations—for the words forms above—are formalized as following:

1. A → e / e C*__
2. A → a
3. I → i / e C*__
4. I → ı

The correct surface representations of words are generated when the rules are applied from left to right, one at a time. However, considering the rules given above, application of these rules gives an incorrect analysis when we apply the dative possessive suffix to *'asker'*, which is represented as the vowel form `/A/` as in the case (i.) above—by producing the following incorrect form:

▷ asker → asker*im* → asker*imA*

This incorrect analysis results from the fact that the rule (1) above imposes that every final syllable vowel, except for /e/, must be followed by /a/ (2), instead of /e/[1]. Thus, for the possessive dative suffix, the left-to-right rule application results in the generation of vowel /a/ for a word form that has /i/ in its preceding syllable *(due to application of rule (3) /I/ for the possessive suffix in asker-im—before the dative suffix is attached)*. If the vowel harmony is fully considered, asker*imA* should be realized as asker*ime*.

Kaplan and Kay (1994) argues that correct analyses of word forms can only be generated if the vowel harmony is described with rules that proceed left to right through the string as a group, by applying the rules at each position for the one that matches. This way, application of a set of rules collected together as a batch gives the correct results in all analyses of the word forms.

However, for a more complete description of the mapping of vowel variations between /a/ and /e/ in the 'lAr' suffix, as well as others; the other vowels–considering their *frontness*, *backness*, and *roundness*, *unroundness* features–must be taken into account. The four rules given above present only a partial description of left-to-right phonological rule application. In this restricted context, they would not result in correct concatenation of suffixes for every possible word form.

[1] In other words, /A/ can only have the representation of /e/ if the final/preceding syllable is also /e/.

In the following section, we provide complete description of the vowel harmony and morphophonological alternations, considering all the contexts where vowel harmony and consonant alternations are applied.

2.2.3 Vowel Harmony

The vowel alternations are conditioned by two types of *vowel harmony*[1] rules:

Two-way Vowel Harmony

This type of vowel harmony, which is occasionally called *'front-back vowel harmony'* dictates the type of vowels within a word according to their *'frontness'* or *'backness'*. According to this vowel harmony, a word cannot have both frontal and back vowels. If we look at the plural suffix *-ler, -lar*, we see that two-way vowel harmony—where the vowel alternates between *-e* and *-a*—determines which one of these plural suffixes a noun can take. The noun examples below show how morphophonological alternations apply to the the plural word formation: As we

Word		**Word Form**	
köp**e**k	→	köpek-l**e**r	*(dog, dogs)*
ik**i**z	→	ikiz-l**e**r	*(twin, twins)*
k**ö**r	→	kör-l**e**r	*(blind, blinds)*
üz**ü**m	→	üzüm-l**e**r	*(grape, grapes)*
çoc**u**k	→	çocuk-l**a**r	*(child, children)*
dokt**o**r	→	doktor-l**a**r	*(doctor, doctors)*
om**u**z	→	omuz-l**a**r	*(shoulder, shoulders)*
ay**a**k	→	ayak-l**a**r	*(foot, feet)*

Table 2.2: Two-way Vowel Harmony & Plr. Word Formation

can see from these examples, plural word formation is done with the alternation of front or back vowels. Words that have a front vowel in their final syllable take 'ler', words with a back vowel in their final syllable take the 'lar' suffix in the plural form. The mapping of vowels to suffix can be shown as following:

i. **Front vowels:** /i, ü, e, ö/ → -l**e**r

ii. **Back vowels:** /ı, u, a, o/ → -l**a**r

[1]Note: Vowel harmony rules apply to words with Turkish origin. Foreign and borrowed words may be exceptions to these rules.

Four-way Vowel Harmony:

Secondary type of vowel harmony, also occasionally called *'rounding'* harmony imposes matching between *high* and *low* vowels. It consists only of *high-front* and *high-back* vowels: *'i, ü, ı, u'*.
According to this type of vowel harmony, words ending with *low-front* vowels are followed by their *high-front* counterparts in their suffix. Namely, after a word ending with *'low-unrounded-front'* vowel, its counterpart *'high-unrounded-front'* vowel follows in the initial suffix. As a result of this, the following mapping of rounded / unrounded vowels comes out:

i. /e, i/ → /i/

ii. /ö, ü/ → /ü/

iii. /a, ı/ → /ı/

iv. /o, u/ → /u/

Therefore, with different suffixes, four-way vowel harmony might be applied. For example, the question-making clitic suffix *-mi* is constrained by the preceding syllable's vowels and the four-way vowel harmony is applied—vowel alternates between *-mi, -mu, -mü, -mı*[1]. Some examples:

Word	**Q-Clitic**	
güz**e**l	m**i**?	*(is it 'nice'?)*
mas**a**	m**ı**?	*(is it 'table'?)*
b**u**	m**u**?	*(is it 'this'?)*
s**ü**t	m**ü**?	*(is it 'milk'?)*

Table 2.3: Four-way Vowel Harmony & Question-Clitic Suffixation

2.2.4 Consonant Alternations

Also depended on the preceding vowel and consonant in the word stem, the consonant alternation occurs in certain environments. Consonant final syllables in the stem of the word, and the suffix following the stem go through assimilation in order to have similar sounds. This type of consonant assimilation is conditioned

[1]high-frontal-rounded: ü, high-frontal-unrounded: i, high-back-rounded: u, high-back-unrounded: ı

depending on whether the final syllable of the stem ends with a voiced or voiceless consonant, and the following suffix initial sound starts with a voiced or voiceless consonant *(or a vowel)*.

The chart 2.4 below shows the voiced and voiceless consonants in Turkish:

Voicing	**Consonants**
`Voiced:`	`b, c, d, g, ğ, j, l, m, n, r, v, y, z`
`Voiceless:`	`ç, f, h, k, s, ş, t, p`

Table 2.4: Chart of Voiced & Voiceless Consonants

According to this chart, two types of assimilation can be described:

1. Voiced-Voiced & Voiceless-Voiceless Consonant Alternation

Words ending with *voiced* consonants are followed by suffixes starting with one of the following *voiced* consonants: /c, d, g/. Words ending with a `voiceless` consonant are followed by a suffix starting with a `voiceless` consonant. Following table illustrates some of the examples:

	Word	**Locative:** /de, da, te, ta/	
`Voiced:`	E**v**	E**v-d**e	*(Home, home-at)*
`Voiced:`	Oku**l**	Oku**l-d**a	*(School, school-at)*
`Voiceless:`	Ofi**s**	Ofi**s-t**e	*(Office, office-at)*

Table 2.5: Voiced&Voiced and Voiceless&Voiceless Consonant Alternation

2. Voiceless to Voiced Consonant Alternation

Words ending with one of the following voiceless consonant /p, ç, t, k/, are followed by suffixes starting with those voiceless consonants.

However, one exception is that if the initial suffix starts with a vowel—as in the dative noun case /e, a/; then the final voiceless consonant /p, ç, t, k/ in the stem of the word is alternated with its *voiced* counterpart. In that case, the final-voiceless consonant in the stem alternated as the voiceless consonant becomes voiced in the surface form.

Corresponding mapping of these alternations can be shown as the following:
▷ p → b
▷ t → d
▷ k → g/ğ
▷ ç → c

The examples in the table below illustrate these alternations:

Word	**Voiceless-to-Voiceless** **Loc**.: /de, da, te, ta/	**Voiceless-to-Voiced** **Dat**.: /e, a/	
Ağa**ç**	Ağa**ç-t**a	Ağa**c**-a	*(Tree, tree.loc, tree.dat)*
Yata**k**	Yata**k-t**a	Yata**ğ**-a	*(Bed, bed.loc, bed.dat)*

Table 2.6: Voiceless to Voiced Consonant Alternation

Irregular Sound Changes

Besides the regular morphophonological alternations described as above, certain *irregular* sound changes that affect both consonants and vowels occur in the following contexts also:

1. Sound Derivation

When the word stem ends with a consonant and either the immediate suffix that follows starts with a vowel, or the stem is followed by an auxiliary word that starts with a vowel[1]; the final consonant of the stem is doubled. For example:

i. hi**s** + etmek → hi**s-s-**etmek *(to feel)*

Additionally, the sound derivation is observed with vowels as well if a single syllable word stem takes ones of the *'-cik, -cık, -cuk, -cük'* diminutive suffixes. For example:

ii. bir → bir-**i**-cik *(one, little one)*

[1]Derivation of vowels and consonants is also common with foreign or borrowed words.

2. Vowel Drop

In general, when a two-syllable word ending with a consonant[1] takes a suffix starting with a vowel—if the vowel in the second syllable of the stem is a high vowel such as /i, ü, ı, u/; then, this vowel is dropped from the stem. For example:

i. bur**u**n → bur~~**u**~~n-**u**m → burn-**u**m *(nose, nose-my)*

Additionally, this kind of vowel drop is also observed with *complex words (as is described in the previous section)*, for verbs that are originally derived from nouns ending with a vowel, and for nouns that are originally derived from verbs ending with a vowel. For example:

ii. duy**u-**: *hearing* → noun
duy~~**u**~~-mak → duy-mak: *to hear* → noun+ˆD_verb[2].

iii. uy**u-**: *(to) sleep* → verb root
uy~~**u**~~-ku → uy-ku: *sleep* → verb+ˆD_noun[3]

3. Consonant Drop

When the words ending with the consonant /k/ take one of the diminutive suffixes *'-cik, -cık, -cuk, -cük'*, the consonant-final from the stem is dropped. For example:

i. küçü**k** → küçü~~**k**~~-cük → 'küçü-cük' *(small, small+dimun.)*

ii. mini**k** → mini~~**k**~~-cik → 'mini-cik' *(mini, mini+dimun.)*

Additionally, in order to avoid double consonants in adjacent syllables, similar type of consonant drop is observed when a noun stem ending with a consonant takes a suffix that starts with the same consonant. In those cases, one of the consonants is dropped. For example:

iii. A**d** → A**d-d**aş → A**d**aş *(name, namesake)*

[1]This kind of vowel drop is most common with the words related to *organs*, that are made of two-syllables in the form of: 'V-CVC' or 'CV-CVC'

[2]Verb that is made via verbal derivational suffixation, shown by 'ˆD_verb', attached to a noun stem.

[3]Noun that is made via nominal derivational suffixation, shown by 'ˆD_noun', attached to a verb stem.

4. Vowel raising

The final *low*-vowel /a, e/ in verb stems is alternated with one of the high vowels /ı, i, u, ü/ if the verb takes verbal aspect inflection *-yor*. In most cases, the alternation occurs as in the following correspondence mapping:

i. the final vowel /a/ → /ı, u/

ii. the final vowel /e/ → /i, ü/. For example:

   a. kokl**a**- *(to smell)* → kokl**u**-yor → *s/he is smelling*

   b. kayn**a**- *(to boil)* → kayn**ı**-yor → *it is boiling*

   c. ekl**e**- *(to add)* → ekl**i**-yor → *s/he is adding*

   d. özl**e**- *(to miss)* → özl**ü**-yor → *s/he is missing*

The next chapter discusses the background and related work regarding how these morphophonological changes and word formations in Turkish have been handled by various morphological processors, implemented for the analysis of Turkish morphology to this day. Furthermore, it introduces the open-source morphological analyzer, TRmorph (Çöltekin, 2010), as the focus of this thesis. In the remaining chapters and sections, it continues with the relevant data coverage assessments and evaluations, followed by a discussion for further improvements necessary for the TRmorph morphological analyzer.

Chapter 3

Overview of Previous Work and Existing Tools in Processing of Turkish

For the task of morphological processing, to our day, several morphological analyzers have been implemented. Most of the morphological analyzers that are being used today differ from one another in the type of methods and approaches used in their implementations. The two of the most commonly used approaches in morphological processing are generally based on rule-based methods, and statistical methods. The rule-based methods largely employ finite-state automata and transducer constructions in their implementations, while the statistical methods employ a variety of different learning approaches, such as supervised, semi-supervised, or unsupervised learning settings for handling of data and building of morphological analyzer models.

Besides the purely rule-based and purely statistical methods, another approach also exists as a *hybrid method*, which employs a combination of these two approaches based on rule-based and statistical methods. For the morphological processing of Turkish text, this hybrid method has also been used in the past. This chapter gives an overview of the background for what has been done in general for the morphological processing task in natural language processing, and it further discusses the previous work that has been done for the morphological processing of Turkish language.

3.1 Rule-Based Methods in Morphological Processing

One of the earliest studies done with Turkish morphology by using finite-state machine techniques is from the work of Hankamer (1986) examines a parser for words formed agglutinatively and investigates the parser's behavior for morphophonological alternations in Turkish. By employing a left-to-right parsing technique, originally based on the study of finite-state transition networks by Koskenniemi (1984), Karttunen (1983) and Hankamer (1984); the parser works by using cyclic generative-rules by reading words from left-to-right in suffixation languages like Turkish.

It considers that phonological rules apply one morpheme at a time, starting with the root and moving forward to the right. Due to cyclic phonological rules, whenever a phonological rule is applied and matches a surface string; the word formation ends, in other words, that word is never examined again.

The Keçi system designed and implemented by Hankamer (1984) for the processing of Turkish, checks for the harmony rules of Turkish, and tags for each morpheme in a suffixed word according to its order as in N0, N1, and so on. While the Keçi system is powerful in showing the strictly concatenative morphology of Turkish, other systems using FSM techniques for morphological processing of Turkish[1] developed after the Keçi system, prove to be more comprehensive in the processing of Turkish.

Studies by Oflazer (1994) and Oflazer et al. (1994) examine the first application of two-level morphology on Turkish, which is originally based on the implementation of 'Kimmo approach' by Karttunen (1983) and Koskenniemi (1984). In their work, Karttunen and Beesley (2005) describes the two-level morphology system as "*a new way to describe phonological alternations in finite-state terms*". With KIMMO-style two-level morphology, morphophonological alternations seen in highly concatenative languages are simulated via rule specifications that work simultenaously.

By taking advantages of the two-level morphology, the system implemented by Oflazer et al. (1994) describes rule-based processing of morphosyntactic and morphophonological phenomena of Turkish. Because the word formation process is done by concating suffixes to the stem, suffix concatenation can often lead to relatively long words (which are frequently equivalent to a whole sentence in En-

[1]For example, Two-level description of Turkish morphology Oflazer (1994)

glish). However, due to the strict word formation (morpho-syntactic) constraint definitions and regular-expression like nature of the two-level system, such word formations allow Turkish to be modeled with FSMs. The example below, given in Oflazer (1994), shows the word formation of a relatively long word:

OSMANLILALAŞTIRAMAYABİLECEKLERİMİZDENMİŞSİNİZCESİNE[1]

This word can be broken down into morphemes as follows:

OSMAN+ LI+ LA+ LAŞ+ TIR+ AMA+ YABİL+ ECEK+ LER+ İMİZ+ DEN+ MİŞ+ SİNİZ+ CESİNE

For the two-level description of Turkish, the two-level system designed by Oflazer (1994) has been implemented based on a root word lexicon of about 23,000 roots words. The morphophonological alternation rules of Turkish have been created using 22 two-level rules. For the word formation process of concatenative word structures, finite-state machines have been implemented for the verbal, nominal and other part-of-speech category paradigms. This example above shows the complex morpho-syntactic word formation process in Turkish. Morphemes attached to a root-form can change the surface word form's part-of-speech from a noun to a verb, or vice-versa; or as in the example above, it can create adverbial clause structures.

As illustrated with the examples in the previous chapters and sections also, the surface *(final)* representations of morphological structures are not only constrained, but also changed by several morphophonological rules affected by both the preceding syllables and the next syllables in the word structure. Therefore, vowels and consonants in the morphemes attached to root forms or preceding morphemes have to agree with the preceding vowel *(and / or consonant in the preceding syllable)* in order to comply with the vowel harmony rules, or other sound alternations. The two-level morphology description of Turkish achieves a description and application of such complex syntactic and phonological phenomena for the morphological processing of Turkish text.

In a different study, for the rule-based morphological processing of Turkish by Eryiğit and Adali (2004), a different approach has been adopted. In their affix-stripping method, Eryiğit and Adali (2004) used no lexicon. As opposed to root-driven morphological analysis approaches, according to this method with

[1]The symbols for '+' in this example indicate morpheme boundaries. This adverb translates into English as "*As if you were of those whom we might consider not converting into an Ottoman.*"

affix-stripping approach, search and find method is applied for affixes first before the word stem of the word is determined. After removing the suffixes, the remaining part(s) of the words are considered to be the stem.

In their study, Eryiğit and Adali (2004) claim that the advantage of using this approach is the fast search for affixes, compared to the root-driven approaches which determine the stem of the words at the cost of a whole lexicon search. In the root-driven method, first the examination of word morphemes is done by deleting characters one by one from the end of the words, and then by making a list of potential stems, which are checked against the list of words in a lexicon consisting of stems only. As a result, the process of determining the stem becomes highly time-consuming.

For the illustration of their own method, the example taken from Eryiğit and Adali (2004), shows the process of affix-stripping applied to a word to find the word stem.

Given the words for `kale` *(castle)* and `kalem` *(pencil)*, all possible analyses of the word 'kalem' are the following:

i. kalem → N (kalem) *(pencil)*

ii. kale-**m** → N(kale)+1PS-POSS(m) → kalem *(my castle)*

Determining the correct analysis of 'kalem' requires the context-in-use in the sentence. However, for the plural word formation as in 'kalem' → 'kalem-ler', we see only one valid analysis:

a. kalem-ler → N(kalem)+PLUR(ler) → kalemler *(pencils)*

b. ~~kale-m-ler → N(kale)+1PS-POSS(m)+PLUR(ler)~~

Therefore, this approach claims that suffix concatenation rules *(and thus, the right-to-left affix-stripping)* help finding the correct stem because the potential analysis cannot be "kale + m + ler" because in Turkish, a plural suffix cannot follow a possessive one *(as it is also shown by the first example (ii.), where the 1PS-POSS suffix attached at the end of the word).*

Another morphological analyser and multi-purpose tool, called *Zemberek, an open source NLP framework for Turkic Languages*, implemented by Akın and Akın (2007) aims to provide a main NLP framework for all Turkic languages, including Turkish. Based on a root-words dictionary, and a separate file containing irregular phonemic and word formation changes via rules, *Zemberek* uses

dictionary-based parser in order to find the root forms of the words. Via a Direct Acyclic Word Graph (DAWG) tree, which enables easy access and extensibility of the word forms; suffix attachment is applied to the root forms of the words after the root-form search and find operation is completed.

In Zemberek, the implementation of DAWG tree, which functions as a dictionary, provides three root-form selections. These root-form selections perform the following tasks:

i. The first, used for normal strict root selection for an input word form. For example, for the word form "elmaslar", it finds all potential root forms from the set as the following:
- el- → noun *(hand)*
- elma- → noun *(apple)*
- elmas → noun *(diamond)*

ii. The second selection, done via a string similarity algorithm, applies a tolerance level to select candidate root-forms, and generates more root-form candidates than the strict root-form selector used in (i).

iii. The third selector applies a tolerance for non-ASCII letters.

The DAWG tree and word root-form selectors function independently from the language implementation. Akın and Akın (2007) defines the structure of the morphological parser as a root-driven dictionary based, top-down parser. The main steps involved in the morphological analysis of an input word are described as following:

i. *Preprocessing Task*: Preparing tokens by removing accents, hyphens etc. and converting it to lower-case. If the word contains characters that cannot occur in the defined alphabet, morphological parsing operation ends.

ii. If pre-processing step is successful, then, using one of the three root-form selector *(as described above)*, root-form candidates for the input word are found.
- For each root-candidate, possible suffixes are attached to the root, and necessary morphophonological and word-formation rules are applied—until either the surface form of the input word is obtained, or there are no more suffix alternatives left.

Among the other rule-based and finite-state morphological analyzers, *Open source multi-platform NooJ*[1] *for NLP*, by Silberztein et al. (2012) provides a

[1] `http://www.nooj4nlp.net/pages/introduction.html`

graph-based corpus processor, NooJ implementation. The underlying implementation is primarily based on the Augmented Transition Networks first introduced by Woods (1970), Bates (1978) and Shapiro (1982). In previous studies, the use of ATNs was also investigated for Turkish morphological analysis by Güngör and Kuru (1993), and (Güngör, 2003). Nooj by providing graphical tools for modeling and the simulation of word-formation in Turkish, it is also used to access large texts and process corpus-like texts simultaneously. However, Nooj architecture also relies on the use of pre-made dictionaries *(with limited number of lexical item tokens)* to be used as inputs in the finite-state-transducers. In Nooj, for the lexical analysis of Turkish[1], four different classification of *Atomic Linguistic Units (aka. ALU)*—which are considered as lexical items are applied:

i. *Simple-words lexicon:* for words without any affix, such as *'table, chair'*, defined similarly to the way it was described for Turkish examples of *simple words*[2].

ii. *Affix lexicon:* for complex word forms, as in the word *'constitutional'*, defined similarly to the way word formations were described for Turkish *complex-words*[3].

iii. *Multiword Units:* for non-compositional multiword expression words, as in *'round table'* for "meeting".

iv. *Frozen Expressions:* for compositional multiword expression words, as in *'take ...into account'*.

NooJ is designed to run on MS-Windows, using the .NET platform. However, with extra effort, and via installation of MONO and GNU projects that provide the .NET framework for other platforms, NooJ is usable on Linux and the Mac OS X platforms as well.[4]

3.2 Statistical Methods in Morphological Processing

Most of the the statistical methods applied to morphological processing of Turkish, come from attempts to solve problems with morphological ambiguity–which directly relates to morphological processing. Some of the notable research in this

[1] `http://nooj4nlp.net/pages/turkish.html`

[2] As described earlier on page 10, see 'Simple words'.

[3] As described earlier on page 11, see 'Complex words'.

[4] For more info, see: `http://nooj4nlp.net/pages/technology.html`

area defines and describes the problem of morphological disambiguation and morphological processing conjointly.

Previous work in morphological disambiguation attempts to alleviate problems resulting from data sparsity problem. Data sparsity, because of the agglutinative and productive nature of Turkish, makes it harder to define common patterns frequently seen in large texts, which is highly needed for statistical methods, and it often poses as a serious challenge for researchers. In order to solve the problems with data sets *(and data sparsity)*, in their work Hakkani-Tür et al. (2002), attempt to handle the issue by breaking down the morphosyntactic tags of tokens into inflectional groups *(aka. IGs)*, so that IG patterns can be better observed—rather than attempting to define each uniquely assigned morphosyntactic feature of tokens.

In this approach, each of the grouped IGs contain inflectional features[1], which include each of the intermediate derivations leading to the word-formation of the token in analysis. In order to design this statistical method, Hakkani-Tür et al. (2002) used the two-level morphological analyzer by Oflazer (1994), described in the previous section. Using the morphological analyses obtained from the analyzer, they developed four different n-gram models, assigning a probability for each morhosyntactic tag by considering statistics over the individual inflectional groups and surface roots in trigram models. In their baseline model, they built a trigram tag model, representing the distribution of morphological parses given the words, by using a hidden Markov model *(aka. HMM)*. As a result of their work, they found that the simplest of the four n-gram models, which ignored the local morphosyntactic features, gave the best results in accuracy.

Following this n-gram model, a contrasting approach was taken by Yüret and Türe (2006) attempting to solve the morphological disambiguation problem by using a hybrid approach that is partially rule-based and partially statistical. In their work on *Learning morphological disambiguation rules for Turkish*, they describe and discuss a rule-based model with supervised training for morphological disambiguation of Turkish. For the generation of rules, they used a decision list learning algorithm. For the disambiguation of lexically ambiguous words, they trained a separate model for 126 morphological features, known to the morphological analyzer by Oflazer (1994), which they used in their experiment.

In contrast to the use of Inflectional Groups *(IGs)* as morphosyntactic fea-

[1]Inflectional feature groups are separated by derivational boundaries, marked by ^DB in each IG.

tures, which was commonly adopted in other experiments, in this work Yüret and Türe (2006) attempted to solve the data sparsity problem by considering each individual morphological feature separately *(instead of in the groups of IGs, which is a more unifying approach)*. Claiming that even if the number of possible tag combinations may be infinite, the number of morphological features remain within a finite set of features, and the number of features recognized by the morphological analyzer was 126. Considering these aspects, they trained a model where they used the subsets of training, for each unique feature `f`, in which one of the parses for each instance had an example of that unique `f`eature. Based on the presence or absence of the feature `f` in the parses, they used the examples to learn rules via the Greedy Prepend Algorithm (GPA), which is a decision list learner algorithm.

The prediction of correct tags of unknown words was performed first by generating all the potential morphological analyses of words via the morphological analyzer, and then by using the decision lists to check for the presence / absence of a feature. The final results were probabilistically computed by considering the accuracy of each decision list to choose the best parse among the list of parses. As a result of using decision lists via the GP algorithm, they were able to reach a final tagging accuracy of 96% on manually-annotated test set. Their results also showed that use of full-tags *(as it was promoted by the use of IGs)* trained with a single decision list—*rather than separate models for each feature*—had 91% accuracy, compared to the 96% accuracy obtained by their model.

Following this hybrid method for the morphological disambiguation of Turkish, Sak et al. (2007) uses a purely statistical approach, describing a new method for *Morphological disambiguation of Turkish text with perceptron algorithm*. In their work, they examined the efficiency of morphological-ranking with perceptron algorithm for the task of morphological disambiguation. Similar to the previous work on statistical morphological disambiguation by Hakkani-Tür et al. (2002), for the representation of morphological tags, Sak et al. (2007) also used the inflectional-groups *(IGs)* obtained from the morphological analyzer by Oflazer (1994)–which was also used in the previous study.

Complementing their study on Morphological disambiguation of Turkish text with perceptron algorithm, (Sak et al., 2009) published another work, which designed a *Stochastic Finite-State Morphological Parser for Turkish*. As different from the previous work in statistical methods for morphological processing, in their work, this time Sak et al. (2007) built their own morphological parser, adapted from the two-level morphology formalism of Koskenniemi (Koskenniemi, 1984), again by using the two-level phonological and word-formation rules bor-

rowed from the original two-level morphological analyzer by Oflazer (1994). They compiled a new root-lexicon of 55K words borrowed from the Turkish Language Institution dictionary, and used the OpenFst weighted finite-state transducer library (Allauzen et al., 2007) for implementation of finite-state operations and running the morphological analyzer. In order to overcome problems commonly seen in their previous work on morphological disambiguation (Sak et al., 2007), this time they took advantage of the morphological disambiguator, which they had already implemented using the averaged perceptron algorithm. For the implementation of the parser with a probabilistic finite-state-transducer, they constructed the probabilistic FST as the composition of morphophonological and the morphosyntactic transducer.

For the estimation of parameters in this probabilistic finite-state transducer, they used the general Expectation Maximization *(aka. EM)* algorithm, according to the description given in Eisner (2002). In this description, the method of *expectation semiring*—a bookkeeping trick—was used in order to compute the expected number of traversals of each arc in the **E**xpectation step. In the **M**aximization step, for the re-estimation of arc probabilities, the arc probabilities were normalized at each state *(so that the arc probabilities from each state is proportional to the number of the expected number of traversals of each arc).*

As a result of this combination of methods using OpenFst with semiring method and Markovian finite-state transducer, and the combination of the averaged perceptron algorithm for the task of morphological disambiguation, Sak et al. (2009) cites that their disambiguation system achieves 97.05% disambiguation accuracy on the test set.

3.3 TRmorph: A Turkish morphological analyzer

TRmorph, by Çöltekin (2010), is morphological analyzer that uses finite-state transducers in its implementations. It has been developed for the morphological analysis of Turkish primarily, by using a lexicon based on the Turkish spell-checker Zemberek[1] by Akın and Akın (2007). However, the flexibility in its implementation also makes it adaptable to other Turkic languages. TRmorph tool was initially implemented as a two-level morphological processor, using SFST system, aka. Stuttgart Finite-State Transducer, which is a *C++* transducer library[2]

[1] `https://code.google.com/p/zemberek/`

[2] `http://www.cis.uni-muenchen.de/~schmid/tools/SFST/`

(Schmid, 2005).

Capable to adaptation to different back-end libraries, besides the main SFST compiler, TRmorph could also be used with HFST libraries, aka. Helsinki Finite-State Transducer Technology[1] (Lindén et al., 2013), which is mostly targeted at the morphological analyzers based on *weighted and unweighted* FST constructions, mainly used for morphologically rich languages such as Finnish and several others.

The main TRmorph distribution, using the SFST back-end library, was based around three underlying structures:

i. *Finite state machinery* that was written using regular expression syntax *(using xfst syntax).*

ii. *Two-level grammar* that enabled alternations both for word formations and for morphophonological changes.

iii. *Lexicon* that consisted only of the `ROOT` forms of the words.

This version of TRmorph was distributed with two different sets of lexicons—one smaller lexicon prepared during development phase, and another lexicon adapted with some corrections and modifications from the Zemberek spell-checker—made of 1500 and 37101 words each. Both lexicons labeled the lexical items into nine parts-of-speech categories: *adjectives, adverbs, conjunctions, interjections, nouns, postpositions, pronouns, proper names and verbs.*

After several ongoing changes to the underlying structure and utilities of the Trmorph analyzer, and lexicon; TRmorph has switched to using the open-source Foma compiler[2] and finite-state library(Hulden, 2009), which is another *C* programming language library, providing a *'multi-purpose finite-state toolkit'*. While the Foma compiler is still similar to the SFST implementation in many ways; switching to the Foma compiler brought some other benefits. For example, Lindén et al. (2013) evaluated the HFST Toolkit by using the HFST 3.3.4. Toolkit version, for Finnish, German, Italian, Swedish, and Turkish morphologies with different HFST back-end libraries such as SFST, OpenFst[3], and Foma. They represented their findings according to the compilation times shown in minutes and seconds, by averaging over 10 compilations. Their findings show that among these three HFST back-ends, the Turkish morphology had the best performance

[1] `http://www.ling.helsinki.fi/kieliteknologia/tutkimus/hfst/`

[2] `https://code.google.com/p/foma/`

[3] `http://www.openfst.org/twiki/bin/view/FST/WebHome`

by using the Foma library, achieving a performance of 0:05 seconds over 10 compilations, using morphology taken from TRmorph. The second best performance, 0:12 seconds, was achieved by using the SFST library; and as last OpenFst library achieved a performance of 0:40 seconds.

3.3.1 Lexicon in TRmorph Baseline.

After this gradual switch to the Foma compiler, the TRmorph lexicon has also been modified. The comparison of the lexicon used in SFST implementation of TRmorph, and the lexicon used in the Foma compiler version—which we use as our *baseline*[1]—is shown in the following table: With the Foma version of

PoS	Count
Adjective	1244
Adverb	483
Conjunction	47
Interjection	131
Noun	23101
Postposition	36
Pronoun	21
Proper Noun	9532
Verb	2488

Table 3.1: PoS Distribution of ROOT Forms in TRmorph-SFST

PoS	Count
Adjective	1403
Adverb	430
Conjunction	113
Interjection	416
Noun	15230
Postposition	111
Pronoun	94
Proper Noun	10050
Verb	2088
**Onomatopoeia*	54
**Reduplication*	11
**Determiner*	37

Table 3.2: PoS Distribution of ROOT Forms in TRmorph-Baseline(Foma)

TRmorph, three more part of speech forms— onomatopoeia, reduplication, and determiner—were added to the TRmorph implementation. At the same time, while some of the part of speech form numbers increased, such as the number of conjunctions, interjections, postpositions and pronouns; some of the part of speech form numbers, such as the noun and verb lexicon, were decreased due to previous redundancy. In this thesis, we aim to expand the lexicon, and thus, expect these numbers to change again.

[1]In our baseline, we use the version of TRmorph, before the latest TRmorph version, dated 2013-10-13, was released.

3.3.2 Processing of Nominal Word Formations

In Çöltekin (2010)'s work for TRmorph, co-occurrence of nominal suffixes attached to nominal word roots are defined as following:

i. A nominal stem may be followed by plural suffix -lAr.

ii. Next possible suffix that can attach to a nominal is one of 6 possessive suffixes, denoting first, second and third person that is either singular or plural.

iii. Next possible suffixes are six of the common nominal cases: accusative -(y)I, dative -(y)A, locative -DA, genitive -(n)In, ablative -DAn, and instrumental -(y)lA.

iv. Nouns with a locative or genitive case can be followed by the suffix *-ki*, and then all the nominal suffixes can be attached again.

v. All nominals can take a verbal person agreement to form nominal predicates.

vi. A nominal predicate that has a verbal person agreement can have an optional copula or the generalizing modality marker –DIr[1].

vii. The nominals consist of a large number of part-of-speech categories including nouns, adjectives and adverbs. In TRmorph specification, only nouns are allowed to take the nominal suffixes. In order to make adjectives and adverbs nominalized, all adjectival and adverbial stems become a noun by a zero derivation.

According to Eryiğit et al. (2008), due to the agglutinative morphology of the language, Turkish nouns can have around 100 inflected forms, and verbs even more. Because Turkish word formation is done through very productive derivations, the number of possible word forms that may be generated from the root-form of the verb increase significantly. Therefore, it is usually very common to find up to four or five derivations in a single word form.

3.3.3 Processing of Verbal Word Formation

In the TRmorph distribution, verbal word formation—which is more complex than nominal word formation process—is described as follows:

i. Verbal roots can take a number of voice suffixes, the negative marker, a number of sufixes that form compound verbs.

[1] Although, usage of this kind of nominal predicates is not common in informal writing and spoken Turkish

ii. The passive suffix has two forms, predictable from the preceding context, and handled by two-level rules, while negation marker precedes all other inflectional suffixes[1] And, +COMP marker is used to represent 8 suffixes to form compound verbs, expressing modality.

iii. The two of the most productive of these modality suffixes are -(y)Abil and -(y)Iver. This compound/modality suffix -(y)Abil becomes -(y)A when it occurs before the negative marker and only in negative forms. After compound suffixes, causative suffix and other suffixes that form a compound verb form can be attached.

iv. For finite verbs, after these optional suffixes, suffixes for person agreement and tense/aspect/modality (TAM) marker are used. Following these obligatory person agreement and TAM markers, several other optional suffixes, such as copular markers and the generalizing modality marker -DIr can be attached.

v. Finite verbs can have one of the 11 TAM markers, together with the person agreement. In finite verb formation, use of one of these markers is obligatory. In some of the tense or modality markers, there are some irregularities. For example, the aorist tense morpheme is irregular when it is directly attached to the root forms, and this must be specified in the lexicon because it is unpredictable.

vi. After T/A/M markers, in finite verbs, there are usually two options. Either, one of the three copular markers forming complex tenses can follow, and after the copula, person agreement attachment is mandatory; or one of the six person agreement morphemes followed by optional generalizing modality marker, -DIr can follow the sequence of morphemes *(The copular markers rely on the context of preceding T/A/M marker, and type of the person agreement depends on the preceding copula.)*

vii. As one of the other irregularities, reflexive and reciprocal suffixes show some unpredictability when attached to a small number of verbs. Therefore, those verbs that can become reflexive, or reciprocal, or both are marked on the lexicon as well.

viii. In the same way, causative verb forms show irregularity as well, by combining with one of the six forms depending on the root, or it usually between -DIr and -t according to the preceding letter. Irregularities of these causative verb forms are again dealth with by marking them in the lexicon.

[1]The negative marker -mA becomes -mI before the suffix -(I)yor, this exception is handled via morpho-phonological rules.

ix. 'Untensed' verbs can be nominalized by a number of subordinating suffixes, forming verbal nouns, participles or converbs[1]. These nominalized verb forms can take most, if not all, of the nominal inflections.

3.3.4 Additional Utilities in TRmorph-Extended Version

Following additional utilities have been added to the TRmorph analyzer much later after we evaluated it as our baseline[2]. In order to give full credit for the developer; and to compare the differences between these two different versions; here we briefly introduce recently added analyzer utilities and explain how they affect the evaluation of the data sets whenever the evaluation is possible and/or necessary.

Stemmer Utility

Additional utility tools that are provided with the updated version of TRmorph work similary to the automaton implementation of the analyzer. The stemmer automaton is defined as `stemmer.fst`, which is a binary file, created using the `makefile` that is included in the main distribution of the TRmorph package. The stemmer automaton produces the lexical root-forms of the word forms as the analysis once the surface form of a word is entered for the query. Additional options to keep the first tag of the stem as the syntactic category word[3], or select to mark the verbs with their infinitive suffix *-mak*—which is the general dictionary form of the verbs—can also be set from the configuration settings[4]. Some basic examples are as following:

```
foma[0]: regex @"stem.fst";
1.0 MB. 29400 states, 68377 arcs, Cyclic.
foma[1]: up
apply up> okumuşlar     ('read-PAST_narr-3p')
oku<V>     ('(to) read')
apply up> evlerinden  ('house-Poss3p-Loc')
ev<N>('house')
apply up> miyavlamış  ('meow-^D_lA-PAST_narr(3s)')
miyavla<V>('(to) meow')
```

[1]Non-finite verb forms, acting similar to verbs but functioning as adverbial subordination

[2]Version dated as 2013-10-13

[3]The stemmer automaton takes the lexical form as the 'stem', even if the final word form ROOT may be different from the stem due to derivational suffixes immediately following the root form.

[4]These options are handled from the file `options.h` in the main directory of the TRmorph distribution.

Unknown Words Guesser Utility

The TRmorph guesser implementation works in the similar way as the analyzer works. The guesser transducer accepts an FSA for lexical items with minimal restrictions *(based on the definitions of symbols—including all Turkish character letters that appear in Turkish text—used in the lexc compiler)* instead of restricting the query words to the list of stems that need to exist in the fixed-lexicon. This way the handling of the unknown words is managed based on the suffixes whose morphological properties are already defined in the `morph.lexc` settings, and the guesser transducer returns the surface strings of unknown words with analyses of potential word candidates depended on two options specified by the user:

i. Either the full analysis of strings, including their potential unknown roots that may result in the final surface form of the words.

ii. Or, only the root word and its part of speech tag.

With the current Foma implementation of the TRmorph analyzer, the only restriction of the guesser is the minimum and maximum root-word length which can also be set from the configurations settings. Similar to the implementation of the analyzer FSA, the guesser transducer was implemented as a standalone automata. Therefore, in order to use the unknown word guesser together with the analyzer, the two automata—separate transducers for the analyzer and the guesser—is used with a priority union so that the guesser is activated only when the analyzer fails. For this, there are two options available, which enable the guesser to be used either as a simple wrapper xfst file[1], or by using foma's *flookup* command, specifying both transducers on the command line[2]. Here are two examples showing how to activate the guesser:

1. In order to use the guesser by itself:

```
foma[0]: regex @"guess.fst";
13.2 MB. 41753 states, 862831 arcs, Cyclic.
foma[1]: up
apply up> biçimbirimsel (‘morphologic-al’)
biçimbirim<N:abbr><sal><Adj>
biçimbirim<N:abbr><sal><Adj><0><N>
biçimbirim<N:abbr><sal><Adj><0><N><0><V><cpl:pres><3p>
biçimbirim<N:abbr><sal><Adj><0><N><0><V><cpl:pres><3s>
biçimbirim<N><sal><Adj>
```

[1]First by compiling the `foma`, and then typing `‘source guesser.xfst’, or ‘regex @"guess.fst";’`

[2]Such as: `flookup -a guess.fst trmorph.fst`

```
biçimbirim<N><sal><Adj><0><N>
biçimbirim<N><sal><Adj><0><N><0><V><cpl:pres><3p>
biçimbirim<N><sal><Adj><0><N><0><V><cpl:pres><3s>
...
...
...
biçîmbîrim<N><sal><Adj>
```

2. Using foma's `flookup` command, activating the guesser together with the analyzer, and to batch-process text files:

```
TRmorph$ flookup -a guess.fst trmorph.fst < test-conll-unknown.txt

biçimbirimsel +?

biçimbirimsel biçimbirim<N:abbr><sal><Adj>
biçimbirimsel biçimbirim<N:abbr><sal><Adj><0><N>
biçimbirimsel biçimbirim<N:abbr><sal><Adj><0><N><0><V><cpl:pres><3p>
biçimbirimsel biçimbirim<N:abbr><sal><Adj><0><N><0><V><cpl:pres><3s>
biçimbirimsel biçimbirim<N><sal><Adj>
biçimbirimsel biçimbirim<N><sal><Adj><0><N>
biçimbirimsel biçimbirim<N><sal><Adj><0><N><0><V><cpl:pres><3p>
biçimbirimsel biçimbirim<N><sal><Adj><0><N><0><V><cpl:pres><3s>
...
...
...
```

Note that, in the second example, we see that once the word form "biçimbirimsel" is returned as unknown denoted by `+?` symbols, the guesser transducer is invoked, and all the potential analysis are returned.[1]

Segmenter Utility

The TRmorph tool comes with a separate automaton—a binary file, `segment.fst`—which finds the morpheme boundaries on the surface strings. The segmentation transducer depends on the root and morpheme boundaries on the surface strings which are deleted during the regular analysis processing. Segmenter works by first analyzing the query words through the regular analyzer, and then passing them through the segmenter transducer in generation mode, without deleting the morpheme boundaries from the surface form of the word. During this process,

[1] Due to space constraints, we did not list all the analyses of the word 'biçimbirimsel' here.

certain word forms might produce multiple and identical segmented forms due to vowel and consonant harmony rules described in the previous chapter—since the environment affecting the morphophonology might come both from the preceding and following vowels and consonants. Here is an example of a noun that is segmented from its morpheme boundaries:

```
foma[0]: regex @"segment.fst";
2.9 MB. 59573 states, 190562 arcs, Cyclic.
foma[1]: up
apply up> evlerinden
ev-leri-nden
ev-leri-nden-ler
ev-ler-in-den
ev-ler-in-den-ler
ev-ler-i-nden
ev-ler-i-nden-ler
ev-ler-i-nden
ev-ler-i-nden-ler
ev-ler-i-n-den
ev-ler-i-n-den-ler
ev-ler-i-nden
ev-ler-i-nden-ler
ev-leri-nden
ev-leri-nden-ler
ev-i-nden
ev-i-nden-ler
```

Token-Hyphenator Utility

Hyphenated words are split from the syllable boundaries in Turkish. Due to regular syllable structure in the spelling, there is no special treatment required. A transducer for hyphenated words is created via the `Makefile`, and produced as a binary file called `hyphenate.fst`. The resulting analysis while using this transducer forms surface strings of words with hyphens placed between the syllables. For example:

```
foma[0]: regex @"hyphenate.fst";
10.6 kB. 12 states, 551 arcs, Cyclic.
foma[1]: up
apply up> okudum (‘read-PAST-1s’)
o-ku-dum
apply up> okudularsa (‘read-PAST-3p-Cond’)
```

```
o-ku-du-lar-sa
apply up> okuduysalar ('read-PAST-Cond-3p')
o-ku-duy-sa-lar
```

Note that in the last two examples, essentially both of the word forms refer to the same thing, as in *'(if) they read'*. However, there are two different ways to form such conditional word forms because the morpheme-order of the '3p' suffix `-lAr` can vary, without losing its meaning. In such cases, the hyphenator still needs to be able to correctly segment the morphemes from the syllable boundaries.

Evaluation of the current architecture in TRmorph

The following chapter examines the evaluation of the TRmorph-Baseline analyzer *(as well as the TRmorph with its additional utilities wherever it is necessary)* on various data sets—consisting of both annotated and unannotated text such as Turkish articles and newspaper texts—including the Turkish Treebank and Turkish Corpus (Oflazer et al., 2003), (Atalay et al., 2003), as well as Morpho Challenge Shared Task[1] 2010 data sets (Kurimo et al., 2010a). It assesses the coverage of these data sets analyzed by the TRmorph-Baseline version, as well as TRmorph-Extended version. At last, it provides evaluation analyses *(Morfessor Reference evaluation results whenever possible)* for the data obtained from Morpho Challenge Task.

[1] `http://research.ics.aalto.fi/events/morphochallenge2010/`

Chapter 4

Evaluation of TRmorph

Data Sets Used in the Evaluation of TRmorph

In order to assess how well the TRmorph analysis covers annotated data set word forms, as well as unannotated text words; we processed several data sets for the assessment of coverage. Among these data sets, we processed the following:

i. Unannotated news paper text.

ii. Unannotated data sets taken from Web-To-Corpus—collected from Wikipedia and web text by Majliš (2011), and Majliš and Žabokrtský (2012).

iii. METU-Sabanci Treebank data sets (Oflazer et al., 2003), (Atalay et al., 2003)—used in CoNLL–Shared Task on Dependency Parsing (2007), (Nilsson et al., 2007).
– METU Turkish Corpus data sets (Say et al., 2002).

iv. Annotated data sets obtained from Morpho Challenge 2010, Semi-supervised and Unsupervised Analysis Shared Task[1] (Kurimo et al., 2010b).

In the next sections, we examine the coverage of these texts processed with the TRmorph analyzer and discuss the evaluation metrics on the gold standard data sets taken from the Morpho Challenge shared task.

4.1 Newspaper Data Coverage

In order to process tokens that are more commonly seen in daily usage, both formally and informally, and to avoid linguistically motivated bias that may exist in annotated tagsets; we processed unannotated newspaper text, collected from

[1] `http://research.ics.aalto.fi/events/morphochallenge2010/`

Milliyet newspaper articles[1]. For coverage assessment of this newspaper text, we used the TRmorph-Baseline analyzer[2]. In order to prepare these data sets for the morphological processing, we first applied some basic pre-processing steps by filtering out tokens such as punctuation marks, and mark-up tags, and new lines—which do not contribute to the morphological analysis. After the pre-processing step, we obtained the following coverage counts:

Milliyet Texts	All Tokens	All Unknown Tokens	Token Coverage
Milliyet1	5028083	283912	0.943534743
Milliyet2	5016963	283962	0.943399622
Milliyet3	5010526	297896	0.940545963
(Avg.) TOTAL	15055572	865770	0.942495044

Table 4.1: TRmorph-Baseline token coverage of Milliyet Newspaper

Milliyet Texts	All Uniq.Tokens	Uniq.Unkn.Tokens	Type Coverage
Milliyet1	351238	75857	0.784029632
Milliyet2	352296	74245	0.789253923
Milliyet3	324246	79133	0.755947645
(Avg.) TOTAL	1027780	229235	0.776961023

Table 4.2: TRmorph-Baseline type coverage of Milliyet Newspaper

The next table 4.3 shows the distribution of unknown words according to their initial letters. Observations obtained from these distributional word counts point out that unknown words starting with a lower-case letter, rejected by the analyzer, consist of words that have either a foreign or borrowed origin—thus, their root-forms do not exist in the fixed lexicon of TRmorph-Baseline version. A second majority of unknown words come from tokens that have been misspelled, or words that have unconventional spelling. The other group of unknown words, starting with an upper-case letter, rejected by the analyzer include mostly proper nouns, and noun phrases, which are generally a group of Named Entities (NE).

[1]Special thanks to Prof. K. Oflazer for giving us permission to use the Milliyet Data Collection. The set of collections *(in raw-text format)* can be accessible from:http://deniz.yuret.com/turkish/Milliyet1.bz2, http://deniz.yuret.com/turkish/Milliyet2.bz2, http://deniz.yuret.com/turkish/Milliyet3.bz2

[2]without the guesser for unknown words

Unique Words	Word Count
All Unique Words	214385
Tokens starting with uppercase	148803
Tokens starting with lowercase	63088
Others (Punct, Digits)	2494

Table 4.3: Distribution of unknown words according to their capitalization in Milliyet newspaper

The frequency distribution of the top ten lower-case and upper-case unknown words obtained from these lists can be shown as the following:

Frequency	Lowercase Words
106	ın
102	ilerde
101	t
100	livaneli
10	açıkhava
10	anakent
10	aprona
10	arttırarak
10	arttırdığı
10	arttırmayı

Frequency	Uppercase Words
100	Amokachi'nin
100	Aluminyum
100	Alcatel
100	Aksigorta
100	Bakanı'nın
100	Banvit
100	Bir'in
100	Borova
100	Çimentaş
100	Demisaş

Table 4.4: Frequency distribution of top 10 unknown words

From the examples given in table 4.4, we can easily see that noun phrases, proper nouns and Named Entities in Turkish are not only used with upper-case initials; but also they use an apostrophe to separate nominal suffixes attached to the stem. Relying on these assumptions, we can identify Named Entities. Intuitively, we may also conclude that if the NE types are annotated according to certain NE characteristics, such as *person, location or organization*, or a given analyzer can detect such patterns; certain unknown tokens carrying the patterns of NEs can be correctly analyzed, rather than being rejected from the analyzer, due to *unknown* word-forms (or stems). In the next sections, for the processing of unannotated data sets, we have continued using only the TRmorph-Baseline version *(without the unknown tokens guesser utility in TRmorph-Extended version).*

4.2 Web-To-Corpus (W2C) Data Coverage

For the coverage assessment of unannotated data sets, we have used Web-2-Corpus data sets prepared by Majliš (2011) and Majliš and Žabokrtský (2012). These data sets consist of a collection of texts extracted from Turkish Wikipedia which contains around 12.5 million words; and other unannotated web text collected via web crawling of various text resources, which contain around 100 million words[1]. Since these kinds of data sets contain both formal and informal (and noisy content, mostly unedited text) language that is commonly used on the internet; we evaluated these data sets to see how well the TRmorph-Baseline analyzer processes such data sets. Before the coverage assessment, we pre-processed the Wikipedia text by removing certain tokens—such as the punctuation, empty lines—that are irrelevant to the analysis of tokens. Table 4.5 below shows the token coverage of Wikipedia words:

Wikipedia Texts	All Tokens	All Unknown Tokens	Token Coverage
Wikipedia-1	4200000	470376	0.888005714
Wikipedia-2	4200000	545077	0.870219762
Wikipedia-3	4193380	449681	0.892764071
(Avg.) TOTAL	12593380	1465134	0.883658398

Table 4.5: TRmorph-Baseline token coverage on Wikipedia data sets

Based on the number of unique words within these Wikipedia data sets, table 4.6 below shows the coverage of token-types: Table 4.6 below shows the token-type coverage of Wikipedia data sets processed with TRmorph-Baseline analyzer:

Wikipedia Texts	Unique Tokens	Unique Unknown	Type Coverage
Wikipedia-1	422861	199054	0.529268483
Wikipedia-2	449103	222440	0.504701594
Wikipedia-3	395619	217163	0.451080459
(Avg.) TOTAL	1267583	638657	0.496161593

Table 4.6: TRmorph-Baseline token-type coverage on Wikipedia data sets

[1]For more details on how these data sets were collected, see the developer's page: `http://ufal.mff.cuni.cz/~majlis/w2c/`

As it was described for the Wikipedia text pre-processing, before the analysis of texts, we applied the same steps for the coverage of Web-2-Corpus web-text assessment. We give the coverage of the web-text, made of 50 million tokens which were processed with TRmorph-Baseline analyzer in the following table 4.8 below:

W2C-Web Data	Number of Tokens
All Tokens	50000000
All Unknown	5449269
Token Coverage	0.89101
Unique Tokens	2312007
Unique Unknown	1436792
Type Coverage	0.37855

Table 4.7: TRmorph-Baseline token & type coverage on W2C-Web data

It is important to note, however, that the web-text that is extracted from the web is normally not expected to be as clean as edited news-paper or annotated text. Therefore, we see a much lower token-type coverage of the web-text than the one obtained for the Wikipedia source–which is also considered as non-edited text, compared to other more formal types of written text. As a result, we see that a significant number of tokens—that were returned as *unknown* from the web-text—contain highly noisy content that is normally not seen in edited and more formal written texts. The type of tokens that were not analyzed consist of not only misspelled or unconventionally spelled tokens, but also tokens that do not have much morphological content. Approximately 62K of tokens out of 1,4M unknown tokens consist of digits/numbers for currencies, dates, phone numbers or certain digital codes, email addresses and similar kinds of tokens. We give a small list of examples taken from the unknown tokens list below:

01/01/1992	01/01/1993
01011993′den	01/01/2007
01/01/2008	01/01/2009
01Haziran2011	02/Haziran/2009
00005YTL	00007YTL
#0206*8373#	*#0206*8378#
aacun92@hotmailcom	abis@abgsgovtr

Table 4.8: Examples of unknown tokens from W2C-Web text

The rest of the unknown tokens that do not have digits in them also con-

sist highly of misspelled tokens, or multi-word tokens that were not properly tokenized via use of white-space. For a general evaluative assessment of the TRmorph-Baseline, we have obtained these coverage results. However, we will mostly consider the issues seen in annotated or edited *(but unannotated)* texts for the improvement of the TRmorph-Baseline since the parsing of noisy content is out of scope of this thesis, and in general out of the scope of generic morphological analyzers.

4.3 METU–Sabanci TreeBank, CoNLL Data Coverage

The METU–Sabanci TreeBank made from a smaller subset of the METU[1] Turkish Corpus—which is a two million word collection of Turkish text written in post-1990s from 10 genres—consists of 7262 sentences. For the morphological representation, words are represented with IG-based annotation[2], and dependency links showing inflectional IGs on the gold-standard data sets.(Eryiğit et al., 2008)

From METU–Sabanci TreeBank, a smaller subset of data, consisting of 5635 sentences, was prepared for the CoNLL 2007 shared task on dependency parsing. However, especially because the word-internal dependencies[3] in the gold-standard data are represented with underscores (Nilsson et al., 2007); the dependency representation of the IGs[4] can suffer in data processing. This often stems from the formatting types, and annotation differences required by different tools. Sometimes, these dependencies can have more than just one internal dependency link, shown as `DERIV` link. For example, the lines 4-7 in the sentence below, taken from the CoNLL test set, show the usage of multiple dependency links where each IG represents a different part of speech than the part of the speech of the lemma and/or the final word form:

[1]Middle East Technical University,`http://ii.metu.edu.tr/corpus`

[2]Inflectional groups represent the root and derivational elements of a word. They are separated from one another by derivational boundaries. Each IG carries its own part of speech and inflectional features(Eryiğit et al., 2008).

[3]Internal dependencies are shown with dependency links labelled as `DERIV`, which connect IGs of words.

[4]Link between word dependencies and their corresponding morphological tags

1	Ama *(However)*	ama	Conj	Conj	_	12	S.MODIFIER
2	,	,	Punc	Punc	_	0	ROOT
3	hareket *(movement)*	hareket	Noun	Noun	A3sg|Pnon|Nom	11	SUBJECT
4	_ *(expect)*	bekle	Verb	Verb	_	5	DERIV
5	_	_	Verb	Verb	Pass|Pos	6	DERIV
6	_	_	Adj	APresPart	_	7	DERIV
7	beklenenden*(_is_expected)*	_	Noun	Zero	A3sg|Pnon|Abl	10	MODIFIER
8	çok *(very)*	çok	Adv	Adv	_	9	MODIFIER
9	daha *(much)*	daha	Adv	Adv	_	10	MODIFIER
10	sert *(hard)*	sert	Adj	Adj	_	11	MODIFIER
11	oluyor *(become)*	ol	Verb	Verb	Pos|Prog1|A3sg	12	SENTENCE
12	.	.	Punc	Punc	_	0	ROOT

Table 4.9: CoNLL sentence format with derivational IGs– Translation of the sentence: 'However, the movement is becoming a lot harder than what is expected'.

Besides such dependency links showing IGs, for the representation of derivational dependencies *(which are labelled as DERIV in the sentence above, showing the connection of IGs to the words)*, another type of internal word structure represented in the data format poses a challenge for the morphological processing of words. In the CoNLL data sets, multi-word expression tokens are also internally linked together by using an underscore, showing their lexical compositionality. The examples below, which were randomly selected from the CoNLL test set, show how those MWEs are represented in the CoNLL format:

10	yer_alan *(taking_place)*	_	Adj	APresPart	_	11	MODIFIER
15	dolaşıp_durmuş *(keep_wandering)*	dolaşıp_dur	Verb	Verb	Pos|Narr|A3sg	16	SENTENCE
11	fark_etmiş *(made_difference)*	fark_et	Verb	Verb	Pos|Narr|A3sg	12	SENTENCE

Table 4.10: Format of MWE tokens in the CoNLL data sets

In our coverage assessments, we removed certain tokens and lines that are not relevant to the coverage estimations for processing the text by the TRmorph analyzer. These include punctuation tokens, empty lines, and the lines starting with underscores that point to the word-internal dependency links. In this section, we provide the analyses of CoNLL data sets with two different coverage estimations.

i. At first, we give the coverage of CoNLL data sets with the original tokens which include the multi-word expression tokens as shown in 4.10. In our evaluation, in order to process these tokens properly, we have removed linking-underscores[1] between tokens composing that multi-word expression.

ii. Secondly, we provide the coverage of all tokens as single-unit lexical items—where we have split the multi-word expressions into separate tokens by re-

[1]By removing those underscores, we replaced them with regular white-spaces, so that tokens given as `'yer_alan'` can be formatted as `'yer alan'` *('taking place')*

moving the underscores within that MWE. This was done in an attempt to process all the tokens individually so that we can evaluate the performance of TRmorph, based on the number of single tokens without linking-underscores, or spaces.

Table 4.11 below shows the TRmorph-Baseline coverage of original tokens for the analyzed word forms in relation to the *unknown*[1] word forms:

Original CoNLL-Data without punctuation

		Tokens		
Treebank Files	**# of. Sents.**	**All**	**Unknown**	**Token Coverage**
`test.conll`	300	3070	112	0.963
`train.conll`	5635	43573	2247	0.948

Table 4.11: TRmorph-Baseline coverage of `tokens` on METU TreeBank

The table 4.12 below shows the TRmorph-Baseline coverage of original token *types*—which were obtained by calculating the unique word counts in the CoNLL data sets—for the analyzed word forms in relation to the *unknown* word forms:

Original CoNLL-Data without punctuation

		Unique Tokens		
Treebank Files	**# of. Sents.**	**All**	**Unknown**	**Type Coverage**
`test.conll`	300	1865	70	0.962
`train.conll`	5635	19413	1952	0.899

Table 4.12: TRmorph-Baseline coverage of `token-types` on METU TreeBank

Processing Tokens with TRmorph-Extended

The following tables show the coverage of CoNLL data when the TRmorph analyzer was run using the unknown tokens *guesser* utility. Table 4.13 below gives the coverage of tokens processed with TRmorph-Extended version:

[1] 'Unknown' in these tables and for the rest of this thesis refers to tokens that the TRmorph analyzer returns "no analysis".

Processing CoNLL Data–Tokens without punctuation

	Tokens		
Treebank Files	**All**	**Unknown**	**Token Coverage**
test.conll	3070	90	0.970
train.conll	43573	1805	0.958

Table 4.13: TRmorph-Extended token coverage on METU TreeBank

And table 4.14 gives the coverage of types for tokens processed with TRmorph-Extended, using the unknown words guesser utility:

Processing CoNLL Data–Token Types without punctuations

	Unique Tokens		
Treebank Files	**All**	**Unknown**	**Type Coverage**
test.conll	1865	65	0.965147453
train.conll	19413	1540	0.920671715

Table 4.14: TRmorph-Extended type coverage on METU TreeBank

Processing Single–Unit Tokens

The following tables show the TRmorph coverage of *single-unit* tokens—where we have replaced the underscores with the \n character from the MWEs to split them into new lines—for the analyzed word forms in relation to the unknown word forms. Regular coverage of tokens *(processed with TRmorph-Baseline, without using the unknown tokens guesser utility)* is given in 4.15:

Processing CoNLL Data without punctuation

		Tokens		
Treebank Files	**# of. Sents.**	**All**	**Unknown**	**Token Coverage**
test.conll	300	3160	50	0.984
train.conll	5635	46010	929	0.979

Table 4.15: TRmorph-Baseline coverage of single-tokens on METU TreeBank

The coverage of *types* for these single-unit tokens *(processed with TRmorph-Baseline)* is given in table 4.16:

Processing CoNLL Data without punctuation

Treebank Files	# of. Sents.	Unique Tokens All	Unknown	Type Coverage
test.conll	300	1875	23	0.987
train.conll	5635	19112	737	0.961

Table 4.16: TRmorph-Baseline coverage of single-token types on METU Tree-Bank

At a first look, coverage counts obtained from these results seem promising. However, splitting of original multi-word unit tokens into single tokens often led to different PoS tags *(and indeed even different morpheme tags)* being assigned to these single-tokens. Multi-word tokens usually compose a lexical entity that is different from the composition of its individual tokens. Therefore, splitting the tokens—while it increases the coverage—does not necessarily return the correct tag assignment.

Processing Single–Unit Tokens with TRmorph-Extended

Following tables below show the coverage of single-unit tokens—which were previously underscored multi_word_expressions in the CoNLL data format—processed with the TRmorph unknown words guesser. Table 4.17, below, shows the coverage of tokens over the Turkish Treebank:

Processing Single-Unit Tokens without punctuation

Treebank Files	Tokens All	Unknown	Token Coverage
test.conll	3160	1	0.999
train.conll	46010	63	0.998

Table 4.17: TRmorph-Extended coverage on METU TreeBank

Table 4.18 below shows the coverage of types for single-unit tokens processed with TRmorph-Extended over the METU Treebank:

Processing Single-Unit Token Types without punctuation

	Unique Tokens		
Treebank Files	**All**	**Unknown**	**Type Coverage**
test.conll	1875	1	0.999
train.conll	19112	56	0.997

Table 4.18: TRmorph-Extended coverage on METU TreeBank

The way tokens are formatted in existing data sets is important because while the gold standard data sets can employ different annotation tags, different types of NLP processing tools and methods may use different annotation methods which then may cause significant drops in accuracy while testing for accuracy. As described in Eryiğit et al. (2011), the lack of MWE handling in the dependency parsing of Turkish—which exists in the gold standard data sets of the METU-Sabanci treebank, as we have shown above in the examples in 4.10—results in 4% accuracy drop when they test their parser on raw data.

In our earlier coverage assessment of CoNLL tokens processed with TRmorph-Extended (*using the unknown tokens guesser utility*), our findings show that a total number of *4294* unknown tokens[1] correspond to *3663* token-types[2] in the previous sections. Among these unknown tokens, a total of *4099* unknowns consist only of MWE *(including, also, Named Entities—aka. NEs)* tokens; and from those, *1791* of them come from unique MWE types. Table 4.19, below, shows the breakdown of MWE token and type counts existing in the CoNLL data sets:

Data Sets	**Word Counts**
All Unknown Tokens	4294
Unique Unknown Types	3663
All MWE Tokens	4099
Unique MWE Types	1791

Table 4.19: Distribution of MWE tokens on METU TreeBank

Table 4.20, below, shows 10 MWE tokens that were randomly selected from the TRmorph-Baseline unknown token analyses.

[1] As shown in table 4.13 for 'TRmorph-Extended Token Coverage on METU TreeBank'
[2] As shown in table 4.14 for 'TRmorph-Extended Type Coverage on METU TreeBank'

Unknown MWE Tokens	
Bayındır_Sokak'taki	*('on the Bayındır Street')*
belli_olmaz	*('cannot be made certain')*
bilgisayar_mühendisi	*('computer engineer')*
Dışişleri_Bakanı	*('external affairs minister')*
gerek_duyulur	*('feeling the need')*
Hemen_hemen	*('almost, nearly')*
riayet_edilmesini	*('to submit, comply with')*
tahmin_ettiğim	*('making a guess')*
tarihe_geçmiştir	*('taking place in the history')*
şifa_veren	*('healer, remedy giving')*

Table 4.20: 10 random MWE tokens on METU TreeBank

Looking at findings from the coverage assessment, we see that the format of tokens in the source corpus directly affect the way they are analyzed in a morphological analyzer.

4.3.1 METU Turkish Corpus Data Coverage

Çöltekin (2010) reports that tokens existing in the METU Turkish Corpus[1] were processed with TRmorph–*using the earliest released version with SFST compiler*; and they give the following assessment findings shown on the table 4.21

Original		Analyzed	
Type	**Token**	**Type**	**Token**
162724	1431513	88%	95%

Table 4.21: TRmorph coverage on METU Turkish Corpus

For our own coverage assessment of the METU-Corpus, using the TRmorph-Baseline analyzer, we processed the current version of METU-Corpus that was available for researchers at the time of this thesis. The corpus data consist of a collection of approximately two million words taken from written Turkish samples dated as post-1990s. For pre-processing of the data, we have filtered out all the corpus related mark-up tags, and punctuation characters, as well as new lines, which are irrelevant to the processing of word types in the TRmorph analyzer. Table 4.22 below shows our findings for the token coverage, showing the total number of tokens in the corpus and the number of tokens that were unknown to the TRmorph analyzer.

[1] `http://ii.metu.edu.tr/corpus`

Original METU-Corpus Data		
Tokens	**Unknown**	**Token Coverage**
2030984	92859	0.954

Table 4.22: TRmorph-Baseline coverage on METU Turkish Corpus

These coverage results show the current baseline version of TRmorph that uses the Foma compiler, compared to the earliest version of TRmorph that used the SFST implementation for the TRmorph analyzer as given in table 4.21. Based on the unique token count of corpus words, table 4.23 below shows the coverage of token-types analyzed with TRmorph-Baseline:

Original METU-Corpus Data		
Token-Types	**Unknown**	**Type Coverage**
217001	25892	0.880

Table 4.23: TRmorph-Baseline coverage of token-types on METU Turkish Corpus

Our findings from this coverage assessment show that most common unknown words that were not analyzed with the TRmorph-Baseline analyzer, again are either of foreign origin, or proper nouns (person names and other Named Entity types) that do not exist in the fixed TRmorph lexicon.

4.4 Morpho Challenge Shared Task Data Coverage

Coverage of Morpho Challenge Tokens Processed with TRmorph-Baseline Since the Morpho Challenge data sets, part of the Morpho Challenge Shared Task (Kurimo et al., 2010a), are the only gold standard data sets[1] with evaluation assessments available; in this section, we examine the coverage of the Morpho Challenge data set tokens to assess the percentage of word forms analyzed by TRmorph in relation to the percentage of unknown word forms.

For the task of assessing the coverage of the Morpho Challenge data sets, in order to process the tokens with the TRmorph analyzer, we had to apply a character conversion first. For this conversion, we replaced the uppercase letter

[1]The Turkish gold-standard analyses were obtained from a morphological parser developed at Boğaziçi University; based on Dr. Oflazer's finite-state machines. See:`http://research.ics.aalto.fi/events/morphochallenge2010/datasets.shtml#goldstd`

characters—that were originally replaced by uppercase letters[1] of the standard alphabet in the Morpho Challenge data—with regular Turkish characters that use UTF-8 characters as this is required by the TRmorph analyzer.

After the character conversion, we assessed only the data gold standard data sets, consisting only of *word list*, taken as a corpus size of a one million word from the Wortschatz collection (Quasthoff et al., 2006) at the University of Leipzig[2] (Germany), *training set, development* and *development pairs set*, as well as *the combined sets*. Our coverage assessment shows that the word list coverage is significantly lower than the other data sets. This is probably due to the fact that in the Morpho Challenge Shared Task, actual Turkish characters were replaced with their corresponding uppercase variants. Certain words, especially the ones where the lowercase/uppercase dotted/dotless letters *(-ı, -I, -i, -İ)* are used—seem to have been annotated incorrectly in the original data. Thus, those words do not appear in the lexicon, which resulted in many *unknown* analyses returned from the TRmorph analyzer, since they cannot be recognized as legitimate word forms. The table 4.24 below shows the coverage of the tokens processed by the TRmorph, and next table 4.25 shows a list of tokens that were returned as *unknown* due to character conversion:

Data Sets	All Tokens	Unknown Tokens	Token Coverage
Word List	617294	291240	0.528198881
Training Set	1000	43	0.957
Development Set	763	35	0.95412844
Combined Sets	1760	78	0.955681818

Table 4.24: TRmorph-Baseline coverage of Morpho Challenge Shared Task (2010)

The table 4.25 below shows the examples of 15 *unknown* words taken from the Morpho Challenge *Word List*, which were corrupted during the character conversion process:

[1] In the Morpho Challenge tokens, letters specific only to the Turkish alphabet replaced by their uppercase letter equivalents, e.g., "açıkgörüşlülüğünü" spelled as "aCIkgOrUSlUlUGUnU".

[2] http://corpora.informatik.uni-leipzig.de/

Unknown Words in Original Morpho Challenge Sets	Unknown Words with Normalized Chars in TRmorph
CGkabilir	çğkabilir
CGkamazlar	çğkamazlar
CGkar	çğkar
CGkarGmda	çğkarğmda
CGkarGmlarGn	çğkarğmlarğn
CGkardG	çğkardğ
CGkarlar	çğkarlar
CGkarmak	çğkarmak
CGkarmayG	çğkarmayğ
CGkmGStGr	çğkmğştğr
CGkmasGna	çğkmasğna
CGkmaya	çğkmaya
CGktGGGnG	çğktğğğnğ
CGnkU	çğnkü
CI	çı

Table 4.25: Mapping of Morpho Challenge 'word list' tokens to normalized tokens in TRmorph

Besides the unknown words returned from the *Word List* file in the Morpho Challenge data sets, the other most common types of unknown word forms obtained from the *training, development* and *combined* data sets come from the following examples:

> muarrem, öveçler, boşboğazlık, derebeyi, müstacel, semih, canbazın, galatasaraylıyım, müstacel, plus, istanbullular, olumlanan, konyaspor, yargıtayımız, dominantlığı, öveçler, adeleler, pasifikteki, dejenerasyonun, arzuluyorsunuz.
> **Example:** *20 random unknown words from Word List*

These examples show that some of the compound noun forms such as *"derebeyi, boşboğazlık, canbazın, galatasaraylıyım, konyaspor"*; and old or borrowed Turkish words such as *"müstacel, semih, öveçler"*; and words with foreign origins such as *"dominantlığı, pasifikteki, dejenerasyonun"* cannot be processed with the current infrastructure of implementation of the TRmorph analyzer. The next sections and chapters examine the morphological analysis with the unknown words guesser, and how that affects the data evaluation and the analysis process in general.

Morpho Challenge Data Processing with TRmorph-Extended
After processing the Morpho Challenge data sets with the TRmorph-Baseline analyzer (without using the unknown word guesser utility); we obtained the coverage estimates given in table 4.24 above. Next, in order to re-examine the coverage, we processed these same data sets with the TRmorph analyzer run together with the guesser predicting unknown words. At this step, the TRmorph analyzer resulted in 100% coverage for all data sets by guessing morphological analyses for the previously-unknown tokens. Table 4.26 below shows the average number of analyses per token:

Data Sets	All Tokens	Avg. # of Analyses	Token Coverage
Word List	617294	475.8	100%
Training Set	1000	659.5	100%
Development Set	763	616.8	100%
Combined Sets	1760	641.4	100%

Table 4.26: Average number of analyses per token on Morpho Challange data processed with TRmorph-Extended

However, as we have noted before—especially regarding the Morpho Challenge data *Word List* file—due to data corruption during character conversion process and the main method of guessing full morphological paradigms of word forms in TRmorph; our findings show that even if certain word forms are nonsense, or legitimately incorrect *(and thus, should be returned as unknown by analyzers in ideal conditions)*, or misspelled, the TRmorph guesser returned spurious analyses for those *previously-unknown* tokens, resulting in 100% coverage in overall. Therefore, the results of 100% coverage does not mean 100% accurate coverage because extensive coverage is not necessarily equal to accuracy. For this reason, in the next section, we discuss the accuracy of morphological tags and evaluation metrics used in Morpho Challenge Shared Task both for tokens analyzed with the TRmorph-Baseline analyzer, and the analyzer from the TRmorph-Extended version, which utilized the guesser for unknown tokens.

4.4.1 Morpho Challenge Shared Task Data Evaluation

TRmorph Analyses and Morfessor Reference Methods

In Morpho Challenge shared task evaluations[1], from the Morfessor Reference methods, *Morfessor Baseline, Morfessor Categories-MAP* (Creutz and Lagus, 2007), and *letters*[2] methods were used. The Morpho Challenge evaluations were done by sampling a set of random word pairs created from the set of words with proposed morphemes, and comparing this word pair set morphemes against the common morphemes from the gold standard development/test set. In these Morpho Challenge Shared Task evaluations, the segmentation with the highest *F-Measure* was considered as the best result. Therefore, the F-measure was considered as the main evaluation measure, and it was calculated as *the harmonic mean of precision and recall*:

$$F = 2 \times Precision \times Recall / Precision + Recall \quad (4.1)$$

Besides the F-Measure[3] value, they evaluated *Precision*, and *Recall*. For precision, they generated random word pairs from the predicted morpheme analyses set. And then, compared this set with the gold standard analyses of word pairs in the original Morpho Challenge data. The ratio of common morphemes found in the predicted analyses and expected morphemes were normalized according to the number of words to be evaluated. This means that for each word pair that shares a morpheme in common with the gold standard analysis, one point was given.

Recall was normalized among all proposed alternative analyses of each word and the ones in the gold standard analyses. If words had more than one proposed alternative analysis, and the word pairs had more than one morpheme in common with the gold standard morphemes, then they assigned an equal weight for each alternative analysis, and word pair analyses[4]. If the words happen to have a lot more proposed alternative analyses, the best # match between the analyses is returned.

[1]For the details of all Morpho Challenge shared tasks, and evaluation & results, see: `http://research.ics.aalto.fi/events/morphochallenge/`

[2]Method for segmentation that results in the highest recall possible for segmentation of words to the letters they are made of

[3]Here, we give the standard formula of F−Measure, as given in (Manning and Schütze, 1999, p. 269), which slightly differs from the Morpho Challenge citation; however this formula matches the numbers obtained from their own evaluation results.

[4]If a word has 3 proposed analyses, the first analysis has 4 morphemes; and the first word pair in that analysis has 2 morphemes in common with the gold standard, then each of the 2 common morphemes amounts to $1/3 * 1/4 * 1/2 = 1/24$ of the 1 point given for each word.

By taking these criteria into account, in the 2010 Morpho Challenge shared task, using the letters *(segmentation)* and Morfessor reference methods evaluation metrics, the best scores obtained for precision, recall, and F-Measure for the Turkish data sets come from the following method and learning types[1,2]:

Morfessor Reference Methods

AUTHOR	METHOD	PRECISION	RECALL	F-MEASURE
Nicolas	MorphAcq (U)	79.02%	19.78%	31.64%
Golenia	MAGIP (S)	32.00%	65.80%	43.06%
Kohonen	Morfessor S+W+L (S)	71.69%	59.97%	65.31%
-	Morfessor CatMAP (U)	79.38%	31.88%	45.49%
-	Morfessor Baseline (U)	89.68%	17.78%	29.67%
-	Letters (-)	8.66%	99.13%	15.93%

Table 4.27: Morpho Challenge shared task results with Morfessor evaluations

We followed the evaluation metrics given for the letters and Morfessor reference methods, and evaluated the same Morpho Challenge word pairs data set in the 'gold standard development word pairs'[3] file. Using their evaluation metrics and instructions, we generated the set of tokens containing a mix of 300 random word pairs analyzed by TRmorph. For our own evaluation, the results were then compared between the following sets:

i. the gold standard word pairs
–TRmorph Set: Analyses of word pairs obtained by using the TRmorph-Baseline

ii. the gold standard development set labels
–TRmorph Set: Analyses of gold standard development set labels obtained by using the TRmorph-Baseline

Using these four files, we obtained the total precision, recall and F-Measure scores for the word analyses found by TRmorph analyzer in the development set. According to these evaluations, we also obtained the precision and recall values for "affixes" and "non-affixes"—in which the algorithm considers morpheme tags starting with an initial + sign as an *affix*, and the others as a *non-affix*—where

[1]The learning types given in the parantheses: S: semi-supervised algorithm, U: fully unsupervised algorithm.

[2]Full list of results at: http://research.ics.aalto.fi/events/morphochallenge2010/comp1-results.shtml

[3]Original Morpho Challenge Data set "goldstd_develset.wordpairs.tur":http://research.ics.aalto.fi/events/morphochallenge2010/data/goldstd_develset.wordpairs.tur

we converted the format of TRmorph analysis annotations to show only the inflectional morphemes as the affix. The evaluation results we got are given in the following table :

Morfessor Evaluation

	Precision	**Recall**	**F-measure**
	42.30% (122/288)	92.66% (265/286)	58.09%
Non-affixes:	69.34% (18/27)	93.73% (93/99)	79.71%
Affixes:	39.55% (103/261)	92.10% (173/187)	55.34%

Table 4.28: TRmorph-Baseline results on Morpho Challenge Data Sets

TRmorph-Extended Analyses and Morfessor Reference Methods

Here we give the results of the Morfessor evaluations based on the analyses of development-word-pairs set processed with the unknown tokens guesser utility in TRmorph-Extended version:

Morfessor Evaluation

	Precision	**Recall**	**F-measure**
	15.20% (46/300)	100.00% (286/286)	26.39%
Non-affixes:	64.84% (2/3)	100.00% (99/99)	78.67%
Affixes:	14.67% (44/297)	100.00% (187/187)	25.58%

Table 4.29: TRmorph-Extended results with Morpho Challenge Data Sets

From this evaluation comparison, our findings show that use of the guesser for unknown tokens significantly damage the precision and F-Score–while it raises the recall value to a higher number, which was already relatively good before the use of guesser for unknown tokens. The number of multiple analyses generated by the guesser for unknown tokens seems to impact the final results a lot more negatively that benefits of using the guesser does not really outweigh the use of TRmorph analyzer by itself.

4.5 Results of TRmorph Evaluation

As described in the Morpho Challenge results by Kurimo et al. (2010a), regarding all morphologically rich languages, our various coverage assessment and evaluation using the Morfessor Reference methods show that high number of inflected word forms *(due to productive nature of Turkish morphology)* poses a challenge

for morphological processing. Given the inflected forms of tokens, the use of a rule-based morphological analyzer, *(such as the TRmorph analyzer as the focus of this thesis)*, is indeed effective in providing the root-forms of words. However, this efficiency goes only as far as the the number of words that exist in the lexicon. As we have seen with the data evaluated, TRmorph does not cover the whole language; and most of the tokens that are unknown come from words that are rarely-used, or have foreign origins as it is the case with unknown proper nouns, or they may have unconventional spelling. The attempt at resolving unknown tokens via a guesser utility also seemed to damage the overall analysis results.

Considering these findings, there are several areas that need to be improved and resolved. For this, in the next chapter, we provide our proposed method for extending the fixed lexicons in TRmorph-Baseline so that the number of tokens returned unknown due to OOV tokens can be reduced. For this, we try to handle the processing of proper names and Named Entities for the nominal tokens group. Secondly, we provide an extension of lexicon for the processing of multi-word expression verbs. For processing of new tokens, we provide our proposed method for tokenization of new lexical items—which is added onto the TRmorph package. At last, we introduce a revised version of a finite-state guesser that will work only for the analysis of nouns. In the next chapter, we discuss the individual components of our proposed method, and provide a summary of architecture at the end.

Chapter 5

Methods for Expanding & Improving the Lexicon

In this chapter, we explore methods to expand and improve the existing TRmorph-Baseline lexicon by using linguistically-motivated, heuristic information extraction techniques and finite-state approaches to make a more flexible and comprehensive lexicon. The main methods we have employed in the construction of new lexicons used in our version of the TRmorph analyzer—*TRmorph+*[1], and the types of data we extracted for the collection of new lexical items are discussed with more detail in the following sections.

5.1 Method 1: Extraction of New Lexical Tokens for the Expansion of Lexicons

5.1.1 Abbreviations

The Abbreviation and Acronym lexicon in TRmorph-Baseline originally consisted of 83 lexical items. For the construction of this new lexicon, we extracted a list of abbreviations from the Turkish Wiktionary[2], and Turkish Wikipedia Person categories. These consisted of 118 N_abbreviations, composed of abbreviations of general common nouns, and 743 NP_abbreviations, composed of the abbreviations of organizations and foundations that stand for a proper name in the lexicon.

With a total of 861 unique additions of abbreviations, the current TRmorph+ Abbreviations lexicon consists of 944 abbreviations.

[1] We will refer to the version of TRmorph that we use in our own experiments as 'TRmorph+' for the rest of this thesis.

[2] https://tr.wiktionary.org/wiki/Kategori:Kısaltma_(Türkçe)

Lexicon Name	**TRmorph-Baseline**	**TRmorph+**
`Abbreviation`	`83`	`944`

Table 5.1: New abbreviations lexicon in TRmorph+

5.1.2 Parsing of Numbers & Digits in Dates

As given in the table 4.8 for *Unknown Tokens from W2C-Web Text* in the previous chapter, our evaluation and assessments of the TRmorph-Baseline analyzer showed that most of the formats of dates, such as `08/06/1933, 08/06/1936, 08/06/1938` were not analyzed in the TRmorph-Baseline analyzer. While it is still difficult, and indeed it is outside the scope of a general morphological analyzer to be able to process noisy content web-text; in our version of TRmorph+, we have proposed a method to parse several date formats.

Provided that the date format given in the input could adhere to the writing and orthography specifications given by the Turkish Language Institution[1], our method for parsing dates via the construction of a finite-state transducer reads the text input, accepts and tags the date format if the format is accepted by the transducer.

Processing Date Formats via the Construction of a FST Parser
For the new utility of TRmorph+, in the construction of the date parser FST, we used the examples provided in the Foma library for English date rules. Following the description of dates, as specified on TDK[2], the new dateparser transducer reads date formats in the input text, and adds '$< DATE >$' tags if the date format is accepted and recognized by the parser automaton. According to this parser, following formats of dates are recognized and accepted:

i. Month and days of the week for specific *dates*, that start with an upper case initial letter, as in '29 Mayıs 1453 (Salı)' *(May 29, 1453, (Tuesday))*, or just the combination of day of the month and name of the month as in '29 Ekim' *(October 29)*

ii. Dates that are written only with digital numbers using one of the two symbols as in '23.04.1923' or '29/10/1920'

[1] As mentioned before, parsing of unconventional text or text with noisy content is essentially impossible without using other exhaustive methods, which are not within the scope of this thesis.

[2] `http://www.tdk.gov.tr/`'Büyük Harflerin Kullanıldığı Yerler'

The binary of the dateparser.fst can be generated from the `makefile` of the TRmorph+[1] package. In order to use this utility, the following `flookup` command is used:

```
$echo "24 Aralık 1999" | flookup -i dateparser.fst
```

This command starts the flookup method of Foma in the inverse mode—which generates a tagged version of the date string. If the string is accepted by the parser, it returns, for example:

```
<DATE>24 Aralık 1999</DATE>    (‘24 December 1999’)
```

Alternatively, the date parser can be used via the Foma command by entering:

```
$foma
$regex @"dateparser.fst";
$down
>24.12.2000                  //entering date_string
```

5.1.3 Multi-word Expressions

Multi-word expressions are word formations that are composed of two or more words. The composed meaning of these lexical units are usually different from the lexical meaning of its individual words. In the previous sections, as we examined different Named Entity types—which are considered a unique type of Multi-word Expression, other lexical categories such as verbs, adjectives and adverbs and other part-of-speech types can form multi-word units that result in a different lexical meaning from their individual tokens. Therefore, the role of multi-word expression tokens in parsing is significant. The importance of these units is especially more visible in other NLP fields such as Machine Translation, as the composed meaning of multi-word expressions can be entirely different from the translation of its individual tokens (Eryiğit et al., 2011).

When these multi-word tokens are correctly tokenized to preserve their own unique lexical meaning, they also help reducing the spurious ambiguity since their morphological interpretation and analysis become more unique than its individual tokens. In their work, Oflazer et al. (2004) state that multi-word expressions in Turkish make heavy use of auxiliary/support verbs with lexicalized direct or oblique objects that have various morphological constraints. They describe the formation of multi-word expressions in four different categories:

[1]By prompting the command: $ make dateparser . Date parser transducer works only in generation mode. For more details, please see the `dateparser.foma` script in the TRmorph+ Directory.

i. *Lexicalized multi-word expression*: All individual tokens of the multi-word expression are fixed. For these types of multi-word expressions, final syntactic function and semantics are not predictable in advance from the structure and the morphological properties of its individual tokens. Some examples under this type are the following:

   a. hiç olmazsa *(at least)* → hiç*(never)*+Adverb, -ol*(be)*+Verb
   → hiç_olmazsa+Adverb

   b. ipe sapa gelmez *(worthless[1])* → ip*(rope)*+Noun, sap*(handle)*+Noun+Dat, gel*(come)*+Verb
   → ipe_sapa_gelmez+Adj

ii. *Semi-lexicalized multi-word expression*: Some individual tokens of the multi-word expression are fixed, and some can vary via inflectional and derivational morphology processes. Usually, the *(lexical)* semantics of the collocation is non-compositional. This group of multi-word expressions come from compound and auxiliary/support verb formations with two or more lexical items that are lexically adjacent. The head of the multi-word expression *(the last word in the sequence)* is a verb, or it is *(any)* word-form that is derived from a verb. Even if the tokens preceding the lexical head may be inflected, only the morpho-syntactic features of the head token give their features to the whole multi-word unit. Thus, only the head token's morpho-syntactic features go through word-formation via morphological derivation and inflection processes. For example:

   - kafayı ye– *(literally to eat the head – to get mentally deranged)* inflected by keeping the first token of the multi-word unit *'kafayı'* as fixed, and the last token—which is a verb—as inflected or derived in many ways, as in the following examples:

   a. kafayı ye-dim *(I got mentally deranged)*

   b. kafayı yi-yeceklerdi *(they were about to get mentally deranged)*

   c. kafayı yi-yenler *(those who got mentally deranged)*

   d. kafayı ye-digi *(the fact that (s/he) got mentally deranged)*

iii. *Non-lexicalized multi-word expression*: The multi-word expression is made by a morpho-syntactic pattern of duplicated and/or contrasting components, as in reduplications or semi-duplications. These are completely non-lexicalized, and only change the semantics of the phrase in use. Following examples can be considered for this group of non-lexicalized multi-word expressions:

[1]Literally, (he) does not come to rope and handle

–mavi mavi *(literally 'blue blue', used in order to enhance the blue color aspect of an object)*
–gelir gelmez *(literally '(s/he) comes, (s/he) does not come', expressing "as soon as s/he comes". This is an example of contrasting verbal duplication.)*

iv. *Multi-word Named Entities*: These are multi-word expressions for proper names for persons, organizations, places, etc. This group of multi-word expressions are also considered as semi-lexicalized multi-word expressions since they behave the same way, as we described with the previous verbal example. These Named Entities occur in the text as all of its tokens except for the last—which is the head—being fixed, and the last token may go through several morphological processes reflecting nominal word formation. The example below shows a proper noun Named Entity, whose very last token carries the nominal case mark, and it represents a case marking on the whole named-entity:

- Türkiye Büyük Millet Meclisi'nde *(In the Turkish Grand National Assembly)* → Türkiye_Büyük_Millet_Meclisi+Noun+Prop+A3sg+Pnon+Loc, (Oflazer et al., 2004)

In TRmorph+, although we do not morphologically mark the differences between these four groups of multi-word expressions, in our data/token collection for the expansion of lexicons, we closely followed these descriptions. As a result, we mostly collected lexicalized and semi-lexicalized multi-word expression forms, besides the various Named Entity types. Non-lexicalized multi-word expression types were already being handled in TRmorph-Baseline. While we did not expand the group of non-lexicalized multi-word expressions very much; nonetheless, we still handle the proper tokenization of all multi-word expression forms. In the sections below, we describe the lexicon changes undertaken for TRmorph+ regarding the multi-word expressions and Named Entity types.

i. Verbal Multi-word Expressions

For the processing of the multi-word expression verbs, we extracted 10038 unique MWE-verbs from the Turkish Wiktionary from different categories, such as phrasal verbs, regular multi-word verbs, and idiomatic verbs *(the ones that have mostly non-compositional meaning)*. For the lexicon preparation and pre-processing, we removed all the verbal endings from these verbs in order to leave them in their root form. After removing all the duplicates, we obtained a unique list of 9985 MWE-verbs collected only from the Turkish Wiktionary.

Besides the MWE verbs taken from Wiktionary, we extracted 50 MWE verbs from the ITU-Validation set[1] (Eryiğit, 2007), (Eryiğit, 2012)–*-which is available in the METU-Treebank, and was prepared as a test set for the CoNLL-XI Shared Task*—containing MWE annotations. After the ITU-Validation sets, we extracted the MWE tokens from the METU-Treebank CoNLL Sets. These were the tokens that were returned as *unknown* by the TRmorph-Baseline earlier, as we had shown in table 4.19:*Distribution of MWE Tokens on METU TreeBank*. From those unknown MWE tokens in the CoNLL set, we extracted 858 MWE verbs. After filtering out duplicates, and removing the verbal inflections and underscore connectors, we obtained the following unique counts from these data sets for the initial preparations:

1. ITU-validation Set consisting of 42 MWE verbs
2. CoNLL Data Sets, consisting of 485 MWE verbs
3. Combination of Wiktionary, ITU, CoNLL tokens: 10500 MWE tokens
   - Unique count of all of the MWEs: 10125/10500

For further pre-processing, we removed all of the punctuations such as quotation marks, exclamations marks, and commas from the MWEs, and also added derivational boundary markers for verbs that have internal stem changes due to vowel harmony and consonant alternations. After we filtered out the original 77 MWE verb tokens that already exist in the TRmorph-Baseline verb lexicon from the new MWE-verb lexicon for TRmorph+, we obtained 10062 unique MWE additions. For the MWE-verb lexicon, for correct verb-formation, we divided the types of verbs into five different categories according to their lexical and derivational types.

1. Verbs ending with *"-et"* auxiliary as its head → *"Aorist"* verb
   –e.g. 'yardım et-' *(make help)*
2. Verbs ending with *"-kal, -ver, -al, -ol"* auxiliary as their heads → *"Regular Causative"* verbs (ending with -dir)
   –e.g. 'aç kal-' *(stay hungry)*, 'akıl ver-' *(give idea/opinion)*, 'ifadesini al-' *(take testimony)*, 'abone ol' *(become a subscriber)*
3. Verbs ending with *"-sür, -et, -gör"* auxiliary as their head→ *"Aorist Causative" (irregular ending causatives)*
   –e.g. 'yokuşa sür' *(ride (against) uphill, make difficult)*, 'alay et-' *(make fun (of))*, 'hoş gör' *(be nice, take it lightly)*

[1] `http://web.itu.edu.tr/gulsenc/treebanks.html`

4. All verbs ending with *"-ol"* auxiliary as its head → *"Irregular Causative" (verbs that cannot be causative–as in the passive forms of the verbs)*
   –e.g. 'şart ol-' *(have/be a condition)*

5. Verbs ending with *"-gör"* auxiliary as its head → *"Reflexive-Reciprocal Irregular Causative" (verbs that take the causativity from the reflexive or reciprocal forms of the specified verb)*
   –e.g. 'kırmızı kart gör (göster)' *(to show red card)*

6. All the remaining verbs that do not fit any of these categories were tagged as *"Verb"* for regular verb formation.

After tagging these verbs according to their appropriate verb-form categories, we obtained the following unique MWE verb counts:

Type of Verbs	**# of Tokens**
Regular verbs	6525
Aorist Verbs	1455
Regular Causatives	1827
Aorist Causatives	1583
Irregular Causatives	956
Reflx-Recp Irr.Causatives	96
TRmorph-Baseline MWE verbs	77
TRmorph+ MWE Verbs	12442

Table 5.2: New MWE-Verbs lexicon in TRmorph+

ii. Nominal Multi-word Expressions

For the expansion of the multi-word expression nouns lexicon, we extracted the list of regular common nouns template from the Turkish Wiktionary[1]. From this list, we filtered out single token nouns from the multi-word expression nouns, and obtained a new list of of 13616 new multi-word expression noun tokens. Together with the 71 multi-word expression nouns already existing in the TRmorph-Baseline lexicon, currently TRmorph+ has 13663 new multi-word expression nouns in the new lexicon. Table 5.3 below shows the number of tokens that have been added to the TRmorph+ lexicon.

[1] `https://tr.wiktionary.org/wiki/Kategori:Ad`

Lexicon Name	**TRmorph-Baseline**	**TRmorph+**
MWE_noun	71	13663

Table 5.3: New multi-word nouns lexicon in TRmorph+

These multi-word expression nouns are almost strictly made of lexicalized multi-word expressions groups, where all individual tokens are fixed, and morphological properties of individual tokens do not change according to context, and they are not predictable in advance. Some examples of this category include the following multi-word expressions:
> maskeli balo → *'(with)*mask' + 'ball' → masquerade
> İngiliz anahtarı[1] → 'English' + 'key' → monkey-wrench
> tüp bebek → 'tube' + 'baby' → *baby born via ivf*
> tükenmez kalem →*'(never running out)* infinite' + 'pencil' → pen

iii. Adjectival Multi-Word Expressions

For the extension and improvement of the Adjective and Adverb lexicons, we examined the adjective and adverb tokens currently existing in the TRmorph-Baseline lexicons. In the adjectives lexicon, TRmorph-Baseline had 53 MWE tokens. However, as it is the case for other lexicons as well, these were only processed correctly as adjectives when they were tokenized as a single unit as in the following token:

```
>$akşamlı sabahlı ('morning and evening, always')
```

This yielded the following analysis:

```
apply up> akşamlı sabahlı
akşamlı sabahlı<adj>
akşamlı sabahlı<adj><JN_0><n>
akşamlı sabahlı<adj><JN_0><n><3p>
akşamlı sabahlı<adj><JN_0><n><3s>
```

However, when these words were tokenized as single units, as in,

```
>akşamlı
>sabahlı
```

[1] One specific group of Turkish mwe-nouns are composed via suffixation of 'noun-compound' making suffix /-ı, -i, -u, -ü, -sı, -si, -su, -sü/ inflections–which are added onto the head of the multi-word expression as in 'İngiliz anahtar-ı'. In order to prevent lexical ambiguity, due to the high syncretism of this suffix with other suffixes, such as possessive pronoun and accusative case suffix, these groups of mwe-nouns have been directly marked as 'MWENComp' on the lexicon.

they incorrectly yielded different analyses. If the words are analyzed as single-word tokens, because the TRmorph-Baseline analyzer does not tokenize MWE units, the analysis often returns different morphological features from the ones that actually compose the meaning of MWE units[1]. In our current version of TRmorph+, we included 53 of the original TRmorph-Baseline MWE adjectives in the construction of our FSA dictionary so that the sequence of such tokens in a context can be properly tokenized and spurious ambiguity can be eliminated. We discuss the method of tokenization and the description in the following sections of this chapter.

iv. Adverbial Multi-Word Expressions

As for the processing of adverbs, we examined the current adverb tokens existing in the TRmorph-Baseline lexicon, and extended this lexicon with other verbs extracted from the CoNLL data sets. The current TRmorph-Baseline adverbs lexicon had 113 MWE tokens. To increase the number of adverb MWEs previously not covered, we extracted 135 MWE tokens from the CoNLL Shared Tasks Data Sets–METU Treebank, which we had used in our coverage assessments earlier.

For the pre-processing of these tokens from the CoNLL data sets, we removed underscores, and inflectional endings to prepare the root-form list to be put on the new adverbs lexicon. After removing duplicate tokens that already existed in the original TRmorph-Baseline adverb list, we obtained 97 unique MWE adverbs extracted from the CoNLL set. Table 5.4 below shows the number of total and MWE tokens in the current TRmorph+:

# of Tokens	**TRmorph-Baseline Adverbs**	**TRmorph+ Adverbs**
`# of MWE tokens`	`113`	`210`
`# of Total tokens`	`562`	`659`

Table 5.4: New adverbs lexicon in TRmorph+

Originally the Trmorph-Baseline had 562 adverbs all together, consisting of single and multi-word token adverbs. After our new addition of 97 MWE adverb tokens, we reached a total of 659 adverbs. 210 of these MWE adverb tokens were also added to the separate FSA tokenization so that they can be correctly analyzed when they occur in sequences.

[1]In order to save space, we describe and discuss these relevant examples in the following section.

5.1.4 Proper Noun and Named Entity Lexicons

For the extension of the proper noun lexicon in the TRmorph-Baseline, we extracted more proper nouns from several resources and divided these proper nouns into several different categories. The original baseline proper noun lexicon consisted of 10049 tokens, including several multi-word proper nouns. In our version of the proper noun lexicon, we categorized the new proper nouns extracted from new resources according to the following groups:

i. *Single-token proper nouns*: These are tagged as 'NP', and have been added to the baseline lexicon, originally consisting of 10049 tokens.

ii. *Multi-word token proper nouns*: These are tagged as 'NeNp' in the lexicon[1] and additionally exist in the Named Entity FSA recognizer dictionary used for the tokenization of named entities.

iii. *Single-token place names*: These are tagged as 'NeLoc'[2] in the lexicon.

iv. *Multi-word place names:* These are tagged as 'NeLoc' as well, and exist within the same 'NeLoc' lexicon as in (iii), in addition to Named Entity FSA recognizer dictionary used for tokenization.

v. *Multi-word token ORGanization names*: These are tagged as 'NeOrg'[3], and exist in the Named Entity FSA recognizer dictionary, as well.

i. Single-Token Proper Nouns

For the extension of the "NP" lexicon in (i), we extracted 19371 tokens of Turkish person names *(male names, female names, last names)*, and 17232 tokens of English proper names from the Turkish Wiktionary. Additionally, we extracted 753 person names from Wikipedia person categories, and added an initial total of around 37K new proper noun tokens.

After filtering out tokens that already exist in the TRmorph-Baseline proper noun lexicon, we obtained 36665 unique single-token proper names. However, we refined this lexicon some more by filtering out acronyms and proper noun place and geographical location names. In the end, we reached a unique number of 23264 new tokens for the proper nouns lexicon, which were added as "NP"s to the new TRmorph+ `proper_noun` lexicon, on top of the original 10049 proper noun tokens existing in the TRmorph-Baseline. As a result, the current version

[1] "NeNp" tokens have `'<ne_np>'` tags in the analyses.
[2] 'NeLoc' tokens have `'<ne_loc>'` in the analyses
[3] 'NeOrg' tokens have `'<ne_org>'` tags in the analyses

of the proper_noun lexicon in TRmorph+ consists of 33311 unique proper name tokens.

Lexicon Name	**TRmorph-Baseline**	**TRmorph+**
`proper_noun`	`10049`	`33311`

Table 5.5: New proper noun lexicon in TRmorph+

ii. Multi-Word Proper Noun Named Entities

For the new lexicon in (ii)—which is an extension of the "NP" tokens—we extracted 32597 tokens of multi-word person names from various Wikipedia categories, and 640 multi-word tokens of Turkish person names, and around 3900 multi-word tokens of English person names from the Turkish Wiktionary.

After combining all of these files and filtering out duplicate tokens, we obtained 35927 unique multi-word tokens for the NE_proper_noun lexicon.

Lexicon Name	**TRmorph-Baseline**	**TRmorph+**
`NE_proper_noun`	`--`	`35927`

Table 5.6: New Named-Entity proper noun lexicon in TRmorph+

iii. Single-token Place Names

As a new extension of the proper noun lexicon, we have added another category of proper nouns for place and geographic location names. We further split this category into single-token *(single word)* and multi-word token location names. For the construction of single-token location names, we extracted location and place names from the following resources:

i. 4197 single-token Turkish location names from a web-blog database[1]

ii. 52 single-token location names from Wikipedia place and location categories

iii. 47126 single-token location names from the Geospatial Intelligence Agency for places in Turkey[2]

[1] `http://blog.fatihaytekin.com/javascript-il-ilce-semt-mahalle-listeleme-db-baglantili-asp`
[2] `http://earth-info.nga.mil/gns/html/namefiles.htm`

After post-processing this file to filter out duplicate tokens and tokens that already exist in different categories or other lexicons and the original TRmorph-Baseline proper_noun lexicon, we obtained 45963 new and unique location names, which are tagged as "NE_Loc".

iv. Multi-word Place Named Entities

For multi-word location and place names, we extracted multi-word tokens from the following resources:

i. 25537 tokens from the directory of Turkish location names from the same web blog used for the single-token location names

ii. 1363 tokens from Wikipedia location and place names

iii. 20267 tokens from the Geospatial Intelligence Agency directory, used for the single-token location names

iv. 550 tokens from English and Turkish Wiktionary for Turkish country names

From a total of 47717 tokens, after removing the duplicates we obtained 47032 unique tokens made of multi-word location and place names. Combined with the single-token location names, the current TRmorph+ contains the following location named entity lexicons:

Lexicon Name	**TRmorph-Baseline**	**TRmorph+**
`NE_Loc: Single-Token`	`--`	`45963`
`NE_Loc: Multi-Token`	`--`	`47032`

Table 5.7: New Named Entity-Location lexicons in TRmorph+

v. Single & Multi-word Organization Named Entities

For the organization names, we extracted a list of categories from Wikipedia for the Turkish organizations. After removing the abbreviated names from this list, we obtained a total of 883 Organization names. Additionally, we collected 1290 other organization names from various web resources such as official websites of the government organization listings, and ministries, and related governmental foundations. From the combined list of organization names, 41 of them consist of single-token organization names, and the rest makes up for the list of multi-word tokens organization names.

Lexicon Name	**TRmorph-Baseline**	**TRmorph+**
NE_Org	--	2132

Table 5.8: New Named Entity-Organization lexicon in TRmorph+

5.2 Method 2: Tokenization of Multi-word Expressions

In this section, we describe our method—which consists of several steps—for proper tokenization and morphological analysis of multi-word expressions in Turkish. In the next section, we introduce the dictionary that was prepared as a preprocessing step, in preparation for the structures needed in our main method of FSA-based recognition and tokenization of multi-word expressions. After that, we discuss the various individual steps that are involved in our method for tokenization of multi-word expressions.

5.2.1 Preprocessing Step: Preparation of the FSA Dictionary

As we have earlier introduced the new lexicon tokens in Method 1 for *the Expansion of TRmorph Lexicons*; as the preprocessing step of our Method 2 for the construction of the FSA-based tokenizer, we compiled a dictionary of multi-word expression tokens, consisting of tokens collected for the general expansion of TRmorph lexicons, and other multi-word expression tokens that already existed in TRmorph-Baseline previously—which were not tokenized in the TRmorph-Baseline. For the construction of this dictionary, we collected multi-word expression tokens from the following lexicons:

Type of MWE-Tokens	# of Tokens
MWE-Verb:	10021
MWE-Noun:	13663
MWE-Adj:	53
MWE-Adv:	210
NE-NP:	35927
NE-LOC:	47032
NE-ORG:	2091
MWE-Pronoun:	18
MWE-Conj:	26
MWE-Det:	9
MWE-Interj:	58
TOTAL:	109108

Table 5.9: MWE tokens in the FSA dictionary

Multi-word expression tokens in this dictionary exist in the same form they exist in their corresponding TRmorph lexicons. That is, multi-word expressions are only uninflected on the head-final token *(last token)* of the MWE-token sequence because the TRmorph lexicons can only hold the root-forms of the lexical items. Because we are only mainly concerned about the processing of lexicalized and semi-lexicalized multi-word expressions, as discussed earlier on 61; morphological features that are fixed on the *non-head, intermediate* tokens within the multi-word expression have not been removed. However, another important difference between the lexicon tokens and the dictionary tokens is that dictionary tokens exist in their plainest *(uninflected)* surface form; while the lexicon tokens carry certain morphophonological features by marking them on the lexicon directly. For example, certain lexical tokens that are irregular, and thus go through irregular vowel harmony or various voicing/devoicing phenomena *(or a combination of both)*, had to be marked on the root-boundaries directly on the lexicon. For the dictionary tokens, since we are only interested in the surface form of the tokens *(which are used only as input for morphological analysis or generation)*, morphophonological root-boundary markers could not be included in the dictionary.

5.2.2 Tokenization Step: Construction of the Finite-State Recognizer

> It is important to realize that tokenization is a non-trivial task. In real life, tokenization is seldom as simple as dividing a text at the spaces and punctuation marks. For one thing, MULTI-WORD TOKENS like the classics to and fro and far and away contain spaces and yet should be kept together as single tokens; the meaning of these tokens is not a syntactic function of the individual orthographical words, which sometimes, like the archaic word fro, cannot even appear outside the fixed construction. (Beesley and Karttunen, 2003, p. 529)

For the task of tokenization of English multi-word tokens, sample solutions have been proposed—which use finite-state transducers that mark maximally long multi-word expressions, only if the expression boundaries correspond to normal word boundaries (Beesley and Karttunen, 2003, p. 537). According to this approach, given a token as a *(fixed-length)* multi-word expressions such as 'New York', it allows other forms of multi-word expressions such as 'New York' and 'New Yorker', both to be recognized and accepted by the transducer, so that not just 'New York' is accepted.

However, in this approach, even if the morpheme boundary `-er` in 'New York**-er**' is recognized as the longest matching token between 'New York' and 'New Yorker', proper tokenization of both multi-word expressions works **only if** both of the multi-word expression tokens exist in the list of MWE tokens in the transducer. In other words, both 'New York' and 'New Yorker' must exist in the list of multi-word expressions, so that the maximally longest match operation can recognize and accept, and properly tokenize both of the multi-word expressions *(namely, given only the multi-word expression token 'New York', the transducer cannot deduce all the derived forms of 'New York', for 'New Yorker', 'New Yorkers', 'New Yorker's', 'New Yorkers' ', and so on...).*

This issue of morphological recognition of derived forms of multi-word expressions presents the same problem we face in the recognition of Turkish multi-word expressions. Indeed, considering, the list of potential inflected forms of multi-word expressions in Turkish is a lot longer *(indeed, considered as infinite)* and complex, recognition of multi-word expressions in Turkish is not directly straightforward. Regarding the challenge in proper tokenization of multi-word tokens, in their work Oflazer et al. (2004) state that for the morphological processing of multi-word expressions, at the most basic level, it is important to recognize a sequence of input tokens whose morphological analyses match the patterns that

exist in the actual multi-word dictionary *(list)*; and then collapse these multi-word tokens into a single multi-word expression representation.

The problematic issue arises from the fact that the multi-word expressions in Turkish exist in different morphological word forms, while the FSA dictionary *(used in the construction of the FSA, responsible for exact pattern matching of the input and the dictionary tokens)* can only keep a finite number of multi-word expressions[1]. Therefore, performing the exact pattern match task—when given a dictionary with a finite set of tokens and another potentially infinite set of word forms—is a *non-trivial* task.

As a solution to overcome the exact pattern match problem[2], in our method, we break down the words into their stems before they can be matched so that both the FSA dictionary tokens, and the input tokens can be in their plainest / uninflected forms *(without their morpho-syntactic features)*. Stemming task is performed within the main FSA algorithm by calling the TRmorph utility `stem.fst` in `flookup` mode. Once the stemming step is done, stemmed words *(from the input)* are matched with the stemmed dictionary tokens by using the FSA recognition method. After a successful match, original input words are coalesced together into a single-unit expression and checked against the multi-word lexicon existing in the TRmorph analyzer which keeps only the final word root form in the sequence of multi-word expression tokens. Hence, our target method checks for three kinds of multi-word strings.

- **Input:** First, it reads the input *(given as single-tokens separated by* $\backslash n$*)*;
- **Stemmed-input:** Second, it performs the recognition of input using the FSA dictionary tokens *(after the stemming pre-processing step)*;
- **Tokenized-output:** Third, recognized tokens are collapsed and new tokenized output is produced—which is then used in the morphological analysis step, matching each tokenized output with the lexicon tokens in the main TRmorph lexicon.

In more detail, the main structure of the steps can be described as the following:

1. User input is read as:
   > izin
   > istediler *((They) 'wanted' 'permission')*

[1]Due to the impossibility of listing all word-formations that can potentially be infinite due the productive morphology in Turkish.

[2]Infinite number of potential word forms against the smaller number of uninflected word forms in the dictionary

2. FSA-Dictionary used in recognition of multi-word expressions has a MWE form, as:
   – izin iste-

   This specific FSA works by reading the user input and matching it with the dictionary tokens. If there is a successful match, it ACCEPTs the input, otherwise it will output FAIL. The construction of the FSA has two possible outgoing arcs for every state and every input token it reads. The first arc is the one going to an ACCEPT state *(when the MWE is complete–in a final state)*, and the second arc is for the *continuation, (pre-fix)* of the MWE to select the possible longest match. For example, given the following lexicon for some multi-words:

   – kaynama$_{t1}$ noktası$_{t2}$ *(boiling point)*
   – kaynama$_{t1}$ noktası$_{t2}$ yükselimi$_{t3}$ *(boiling point elevation)*

   If "*kaynama noktası*" is read by the FSA, it will output an 'ACCEPT', when it reads the token *noktası*$_{t2}$ after *kaynama*$_{t1}$. However, since '*kaynama noktası*' is a sub-string of '*kaynama noktası yükselimi*', the FSA will continue reading *yükselimi*$_{t3}$ and also output an 'ACCEPT', so that overlaps are resolved, and the longest match is always selected–which results in accepting both of the MWEs.

   Given this recipe, in our version the FSA's task is to match the sequence of user input *(e.g. 'izin', 'istediler')* against the sequence of dictionary tokens *('izin iste-')*. For this task, the FSA algorithm first performs the stemming task over the user input and the dictionary tokens[1]. If there is a match between the stemmed tokens of the input and the dictionary tokens, the algorithm collapses the user input tokens into a single-unit expression of multi-words. These steps are applied in the following order:

   i. STEM the user input tokens, as in:

Original Input	Stemmed Input
izin	izin
istediler	iste-

[1]Dictionary tokens are stemmed in advance during the training step

ii. MATCH & RECOGNIZE the sequence of stemmed tokens with the dictionary tokens as in:

Stemmed Input	Dictionary Tokens
izin$_{t1}$ iste–$_{t2}$	izin$_{t1}$ iste–$_{t2}$

If the matching of the sequence of tokens is successful, as it is in this example...

iii. COLLAPSE the original user input tokens into single-unit expression for further morphological processing, as in:

Original Input	Successful Match	Collapsed Input
izin istediler	izin iste-	izin istediler

3. TRmorph MWE Lexicon has the MWE token as in:
– izin iste- #;

Once the collapsed token obtained from the previous step is queried for morphological analysis, the TRmorph analyzer scans the list of MWE tokens in the MWE lexicon. If there is a match between the root-form of the collapsed input and the lexicon token[1], analysis succeeds, as in:

```
apply up> izin istediler
izin iste<v><t_past><3p>
```

The benefit of single-unit tokenization of multi-word expressions is that when such multi-words are collapsed into a single lexical unit, the morphological ambiguity is reduced—which helps greatly in the task of further syntactic parsing and other NLP methods that require unambiguous input. Following the MWE example we have used for the description of the tokenization steps above, here we can see the effects of how collapsed tokens reduce the number of potential morphological analyses:

[1]Because the lexicon tokens are kept in their root forms, and TRmorph itself finds the root-forms of input words so that it can match the input with the lexicon tokens.

```
apply up> izin     (permission)
izin<n>
izin<n><3p>
izin<n><3s>
*iz<n><p2s>
*iz<n><p2s><3p>
*iz<n><p2s><3s>
*iz<n><gen>
*iz<n><gen><3p>
*iz<n><gen><3s>
*iz<n><ncomp><p2s>
*iz<n><ncomp><p2s><3p>
*iz<n><ncomp><p2s><3s>

...

apply up> istediler     (want)
iste<v><t_past><3p>
```

As we can see from the individual analyses of these tokens; the token *izin* results in 12 different analyses, 9 of which *(marked with * above)* have root forms that are completely different from the original root form, which is caused by lexical ambiguity, resulting in many different morphological analyses. In the morphological analysis of multi-word expressions, since the (semi-)lexicalized tokens are analyzed based only on the morpho-syntactic features of the last token *(the head of the multi-word expression)*, spurious analyses are eliminated. Our method shows that proper tokenization and morphological analysis of multi-word expressions greatly help reducing the number of multiple analyses of tokens, caused by lexical ambiguity; therefore, decreasing the morphological ambiguity altogether, which aids in preparing the text for other higher level tasks, such as syntactic parsing and machine translation.

5.3 Method 3: Finite-State Guesser for Proper Nouns

This is a revised version of the guesser utility that is distributed with the extended version of the TRmorph-Baseline. In order to make this guesser more robust, and compliant with TRmorph+, we have decided to remove certain features, and add some other features to the distributed version of the unknown words guesser in TRmorph-Baseline. In our version of the guesser for TRmorph+, only single to-

ken noun and 'NP' *(proper noun)* word forms are considered for unknown-token guesser processing.

Our evaluations and observations with guessing other POS categories—which is the case with the guesser implementation in the TRmorph-Baseline version—have shown that non-nominal word forms are generally 'known' and analyzed by the analyzer—as long as they have proper spelling that adhere to the spelling and orthography guidelines. Our evaluations of TRmorph-Baseline also showed that the majority of unknown tokens come from the nominal categories. Among those, the most frequent unknown tokens come from proper nouns starting with upper-case initials letters, including Turkish and non-Turkish letters. In order to reduce the number of analyses generated by the regular analyzer and the guesser.fst, we have decided to guess tokens only for the 'noun' and 'NP' part-of-speech categories.

The rules that define nouns and proper nouns that are expected to be seen in text have been revised in order to allow both Turkish letters, and the non-Turkish letters used in Latin alphabets, and their upper-case and lower-case letter equivalents[1]. In order to use the guesser transducer[2] for the analysis of unknown tokens, the following commands can be used:

```
$foma
$regex @"guess.fst";
$up
$up> 'enter_token_to_guess'
```

Alternatively, the following `flookup` commands can be used in order to process a file with a list of tokens:

`$ flookup guess.fst < file_to_guess_tokens` *(invokes only the guess.fst)*
`$ flookup guess.fst trmorph.fst < file_to_guess_tokens`
(Second flookup command invokes both guess.fst and trmorph.fst concurrently. Tokens that are unknown to trmorph.fst are analyzed by guess.fst automatically.)

However, it should be noted that there is a compromise to be made in using the guesser utility because while the guesser can provide benefits for analyzing important unknown tokens; it can also hurt the overall results of an evaluation. While the guesser can analyze all the legitimate guessable noun and NP types correctly *(that is, if the tokens are correctly spelled and tokenized, adhering to*

[1]For more details, see the `guesser.lexc` in the main `TRmorph+` directory.

[2]The guesser transducer can be regenerated via the `makefile` by entering `'make guesser'`, which generates the `guess.fst` binary file.

the general orthography rules); the usage of the guesser utility also increases the number of multiple analyses overall because the transducer enumerates over all the potential word-forms that match the character boundaries as specified in the guesser lexicon. This leads to more tokens being analyzed, generating analyses for all the potential word-forms, than the actual word-forms that may be seen in real data, and the actual number of analyses that would normally be generated with the regular analyzer, which depends on the existence of tokens in the lexicon.

Therefore, it is important to determine in advance whether the guessing *(and analyses of)* all potential word forms is more important, by accepting the compromise in the increase of the number of analyses, or whether keeping the number of analyses as minimal as possible is more important even if it means the coverage may hurt. In order to demonstrate how this plays a role in evaluation, we give a short comparison of the usage of the guesser utility by using 1000 tokens, randomly extracted from the Web-to-Corpus, Wikipedia Data Set. Table 5.10, below shows the results of the analyses obtained by using just the regular analyzer `trmorph.fst`, and `guesser.fst` together with the `trmorph.fst` analyzer. These results are further compared with the TRmorph-Baseline *(in the trmorph.fst column)*, and TRmorph-Extended *(in the guess.fst column)* by using the same 1K wikipedia tokens.

	`trmorph.fst`	`guess.fst + trmorph.fst`
# of Tokens	1000	1000
# of Unknown Tokens	`445`	`207`
TRmorph-Baseline:	*484*	*202*
# of Analyzed Tokens	`555`	`793`
TRmorph-Baseline:	*516*	*798*
# of Avg. Analyses	`6.01`	`10.5`
TRmorph-Baseline:	*23.09*	*14.9*

Table 5.10: Comparison of analyses with `trmorph.fst` and `guess.fst + trmorph.fst`

The results obtained from the evaluation of this 1K text file taken from the Wikipedia text show that the new guesser that we introduce with TRmorph+ is able to analyze 207 of the total 445 tokens that were unknown to the trmorph.fst analyzer. Tokens that are 'known' to trmorph.fst have the same number of analyses in both cases *(i.e. guess.fst has no effect on tokens that are already known to trmorph.fst)*. However, the unknown tokens analyzed via guess.fst get so many

analyses that the number of average analyses over all tokens increases. The remaining tokens that are still unknown to guess.fst as well can be explained by the fact that the text we analyzed in general had noisy content, including a lot of unconventional spelling and misspelled tokens. Looking at the list of tokens that are still unknown in more detail, we see the following examples:

```
bulgularDoğrulanabilirlik +?
DosyaTiflis +?
El-Hakim'in +?
kav-ramı +?
Güzelyurt?ta +?
```

We can see that the unknown analyses stem from the way these tokens are spelled—because the guesser utility is not designed to analyze mixed-character tokens, or tokens that have a hyphen, or other miscellaneous internal punctuation marks.

The difference between the number of analyzed tokens *(793/798)* and remaining unknown tokens *(202/207)* between the TRmorph+ and TRmorph-Baseline can be explained by the fact that TRmorph+ is restricted to guessing only nouns and proper nouns, while TRmorph-Baseline guesses all open-class word-forms. Therefore, for a guesser that has far less restrictions, it is normal to see a higher number of tokens being guessed than a different guesser that has more restrictions. It is important to note that the difference between the number of tokens analyzed and the number of tokens that are guessed is rather small *(even if it is difficult to assume that the difference is insignificant)*, our evaluation shows that guessing more tokens does not outweigh the number of average analyses per token, because the ability to guess only five more tokens increases the average from `10.5` in TRmorph+ to `14.9` in TRmorph-Baseline. Therefore, in our version of TRmorph+, we keep the guesser utility optional by default.

5.4 General Outline and Workflow of Methods

In the previous sections, we described and discussed the methods we have added for TRmorph+. In this section, we give a general summary of what has changed, and describe the general workflow of TRmorph+.

Summary of New Lexicon Additions in TRmorph+

Table 5.11 below shows the list of lexicons and token numbers that have been added to the TRmorph+, and gives the corresponding token numbers of lexicons in TRmorph-Baseline:

	TRmorph+	TRmorph-Baseline	
Abbreviation	944	84	
Adverb	210	113	
MWE_noun	13663	71	
Proper Noun	33311	10049	*(single tokens)*
NE-Proper Noun	35927	–	*(multiple tokens)*
NE-Loc	45963	–	*(single tokens)*
NE-Loc2	47032	–	*(multiple tokens)*
NE-Org	2131	–	
MWE_verb	12442	77	
TOTAL	191623	–	

Table 5.11: New lexicons in TRmorph+

Word Formation Changes in TRmorph+

In our attempts to reduce morphological ambiguity, we have decided to modify two of the following morphological features, which were causing a high number of morphological analyses to be generated for each word form:

i. Noun vs. NComp & AdjComp: Previously, in TRmorph-Baseline, all the *noun* tokens that had the ambiguous morphological feature, noun-compound making suffix /-ı, -i, -u, -ü, -sı, -si, -su, -sü/ were also marked as 'NComp' by default. This was made in an effort to cover all the potential cases, where a noun could also be used as part of a noun-compound phrase.

As we have seen the examples of *lexicalized multi-word expression nouns* in the previous sections[1], these tokens actually can only be a part of a noun-compound within a "noun phrase", rather than when used as stand-alone 'noun' tokens. In other words, the noun-compound suffix cannot be used in a stand-alone token to give the meaning of a noun-compound. When the ambiguous morphological feature is observed on a single token, when it is not part of a noun-phrase, it functions either as a possessive-making or accusative-case making suffix *(depending on the context)*. Therefore, the addition of the 'Ncomp' feature for an already ambiguous suffix was causing unnecessary analyses.

Relying on these observations, we have removed the 'NComp' feature from the analysis of nouns, as well as the feature for 'AdjComp' because all the Adjectival

[1]Lexicalized multi-word expressions, on page 65

morphological features are a continuation class of the nominal morphological features. In the current version of TRmorph+, single token nouns or adjectives are no longer marked as 'NComp' (or 'AdjComp' respectively) by default–even when they have the noun-compound making suffix. Marking of actual 'noun-compounds' are now handled via the 'MWE_NComp' group, according to the suffix they carry. Two of the examples below illustrate the current changes in TRmorph+ for nouns and multi-word expression nouns:

i. When the ambiguous suffix is attached to a multi-word expression noun:

```
$ echo "İngiliz anahtarı" | flookup trmorph.fst
İngiliz anahtarı İngiliz anahtarı<mwe_ncomp><n><p3s> ('monkey-wrench')
İngiliz anahtarı İngiliz anahtarı<mwe_ncomp><n><p3s><3s>
İngiliz anahtarı İngiliz anahtarı<mwe_ncomp><n>
İngiliz anahtarı İngiliz anahtarı<mwe_ncomp><n><3s>
```

ii. When the ambiguous suffix is attached to a single-token noun:

```
$ echo "anahtarı" | flookup trmorph.fst
anahtarı anahtar<n><acc>('key')
anahtarı anahtar<n><acc><3s>
anahtarı anahtar<n><p3s>
anahtarı anahtar<n><p3s><3s>
```

ii. 3rd Person Agreement in Verbal Predicates:

Segmentation generation of the following verb, 'okur' *(reads)* which has 3rd person singular agreement *(with a null-morpheme, signifying the 3rd person agreement)*, shows to have also the plural morpheme -lar. This is an example of an overgeneration because TRmorph-Baseline's verbal predicate word-formation rules had a plural null-morpheme attachment as 3rd person plural agreement for every verb formation. The following segmentation analysis shows that even when the verb is singular, segmentation generates the plural suffix:

```
$ echo "okur" | flookup  segment.fst
*okur oku-r-lar
okur oku-r
okur okur
```

In the same way, already segmented word generation of the same word *(invoked by using the flookup command in the inverse analysis mode)*, generates 6 different forms of the verb—3 of which *(marked with *)* forms the plural form the same verb.

```
$ echo "oku-r" | flookup -i  segment.fst
oku-r okur
*oku-r okurlar
oku-r Okur
*oku-r Okurlar
oku-r OKUR
*oku-r OKURLAR
```

In our version of TRmorph+, we removed the null-plural morpheme feature for 3rd person agreement from the analysis of true singular verbs. In the current version, given verbs with 3rd person singular agreement, we get a segmentation analysis only for 3rd person singular agreement.

```
$ echo "okur" | flookup  segment.fst
okur oku-r
okur okur
```

Similarly, if given a segmented form of a verb *(in generation mode)* with 3rd person singular agreement *(e.g. oku-r)*; TRmorph+ returns only the surface-form of verb with the 3rd person singular agreement. In other words, the analyzer no longer generates extra plural surface-forms of the same word form for true singular verbs:

```
$ echo "oku-r" | flookup -i  segment.fst
oku-r okur
oku-r Okur
oku-r OKUR
```

In the analysis mode, similar to the segmentation of word forms, for verbs with 3rd person singular agreement *(e.g. okudu)*, TRmorph-Baseline returned 3rd person plural agreement by default, as in the following example:

```
$ echo "okudu" | flookup  trmorph.fst
*okudu oku<v><t_past><3p>
okudu oku<v><t_past><3s>
```

This was also further complicated by the fact that when we used the lexical forms of words in generation mode to get the corresponding surface word forms, the corresponding word forms did not actually map to the original word forms. For example, when we try to regenerate the surface word form of `oku<v><t_past><3p>` by relying on the lexical form analyses above, we expect to get only `okudu` as the original word form. However, TRmorph-Baseline returns both singular and plural

3rd person agreement features because verbal predicate word formation rules in TRmorph-Baseline mark all verbs as both plural and singular by default. Therefore, the following analyses are generated:

```
$ echo "oku<v><t_past><3p>" | flookup -i trmorph.fst
oku<v><t_past><3p>okudular
*oku<v><t_past><3p>okudu
oku<v><t_past><3p>Okudular
*oku<v><t_past><3p>Okudu
oku<v><t_past><3p>OKUDULAR
*oku<v><t_past><3p>OKUDU
```

Our evaluations and observations have shown that tagging of the null-plural morpheme feature in the absence of an overt plural morpheme not only hurt the evaluation of data seriously by contributing to the overgeneration of analyses; but also it caused confusion when comparing various gold-standard tagsets with the tagset of TRmorph-Baseline. Relying on these observations, in TRmorph+, we have decided to remove the 3rd person null-plural morpheme agreement from the verbal predicates. In the current version of TRmorph+, when we analyze a verb that has 3rd person singular agreement only *(namely, no overt-plural morpheme)*, we currently get only the following analysis:

```
$ echo "okudu" | flookup  trmorph.fst
okudu oku<v><t_past><3s>
```

In the same way, re-generation of `oku<v><t_past><3s>` now maps to the correct word form:

```
$ echo "oku<v><t_past><3s>" | flookup -i trmorph.fst
oku<v><t_past><3s>okudu
oku<v><t_past><3s>Okudu
oku<v><t_past><3s>OKUDU
```

And similarly, when we use the analyzer in the generation mode, we get a plural word form, only when there is an overt-plural agreement; otherwise, in the absence of an overt-plural morpheme or an overt-plural feature, we get 3rd person singular agreement only. For example, regeneration of `oku<v><t_past><3p>` now results only in three *(correct)* word forms:

```
$ echo "oku<v><t_past><3p>" | flookup -i trmorph.fst
oku<v><t_past><3p>okudular
oku<v><t_past><3p>Okudular
oku<v><t_past><3p>OKUDULAR
```

Workflow of TRmorph+

What does TRmorph+ do, what does it not do?
The algorithm we implemented does not do multi-word expression or Named Entity detection in the sense that given unseen data with new words that do *not* exist in the TRmorph lexicons or the FSA dictionary, we cannot infer any prediction from the existing tokens in the lexicon or the dictionary. In general, prediction of multi-word expression tokens and Named Entities is not the task of a morphological analyzer.

Such tasks are often done via more sophisticated NER systems that use statistical measures based on large quantities of data with high frequency of multi-word expression tokens. What TRmorph+ provides is only a tool for the tokenization of multi-word expressions which are untokenized, and their morphological analysis based on already-predicted multi-word expressions *(i.e. in that sense, our method uses already-predicted multi-word tokens that are available in TRmorph+ lexicons).* Even for actual NER systems, or multi-word expression detectors that predict unseen tokens based on a previous training; further morphological analysis of such predicted tokens require proper tokenization before they can be sent for morphological analysis.

Furthermore, in rule-based, and dictionary-based morphological analyzers, predicted tokens must exist in the morphological lexicon for further morphological processing, regardless of how they have been predicted *(i.e. whether their prediction was based on an external NER tool, or multi-word expression detector; or whether they were recognized by the FSA tokenizer, which relies on the availability of tokens in the FSA dictionary in TRmorph+.)* In other words, the NER task or multi-word expression detection is not the task of TRmorph+; however, morphological analysis and processing of already predicted NEs and multi-word expressions, and their tokenization *(if they are untokenized) is* the task of TRmorph+. Figure 5.1 shows the main architecture outlining the core components of TRmorph+:

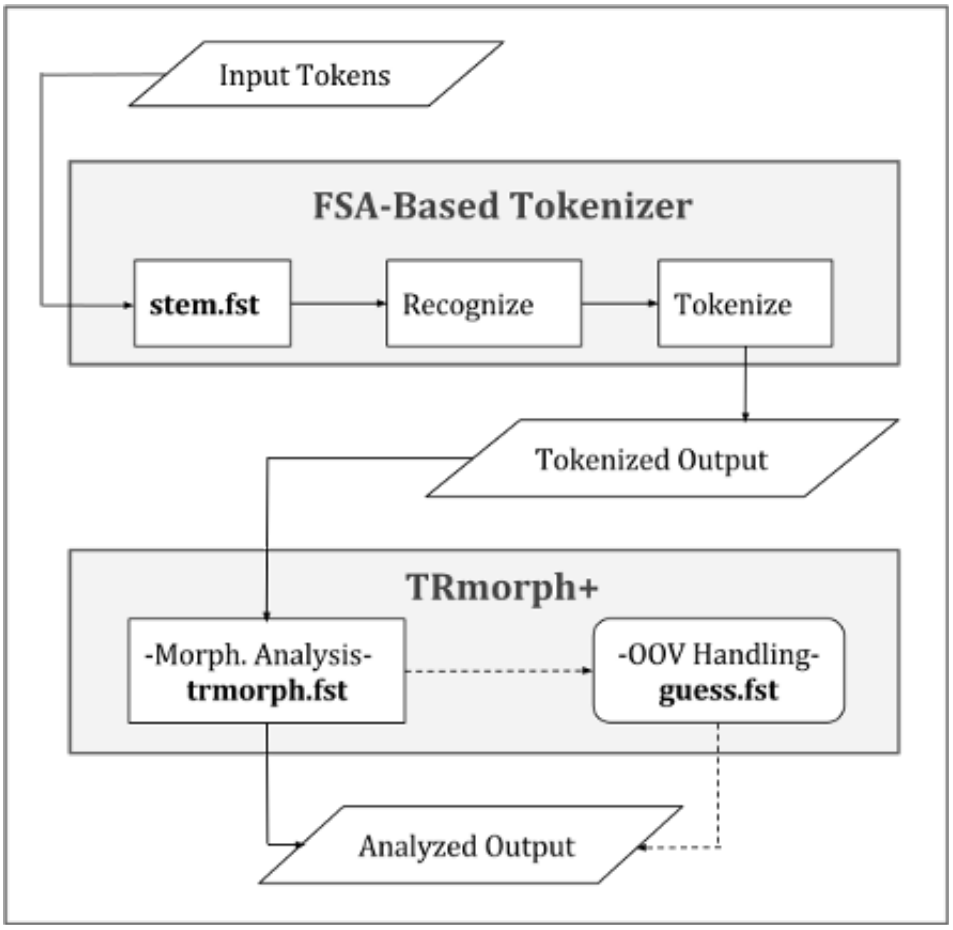


Figure 5.1: TRmorph+ : Main work flow of morphological analysis

According to this framework that outlines the core components of TRmorph+, work flow of tasks in TRmorph+ is ordered as following:

i. First, words go through stemming process

ii. After the stemming task, stemmed words go through FSA recognizer based on the list of multi-word expressions in the FSA dictionary

iii. If there are any input words that match with tokens in the dictionary; then they are collapsed into one 'single line', forming a single-unit multi-word expression token

iv. The new *tokenized output* is generated

v. And then, generated output is sent for morphological analysis.

vi. After the morphological analysis of tokenized output, guessing of unknown tokens is optional.

- Output of analyzed tokens can be obtained either directly as the output of trmorph.fst,
- Or, if any OOV handling of unknown tokens is needed, using TRmorph+ analyzer with a priority union of guess.fst[1], generates the *analyzed output* of tokens that are analyzed by TRmorph+ analyzer, and predicted by guess.fst.

Evaluation of TRmorph+ on Data Sets with Multi-word Expressions

For a general evaluation of TRmorph+, in the next chapter, we introduce a small data set that was manually collected for the processing of sentences with multi-word expressions. Further, we re-evaluate one of the subset of CoNLL data sets, `test.conll`, which provides gold standard annotation of multi-word expressions. Finally, we present results from the evaluation of these data sets, by following up with a discussion of remaining challenges that are still yet to be solved for TRmorph+.

[1]Priority union of two different transducers is enabled by the following command:
`$ flookup -a guess.fst trmorph.fst < tokenized_output_file`
For more details on the usage of TRmorph+, and available list of commands, see the usage guide in the main TRmorph+ directory

Chapter 6

Evaluation of Methods in TRmorph+

For the overall evaluation of the methods introduced in the previous chapter, in this chapter we provide the results of our methods reflected on several data sets. For the evaluation of TRmorph+, in the next sections we give the coverage assessment of two data sets:

i. First, we give the coverage assessment for sentences collected from the dictionary of Turkish Language Institute

ii. Second, we evaluate the CoNLL-Test Set *(which we also had used for the evaluation of the TRmorph-Baseline)*. This time, we give the evaluation results based on actual gold standard annotations of multi-word expressions, as described in *ITU validation set for the Metu-Sabancı Turkish treebank*, prepared by Eryiğit (2007).

6.1 Evaluation of TDK Sentence Data Set with TRmorph+

For the evaluation of our methods, we collected a small data set of sentences from the dictionary of Turkish Language Institute, TDK[1]. These sentences were manually selected from the dictionary of *idioms*. Example sentences, borrowed from Turkish novels, were given in the definitions of these idiomatic multi-word expression verbs. For the construction of our own data set, we collected 1013 sentences *(9942 tokens)* manually, as independent from the collection of multi-word expressions that exist in the TRmorph+ lexicons. In preparation of these

[1] `http://www.tdk.gov.tr/index.php?option=com_atasozleri&view=atasozleri`

sentences for their morphological analysis, we filtered out punctuation marks, but left the empty lines as sentence separators between the sentences. Coverage assessment results given in the table, 6.1, below show:

i. Coverage assessment of tokens based on the total number of single-tokens *(before the multi-word expressions were collapsed into single-tokens)*

ii. Coverage assessment of tokens based on the total number of tokens, including *collapsed* multi-word expressions, processed with TRmorph+

iii. For a direct comparison of TRmorph-Baseline, and the TRmorph+ analyzer, coverage assessments of both single-tokens and collapsed tokens processed with TRmorph-Baseline analyzer are given in the table.

Analyzer	Single Tokens	Collapsed Tokens	Single-Tok. Unknown	Collapsed-Tok. Unknown	Single-Tok. Coverage	Collapsed-Tok. Coverage
TRmorph+	9942	9189	118	237	0.988	0.974
TR-Baseline	9942	9189	125	687	0.987	0.925

Table 6.1: TRmorph+ and TRmorph-Baseline coverage of TDK sentences

In these sentences, due to lack of gold standard annotation of multi-word expressions and other tokens, we cannot provide evaluative measures such as precision, recall, and F-Scores. However, in the table, 6.2, below we give the coverage of multi-word expressions[1], based on the total number of *collapsed* multi-word expression tokens, processed with TRmorph+ and TRmorph-Baseline:

	TRmorph+	TRmorph-Baseline
Total Collapsed MWEs	753	753
Total Analyzed MWEs	622	180
Total Unknown MWEs	131	573
MWE-Token Coverage	0.82	0.23

Table 6.2: TRmorph+ and TRmorph-Baseline coverage of collapsed multi-word expressions

These results show that from the total of 9189 tokens that were processed both with TRmorph+ and TRmorph-Baseline, in both sets, there were 753 already tokenized and collapsed multi-word expression tokens. Out of the 753 multi-word

[1]When given already tokenized/collapsed multi-word expressions as the input

expressions, 131 of them were unknown to TRmorph+, while 573 of them were unknown to TRmorph-Baseline. The number of analyzed tokens processed with each of the analyzers shows also the number of tokens that exist in the lexicons of these analyzers. The numbers of tokens that were *unknown* to either one of the analyzers can be explained by a variety of reasons: Either a given multi-word expression token may not exist in the lexicon, or one of the rules that would normally give the analysis of the token fails for a variety of reasons.

It should be noted that the processes for tokenization and morphological analysis of multi-word expressions are different. This means that the 'number of tokenized *(recognized)* multi-word expressions' may not necessarily be equal to the 'number of analyzed multi-word expressions'. The difference between tokenization and analysis of multi-word expressions comes from the fact that the morphological analysis of multi-word expression *(as well as others)*, requires different sets of solutions to problems that are hard to solve. In the processing of multi-word expressions, the challenge is *not* about the tokenization only—although it is the first problem that needs to be solved. This means that even if a multi-word expression can successfully be recognized and tokenized; tokenization does not necessarily guarantee a way for analysis, because morphological analysis depends on two important morphological levels:
–definition of word-formation process for multi-word expressions
–definition of word-alternation *(morphophonological rules)* process for multi-word expressions.

In rule-based morphological analyzers based on cyclic rules, in order to analyze a multi-word expression, both of the levels of morphological process must be well-defined. If one of the sub-rules that depends on a higher-level rule within the definition of word-formation fails *(e.g. the rule for plural-copula suffixation within a set of rules that define the word-formation of verbs)*, even if the rest of sub-rules or higher-level rules match the word-formation of the input token, the analysis fails. In the same way, even if all the word-formation rules that define the input token match, if one of alternation rules fails *(e.g. final consonant voicing, from /ç/ to /c/)*, then the analysis fails. Likewise, even if the multi-word expression token may be recognized through the recognition algorithm; if the multi-word expression token happens to not exist in the lexicon of the analyzer, the analysis fails regardless of everything else. This is often the downside of rule-based systems, because they require a lot of expertise in the development of rules, and it is often the case that the exact reasons of why an analysis fails may not always be easily known. Therefore, for a successful and complete morphological processing of a multi-word expression token, both the tokenization step, and the morphological analysis step *(word-formation and word-alternations that may be*

involved) must be well-defined.

6.2 Evaluation of CoNLL Test Set using ITU-Validation Gold Standard

The ITU validation set was prepared by Eryiğit (2007), as a subset of the CoNLL data sets. Gold standard annotation of `test.conll` was made available by including the annotation of multi-word expressions according to the list of multi-word expressions in the dictionary of the Turkish Language Institute. This was prepared for the update of METU-Treebank Eryiğit et al. (2011).

The gold standard annotated version of this set consists of the same tokens as those existing in `test.conll`, made of 300 sentences consisting of 3699 single-tokens, and 3609 tokens, including collapsed multi-word expressions *(including punctuations)*. In the version of the gold standard set with collapsed tokens, the number of annotated multi-word expressions is 90. In our own evaluation, in order to see how well TRmorph+ recognizes single-tokens that need to be collapsed into multi-word expressions, we initially evaluated three sets:

i. **Set 0**: Re-evaluation of gold standard collapsed tokens by using the TRmorph-Baseline analyzer[1] for comparison with our methods

ii. **Set 1**: Evaluation of gold standard collapsed tokens, without applying the tokenization method by TRmorph+, which shows us the percentage of already collapsed tokens existing in the TRmorph+ lexicons

iii. **Set 2**: Evaluation of gold standard single-tokens, collapsed by applying our tokenization method by TRmorph+, showing us how well our method performs

After that, we give the statistics for coverage of analyzed and unknown tokens, as well as additional statistics for precision, recall, and Fscore of tokens that were tokenized and analyzed with TRmorph+.

[1]We had given the evaluation of `test.conll` tokens earlier, in Ch. 4, in table 4.11. Here, we repeat the results again, based on the evaluation of the *total* number of tokens collapsed in the gold standard, *including punctuations*,

Coverage of tokens in Set 0, Set 1, Set 2			
	Set 0	Set 1	Set 2
Num. of Single Tokens	3699	3699	3699
Num. of Collapsed Tokens	3609	3609	3581
Num. of Collapsed MWEs	90	90	117
Num. of Unknown Tokens	112	28	50
Token Coverage	0.968	0.992	0.986

Table 6.3: SET0: Analysis of collapsed tokens with Baseline analyzer, SET1: Analysis of collapsed tokens in ITU-Val Set with TRmorph+, SET2: Analysis of single tokens in ITU-Val Set with TRmorph+ & tokenization method

The difference between the number of unknown tokens in Set 1 and Set 2 shows the number of extra 27 multi-word expression tokens, which were collapsed as a result of the tokenization method in TRmorph+, in addition to the 28 tokens[1] that were also unknown in Set 1. This stems from the fact that even though the words are collapsed and tokenized based on their stems, not all of the inflected forms of multi-word expressions exist in the lexicon of TRmorph+. Therefore, even if a multi-word expression may be recognized by applying the tokenization method, there are instances in which the resulting word form may not exist in the fixed lexicon of TRmorph.

For a closer examination of which multi-word expressions are correctly tokenized *in relation to the multi-word expressions existing in the gold standard*, the table below shows the following statistics[2] about the recognition of multi-word expressions by TRmorph+:

Set 2	# of MWE-Tokens
Correct MWEs	40
Missed MWEs	50
`Precision`	0.34
`Recall`	0.44
`Fscore`	0.38

Table 6.4: Additional statistics on multi-word expressions in Set 2

[1] 28 unknown tokens in Set 1 come from a mix of multi-word and single tokens

[2] The statistics above were calculated as follows:
Precision= *Num. of correctly recognized MWEs / total number of MWE found*
Recall= *Num. of correctly recognized MWEs / correct mwes + missing MWEs*,
Fscore= 2PR / P+R

The findings given in table 6.4 show that there is still room for improvement in regards to the tokenization of multi-word expressions that are missed by our method. We were aiming to integrate the method for proper recognition and tokenization of multi-word expressions as compact as possible with TRmorph+. For this reason, we used the stemmer utility that is part of the TRmorph analyzer. In our method, stemming is applied both to the dictionary tokens *(which is part of the FSA tokenizer)*, and the input tokens at runtime. Because the stemming method also depends on the tokens existing in the lexicon; considering the lexical ambiguity of words, TRmorph can return multiple results for a given token. This means that stems for tokens that exist in multiple lexicons result in multiple stem analyses, as in the following example:

```
$ echo "alan" | flookup stem.fst
alan    alan    (field, area)
alan    ala     (pretty)
alan    al      ((to) take)
```

In the initial version of the our method, in case of ambiguity of stems, we always selected the first stem returned by the TRmorph stemmer. However, in the evaluation of Set 2 above, we could observe that in a significant number of cases, the stems of the dictionary tokens and the input tokens *(processed at runtime)* did not match entirely. If the dictionary tokens were trained based on a specific stem—by selecting the first stem analysis in the list of all possible stems for a token; there were instances that the runtime input token stems could end up with different stems than intended. For example, for the multi-word expression token, given as input in runtime 'yer alan' *('taking place')*, the stemming analysis returned for the input token and the dictionary token had a mismatch. In the sequence of this token, we observed the following:

i. input token stem: 'yer alan' → 'yer **alan**'
   'alan' gets the first stem analysis in the list of stems

ii. dictionary token stem: 'yer al' → 'yer **al**'

In order to reduce the number of this kind of mismatched stems and increase the number of matched stems between the input and dictionary tokens, we applied another heuristic to our stemming method. In our new stemming method, we selected the shortest stem out of all possible stems for a token returned by TRmorph. After application of the new stemming method, we re-evaluated the tokens analyzed in Set 2, by creating an evaluation Set 3. Table 6.5 below shows our findings for tokens recognized by using the new heuristic stemming method:

TRmorph+ Coverage of tokens in	Set 3
Num. of Single Tokens	3699
Num. of Collapsed Tokens	3557
Num. of Collapsed MWEs	139
Num. of Unknown Tokens	75
Token Coverage	0.978

Table 6.5: SET3: Analysis of single tokens in ITU-Val Set with TRmorph+ & tokenization, using the shortest-stem selection method

Further, for comparison with the statistics obtained from Set 2, we applied the same measures for our findings from Set 3. Table 6.6 below shows the statistics with precision, recall, and Fscore obtained from Set 3, using the shortest-stem selection method:

Set 3	# of MWE-Tokens
Correct MWEs	43
Missed MWEs	47
`Precision`	0.30
`Recall`	0.47
`Fscore`	0.36

Table 6.6: Additional statistics on multi-word expressions in Set 3

Evaluation results show that most of the unrecognized multi-word expressions result from tokens that have ambiguous stems. In order to avoid the ambiguity and increase the number of matched stems, we tried two different methods:

i. Selecting the first stem of tokens from the list of ambiguous stems

ii. Selecting the shortest stem of tokens from the list ambiguous stems

In both methods, some of the tokens were recognized, and some of them were unknown. By selecting the first stem, *input tokens* could get multiple analyses, and the first stem may be different from the first stem *(or the only available stem)* of the dictionary token. On the other hand, when selecting the shortest stem, the *dictionary tokens* could get multiple stems, and the shortest stem selected may be different from the shortest stem *(or the only available stem)* of the input token. The two following examples show the mismatch of stems when different stemming heuristics are applied:

1. *First stem selection:*
   –input token: `'yer alan'` *(taking place)*→ 'yer alan-' as the first stem from the multiple stem analyses.
   –dictionary token: 'yer al-' → 'yer al-', which is the only available stem.
   This shows that the first stem of *input token* may not always match the first stem of the *dictionary token*—if the *input token* is ambiguous.

2. *Shortest stem selection:*
   –input token: `'ağlamaya başlamış'` *(start crying)* → 'ağlamaya başla-' as the shortest stem *(which is the only available stem)*
   –dictionary token: 'ağlamaya başla-' → 'ağlamaya baş-' as the shortest stem from the multiple stem analyses.
   This shows that the shortest stem of *input token* may not always match the shortest stem of the *dictionary token*—if the *dictionary token* is ambiguous.

This shows that the selection of shortest-stem only does *not* guarantee that inflected forms of input tokens always get multiple stem analyses. Therefore, the selection of shortest stems can be ineffective if the input token does *not* get multiple stem analyses, while the dictionary token gets multiple stem analyses. The example for 'ağlamaya başlamış' shows such difference of mismatched stems between the unambiguous input token, and ambiguous dictionary token. Unfortunately, as discussed before, listing all possible word forms in the dictionary is not feasible because of the high number of potential word forms, caused by productive derivations for each token. Recognition of multi-word expressions helps reducing morphological ambiguity by eliminating a significant number of *(multiple)* morphological analyses; however, context-based stemming is also required in order to avoid ambiguous stems, for a proper tokenization of multi-word expression, and to aid in reducing morphological ambiguity.

Our evaluations and methods show that all of the processes involved in morphological processing are interrelated, and the absence of one method can impact another method in various ways. Our findings show that even if the handling of ambiguous stems remains to be an issue, our current method for recognition and processing of multi-word expressions gives better results over the baseline analyzer. Evaluation results show that the method we have implemented manages to decrease the number of unknown tokens, which was previously higher in the baseline, and at the same time it manages to increase the coverage of tokens over the baseline coverage. As a summary of our findings, table 6.7 below shows the overall results of our method for unknown tokens and coverage obtained from several different evaluation sets:

ITU-Val Sets	Tokens Unknown	Token Coverage
SET 0:	112	*0.963*
SET 1:	28	0.992
SET 2:	50	**0.986**
SET 3:	75	0.978

Table 6.7: Summary of coverage results ITU-Val Set

Looking at these results, it is also worth noting that the number of tokens that are unknown to TRmorph+, and a surplus of tokens that have been analyzed as MWEs by TRmorph+ *(which are not seen in the gold standard annotations)* could be attributed to the data coverage of the treebank and their method of collection of multi-word expressions. As it has been reported by Eryiğit et al. (2011), the list of MWEs was automatically extracted from the dictionary of the Turkish Language Institute (TDK, 2011). And only the words in the treebank whose lemmas matched the lemmas of MWEs from the dictionary were annotated as MWEs in the treebank, as well as in the gold standards. The fact that our collection of multi-word expressions was independent from the MWEs existing in the dictionary of the Turkish Language Institute[1], a certain discrepancy in the coverage could be expected.

Regarding the number of surplus MWEs analyzed as multi-word expressions in TRmorph+, our observation of the data sets and evaluation results showed that there were certain inconsistencies in the gold standard multi-word expression annotations. Certain tokens that were annotated as MWEs in some of the sentences were later not annotated as MWEs in other sentences. This could be explained by the fact that gold standard annotation of MWEs in the treebank was directly depended on the matching of lemmas between the TDK dictionary MWEs and the treebank words. Therefore, if a certain dependency parsing analysis of a previously seen MWE resulted in a different lemma in a different sentence, that word was not annotated as a multi-word expression for the second time. However, TRmorph+ does not evaluate words based on the syntax of sentences, or the recognition of words based on where they occur or what kind of function they have in a sentence. As a result, all occurrences of MWEs were treated the same way by TRmorph+—which caused an increase in the number of tokens that were analyzed as MWEs[2].

[1]Most of our collection of MWEs came from the collection of tokens extracted from Wiktionary

[2]One of the main reasons of low scores obtained in the precision calculations.

Remaining Challenges: Context-Based Disambiguation

As our findings from the previous sections show, morphological ambiguity of tokens causes serious problems not only for finding the best morphological features for a given token *(which is required in higher level tasks such as syntactic parsing)*, but also for determining the best stem based on the context of tokens, which is required for correct tokenization, and morphological processing of multi-word expressions.

As described by Hakkani-Tür et al. (2002), the morphological disambiguation task is responsible for selecting the sequence of morphological parses corresponding to a sequence of words and word forms, given a set of possible parses for each given word. When it comes to the morphological processing of agglutinative languages, the role of morphological disambiguators becomes more complex and challenging than just assigning a single main part-of-speech tag *(such as noun, verb, adj, etc.)* to a word (Eryiğit, 2012). As we can see from the example given in the work by Sak et al. (2007), the Turkish word *'alın'* below receives multiple morphological analyses as output from the morphological analyzer built by Oflazer (1994):

`alın+Noun+A3sg+Pnon+Nom` (forehead)
`al+Adj^DB+Noun+Zero+A3sg+P2sg+Nom` (your red)
`al+Adj^DB+Noun+Zero+A3sg+Pnon+Gen` (of red)
`al+Verb+Pos+Imp+A2pl` ((you) take)
`al+Verb^DB+Verb+Pass+Pos+Imp+A2sg`((you) be taken)
`alın+Verb+Pos+Imp+A2sg` ((you) be offended)

The morphological analyses obtained from TRmorph+ for the same word *'alın'* shows similar analyses:

```
alın al<n><gen>
alın al<n><gen><3s>
alın al<n><p2s>
alın al<n><p2s><3s>
alın al<adj><JN_0><n><gen>
alın al<adj><JN_0><n><gen><3s>
alın al<adj><JN_0><n><p2s>
alın al<adj><JN_0><n><p2s><3s>
alın al<adj><p2s><prn>
alın al<adj><p2s><prn><3s>
alın al<v><t_imp><2p>
alın al<v><t_imp><2s>
alın al<v><pass><t_imp><2s>
alın alın<n>
alın alın<n><3s>
```

```
alın alın<v><t_imp><2s>
```

As we have also seen from the ambiguous word stem examples and their effects on tokenization and morphological analysis; it is clear that morphological disambiguation task is necessary for further improvement of TRmorph+. In order to provide a disambiguator that enables morphological analyses with n-best ranking, and reduce the ambiguity of the Turkish text, we investigated the option as outlined in the *Morphological Disambiguation of Turkish Text with Perceptron Algorithm* by Sak et al. (2007). In their ranking method with the perceptron algorithm for the task of morphological disambiguation, they adopted both the baseline model and the tag representation from the previous work on *Statistical Morphological Disambiguation for Agglutinative Languages* by Hakkani-Tür et al. (2002). For the morphological analysis of the data sets used in training and development, they used the two-level finite-state morphological analyzer by Oflazer (1994).

Given our options with the availability of annotated data sets to use for the training, we decided to use the CoNLL Shared Task Data set. However, use of tagsets different from the tagset of TRmorph+ requires another task, which is *tagset conversion*. As described by Zeman (2010), tagset conversion of data sets, from one corpus to another data set presents its own difficulties. Some of the morphosyntactic features that are well-defined in one set may not necessarily be well-defined in another tagset, or it may not be defined at all. In order to see how well the CoNLL and TRmorph tagset features correspond to one another, we drafted out an initial mapping of features from CoNLL to TRmorph features.

In the planning stage of the initial conversion of tagsets from CoNLL tagsets to the TRmorph+ tagsets, we examined the mapping of morphological features seen in the CoNLL *training* data, and their mapping to the morphological features provided by the TRmorph+ analyzer. Examining the training data in CoNLL, from 70816 tokens *(including sentence separating empty lines and punctuation)*, we found 46010 unique tokens. By extracting the morphological tags from these unique tokens, we obtained 1051 unique morphological tags, showing all three of the CoNLL based annotation of coarse-grained POS tags, POS-tag, and the features for the word form. In order to see which types of features are mappable and not mappable to TRmorph tags, given these features obtained from the `training.conll` set, we attempted to draft a preliminary mapping of features between tagsets. From this initial draft of mapped *(and non-mapped features)*, we found that most of the tag features returned by TRmorph+ were a lot more fine-grained and detailed in comparison to the tagset used in the CoNLL training data. Our initial examination of the tagsets showed the following problematic

features existing in the conversion sets:

1. Person agreement features in CoNLL were not easily mappable to the same features provided in the TRmorph tagsets due to the use of 3rd-person null-morphemes used in the TRmorph tags

2. One single conjunction type, `Conj` in CoNLL mapped to three different conjunction types in TRmorph tagsets: *Adverbial, coordinating, and subordinating conjunctions*

3. Different types of number features existing in CoNLL, such as `Distribution, Cardinal, Real, Ratio, Ordinal` had mapping only to Number, and Ordinal feature types in TRmorph tag sets

4. The feature `ZERO` used in CoNLL to show derivational dependencies had many different mappings in TRmorph tagset, which treats the `ZERO` feature according to their main POS-types as nominal, verbal, adverbial and adjectival, etc.

The examination of the preliminary mapping for features in these sets showed that tagset conversion is not a straightforward task. While converting from TRmorph to CoNLL, for multiple features in TRmorph that correspond to one feature in CoNLL, we could select the single feature that is available (e.g. all of the three *Conj* features in TRmorph map to one *Conj* feature in CoNLL). However, the same thing is not possible while converting from CoNLL to TRmorph tagset, since one single feature in CoNLL can map to multiple features in TRmorph tagsets. Determining which features from one tagset can directly map to the other tagset, and which features must be dropped from both tagsets in order to create a unified tagset that covers all the important features in both tagsets requires a thorough examination of features. For this reason, we leave the remaining investigation for future research.

Chapter 7

Conclusion

In this thesis, we have attempted to give an overview of the morphological tools that are available for processing of Turkish text. However, it is important to note that even though at first look it may seem like there are many tools available for use, most of these tools that are available do not provide all the components required for a complete morphological processing of Turkish text.

Some of the studies we have discussed are either not publicly available, or their data sets are not available. Or, only partial components of tools are publicly available. We see that most of the previous work in morphological analysis of Turkish relies on the morphological analyzer implemented by Oflazer (1994), which is not readily available. On the other hand, the only disambiguation tool (Sak et al., 2007) that is publicly available, is not easily usable by other morphological analyzers because the tool itself relies on the morphological analyzer by Oflazer (1994). As we have attempted to do, adaptation of the disambiguation tool to other morphological analyzers is also not feasible, because annotated, gold standard data required for the training of disambiguator module uses a different tagset *(e.g. the tagset used in the morphological analyzer by Oflazer (1994))* than other tagsets used in publicly available morphological analyzers *(e.g. TRmorph)*.

Therefore, challenges present themselves in the partial availability of tools, and the incompatibility of available tools with one another *(e.g. the TRmorph tagset is not readily usable with the disambiguator by Sak et al. (2007)*. Available tagsets provided with CoNLL differ from TRmorph and other morphological analyzers, and the tagset used in the two-level morphological analyzer by Oflazer (1994) differ in their features—which causes problems with the conversion of tagsets and adaptation of tools. Considering these challenges, we aimed to provide a tool that can work independently of these issues and can be made publicly available with easy access to the data sets used in this thesis. Furthermore, the methodologies

used for the processing of Turkish text could also be adaptable to other Turkic languages that are low-resourced, and thus help development of more language resources and methodologies for those languages.

7.1 Future Work

As future work, and for the improvement of methods we have used in this thesis, we plan on making full use of available language resources to expand the lexicon in TRmorph+ in larger scale. For this, we plan on supporting the TRmorph+ lexicons with the resources from the Turkish Language Institute database. Additionally, methods for open lexicon and user-defined lexicons could be explored. In order to improve the performance of the recognition method for the processing of multi-word expressions, different stemming algorithms that could give more accurate stems could be used.

Since there are several tools available, with each one of them having their own unique tagsets, as one of the top priority future work, methods for the tagset conversion and morphological disambiguation will be explored. In the end, it is our goal that a freely available morphological tool for Turkish can unify all the necessary components of a morphological processing tool, by proving both morphological processing of multi-word expressions, and providing a tool for morphological disambigution. Finally, it is our goal that the methods we explore could be applied to other Turkic languages that can benefit from the availability of a Turkish morphological analyzer.

Appendix A

Turkish Morphology

`bilimcileştiremeyebileceklerimizdenmiydiniz`: The structure of this word became famous as one of the most popular, longest words of Turkish in the linguistics and Turkish grammar circles, *'Çekoslovakyalılaştıramadıklarımızdan_mısınız?'*, which could be translated as *'Are you one of those whom we could not turn into a Czechoslovak?'*. This word lost its popularity due to differing opinions on the discussion of whether the question clitic *-mi* should be considered as a separate unit or not. Later on, after the separation of the two countries, this famous longest word lost its popularity all together, and new longest words were made by the same way of suffixation. For illustration, we show the morphological analysis of this word below:

```
bilim<n><D_CI><n><D_lAS><v><caus><abil><neg><abil><vn_acak><pl><p1p><abl><q><cpl_di><2p>
bilim<n><D_CI><n><D_lAS><v><caus><abil><neg><abil><vn_acak><ncomp><pl><p1p><abl><q><cpl_di><2p>
```

bilim<n><D_CI><n><D_lAS><v><caus><abil><neg><abil><part_acak><pl><p1p><abl><q><cpl_di><2p>

bilim<n><D_CI><n><D_lAS><v><caus><abil><neg><abil><part_acak><ncomp><pl><p1p><abl><q><cpl_di><2p>

bilim<n><D_CI><n><D_lAS><v><caus><abil><neg><abil><t_fut><VJ_0><adj><JN_0><n><pl><p1p><abl><q><cpl_di><2p>

bilim<n><D_CI><n><D_lAS><v><caus><abil><neg><abil><t_fut><VJ_0><adj><JN_0><n><ncomp><pl><p1p><abl><q><cpl_di><2p>

bilim<n><D_CI><adj><D_lAS><v><caus><abil><neg><abil><vn_acak><pl><p1p><abl><q><cpl_di><2p>

bilim<n><D_CI><adj><D_lAS><v><caus><abil><neg><abil><vn_acak><ncomp><pl><p1p><abl><q><cpl_di><2p>

bilim<n><D_CI><adj><D_lAS><v><caus><abil><neg><abil><part_acak><pl><p1p><abl><q><cpl_di><2p>

bilim<n><D_CI><adj><D_lAS><v><caus><abil><neg><abil><part_acak><ncomp><pl><p1p><abl><q><cpl_di><2p>

bilim<n><D_CI><adj><D_lAS><v><caus><abil><neg><abil><t_fut><VJ_0><adj><JN_0><n><pl><p1p><abl><q><cpl_di><2p>

bilim<n><D_CI><adj><D_lAS><v><caus><abil><neg><abil><t_fut><VJ_0><adj><JN_0><n><ncomp><pl><p1p><abl><q><cpl_di><2p>

bilim<n><D_CI><adj><JN_0><n><D_lAS><v><caus><abil><neg><abil><vn_acak><pl><p1p><abl><q><cpl_di><2p>

bilim<n><D_CI><adj><JN_0><n><D_lAS><v><caus><abil><neg><abil><vn_acak><ncomp><pl><p1p><abl><q><cpl_di><2p>

bilim<n><D_CI><adj><JN_0><n><D_lAS><v><caus><abil><neg><abil><part_acak><pl><p1p><abl><q><cpl_di><2p>

bilim<n><D_CI><adj><JN_0><n><D_lAS><v><caus><abil><neg><abil><part_acak><ncomp><pl><p1p><abl><q><cpl_di><2p>

bilim<n><D_CI><adj><JN_0><n><D_lAS><v><caus><abil><neg><abil><t_fut><VJ_0><adj><JN_0><n><pl><p1p><abl><q><cpl_di><2p>

bilim<n><D_CI><adj><JN_0><n><D_lAS><v><caus><abil><neg><abil><t_fut><VJ_0><adj><JN_0><n><ncomp><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><n><D_lAS><v><caus><abil><neg><abil><vn_acak><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><n><D_lAS><v><caus><abil><neg><abil><vn_acak><ncomp><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><n><D_lAS><v><caus><abil><neg><abil><part_acak><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><n><D_lAS><v><caus><abil><neg><abil><part_acak><ncomp><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><n><D_lAS><v><caus><abil><neg><abil><t_fut><VJ_0><adj><JN_0><n><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><n><D_lAS><v><caus><abil><neg><abil><t_fut><VJ_0><adj><JN_0><n><ncomp><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><adj><D_lAS><v><caus><abil><neg><abil><vn_acak><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><adj><D_lAS><v><caus><abil><neg><abil><vn_acak><ncomp><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><adj><D_lAS><v><caus><abil><neg><abil><part_acak><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><adj><D_lAS><v><caus><abil><neg><abil><part_acak><ncomp><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><adj><D_lAS><v><caus><abil><neg><abil><t_fut><VJ_0><adj><JN_0><n><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><adj><D_lAS><v><caus><abil><neg><abil><t_fut><VJ_0><adj><JN_0><n><ncomp><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><adj><JN_0><n><D_lAS><v><caus><abil><neg><abil><vn_acak><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><adj><JN_0><n><D_lAS><v><caus><abil><neg><abil><vn_acak><ncomp><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><adj><JN_0><n><D_lAS><v><caus><abil><neg><abil><part_acak><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><adj><JN_0><n><D_lAS><v><caus><abil><neg><abil><part_acak><ncomp><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><adj><JN_0><n><D_lAS><v><caus><abil><neg><abil><t_fut><VJ_0><adj><JN_0><n><pl><p1p><abl><q><cpl_di><2p>

bilim<n><ncomp><D_CI><adj><JN_0><n><D_lAS><v><caus><abil><neg><abil><t_fut><VJ_0><adj><JN_0><n><ncomp><pl><p1p><abl><q><cpl_di><2p>

List of Tables

References

Ahmet Afsin Akın and Mehmet Dündar Akın. Zemberek, an open source nlp framework for turkic languages. *Structure*, 2007. 23, 24, 28

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. Openfst: A general and efficient weighted finite-state transducer library. In *Implementation and Application of Automata*, pages 11–23. Springer, 2007. URL `http://link.springer.com/chapter/10.1007/978-3-540-76336-9_3`. 28

Nart B Atalay, Kemal Oflazer, Bilge Say, et al. The annotation process in the turkish treebank. In *Proc. of the 4th Intern. Workshop on Linguistically Interpreteted Corpora (LINC)*, 2003. 37, 38

Madeleine Bates. The theory and practice of augmented transition network grammars. In *Natural language communication with computers*, pages 191–254. Springer, 1978. URL `http://link.springer.com/content/pdf/10.1007/BFb0031372.pdf`. 25

Kenneth R. Beesley and Lauri Karttunen. *Finite state morphology*. CSLI Publications, Stanford, Calif., 2003. ISBN 1-57586-433-9. 5, 8, 72

Mathias Creutz and Krista Lagus. Unsupervised models for morpheme segmentation and morphology learning. *ACM Transactions on Speech and Language Processing (TSLP)*, 4(1):3, 2007. URL `http://dl.acm.org/citation.cfm?id=1187418`. 54

Jason Eisner. Parameter estimation for probabilistic finite-state transducers. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 1–8. Association for Computational Linguistics, 2002. URL `http://dl.acm.org/citation.cfm?id=1073085`. 28

Gülşen Eryiğit. Itu validation set for metu-sabancı turkish treebank. *URL: http://www3. itu. edu. tr/ gulsenc/papers/validationset. pdf*, 2007. 63, 87, 90

Gülşen Eryiğit. The impact of automatic morphological analysis & disambiguation on dependency parsing of turkish. In *LREC*, pages 1960–1965, 2012. 63, 96

Gülşen Eryiğit and Eref Adali. An affix stripping morphological analyzer for turkish. In *Proceedings of the IASTED International Conference on Artificial Intelligence and Applications, Innsbruck, Austria*, pages 299–304, 2004. URL `http://www.actapress.com/PaperInfo.aspx?paperId=15800`. 22, 23

Gülşen Eryiğit, Joakim Nivre, and Kemal Oflazer. Dependency parsing of turkish. *Computational Linguistics*, 34(3):357–389, 2008. URL `http://www.mitpressjournals.org/doi/abs/10.1162/coli.2008.07-017-R1-06-83`. 9, 31, 43

Gülşen Eryiğit, Tugay Ilbay, and Ozan Arkan Can. Multiword expressions in statistical dependency parsing. In *Proceedings of the Second Workshop on Statistical Parsing of Morphologically Rich Languages*, pages 45–55. Association for Computational Linguistics, 2011. URL `http://dl.acm.org/citation.cfm?id=2206365`. 48, 60, 90, 95

Tunga Güngör. Lexical and morphological statistics for turkish. *Proceedings of TAINN*, pages 409–412, 2003. 25

Tunga Güngör and Selahattin Kuru. Representation of turkish morphology in atn. In *Proceedings of Second Symposium on Artificial Intelligence and Artificial Neural Networks*, pages 92–104, 1993. 25

Dilek Z Hakkani-Tür, Kemal Oflazer, and Gökhan Tür. Statistical morphological disambiguation for agglutinative languages. *Computers and the Humanities*, 36 (4):381–410, 2002. URL `http://link.springer.com/article/10.1023/A:1020271707826`. 26, 27, 96, 97

Jorge Hankamer. Turkish generative morphology and morphological parsing. In *Second International Conference on Turkish Linguistics. Istanbul, Turkey*, 1984. 21

Jorge Hankamer. Finite state morphology and left to right phonology. In *Proceedings of the West Coast Conference on Formal Linguistics*, volume 5, pages 41–52, 1986. 21

Mehmet Hengirmen. *Türkçe Dilbilgisi*. Engin Yayınevi / Eğitim Dizisi, 2005. 10

Mans Hulden. Foma: a finite-state compiler and library. In *Proceedings of the Demonstrations Session at EACL 2009*, pages 29–32, Athens, Greece, April 2009. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/E09-2008`. 29

Dan Jurafsky, James H Martin, Andrew Kehler, Keith Vander Linden, and Nigel Ward. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*, volume 2. MIT Press, 2000. 4

Ronald M Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational linguistics*, 20(3):331–378, 1994. URL `http://dl.acm.org/citation.cfm?id=204917`. 12, 13

Lauri Karttunen. Kimmo: a general morphological processor. In *Texas Linguistic Forum*, volume 22, pages 163–186, 1983. 21

Lauri Karttunen. Finite-state constraints. In *Proceedings International Conference on Current Issues in Computational Linguistics*, Universiti Sains Malaysia, Penang, 1991. 8

Lauri Karttunen and Kenneth R Beesley. Twenty-five years of finite-state morphology. *Inquiries Into Words, a Festschrift for Kimmo Koskenniemi on his 60th Birthday*, pages 71–83, 2005. 21

Celia Kerslake and Asli Göksel. *Turkish: A Comprehensive Grammar*. Comprehensive Grammars. Routledge (Taylor and Francis), New York, 2005. 10, 11

Kimmo Koskenniemi. A general computational model for word-form recognition and production. In *Proceedings of the 10th international conference on Computational linguistics*, pages 178–181. Association for Computational Linguistics, 1984. URL `http://dl.acm.org/citation.cfm?id=980529`. 21, 27

Mikko Kurimo, Sami Virpioja, Ville Turunen, and Krista Lagus. Morpho challenge competition 2005–2010: evaluations and results. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, pages 87–95. Association for Computational Linguistics, 2010a. URL `http://dl.acm.org/citation.cfm?id=1870489`. 37, 50, 56

Mikko Kurimo, Sami Virpioja, Ville T Turunen, et al. Proceedings of the morpho challenge 2010 workshop. In *Morpho Challenge Workshop; 2010; Espoo*. Aalto

University School of Science and Technology, 2010b. URL https://aaltodoc.aalto.fi/handle/123456789/827. 38

Krister Lindén, Erik Axelson, Senka Drobac, Sam Hardwick, Miikka Silfverberg, and Tommi A Pirinen. Using hfst for creating computational linguistic applications. In *Computational Linguistics*, pages 3–25. Springer, 2013. URL http://link.springer.com/chapter/10.1007/978-3-642-34399-5_1. 29

Martin Majliš. W2c–web to corpus–corpora. 2011. 38, 41

Martin Majliš and Zdeněk Žabokrtský. Language richness of the web. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC 2012)*, 2012. 38, 41

Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*, volume 999. MIT Press, 1999. 54

Jens Nilsson, Sebastian Riedel, and Deniz Yuret. The conll 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, pages 915–932. sn, 2007. 38, 43

Kemal Oflazer. Two-level description of turkish morphology. *Literary and linguistic computing*, 9(2):137–148, 1994. URL http://llc.oxfordjournals.org/content/9/2/137.short. 21, 22, 26, 27, 28, 96, 97, 99

Kemal Oflazer, Elvan Göçmen, Elvan Gocmen, and Cem Bozsahin. An outline of turkish morphology. 1994. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.6951. 21

Kemal Oflazer, Bilge Say, Dilek Zeynep Hakkani-Tür, and Gökhan Tür. Building a turkish treebank. In *Treebanks*, pages 261–277. Springer, 2003. URL http://link.springer.com/chapter/10.1007/978-94-010-0201-1_15. 37, 38

Kemal Oflazer, Bilge Say, et al. Integrating morphology with multi-word expression processing in turkish. In *Proceedings of the Workshop on Multiword Expressions: Integrating Processing*, pages 64–71. Association for Computational Linguistics, 2004. URL http://dl.acm.org/citation.cfm?id=1613195. 60, 62, 72

Uwe Quasthoff, Matthias Richter, and Chris Biemann. Corpus portal for search in monolingual corpora. In *Proceedings of the fifth international conference on language resources and evaluation*, pages 1799–1802, 2006. 51

Haşim Sak, Tunga Güngör, and Murat Saraçlar. Morphological disambiguation of turkish text with perceptron algorithm. In *Computational Linguistics and Intelligent Text Processing*, pages 107–118. Springer, 2007. URL `http://link.springer.com/chapter/10.1007/978-3-540-70939-8_10`. 27, 28, 96, 97, 99

Haşim Sak, Tunga Güngör, and Murat Saraçlar. A stochastic finite-state morphological parser for turkish. In *Proceedings of the ACL-IJCNLP 2009 Conference short papers*, pages 273–276. Association for Computational Linguistics, 2009. URL `http://dl.acm.org/citation.cfm?id=1667667`. 27, 28

Bilge Say, Deniz Zeyrek, Kemal Oflazer, and Umut Özge. Development of a corpus and a treebank for present-day written turkish. In *Proceedings of the eleventh international conference of Turkish linguistics*, pages 183–192, 2002. 38

Helmut Schmid. A programming language for finite state transducers. In *FSMNLP*, volume 4002, pages 308–309, 2005. 29

Stuart C Shapiro. Generalized augmented transition network grammars for generation from semantic networks. *Computational Linguistics*, 8(1):12–25, 1982. URL `http://dl.acm.org/citation.cfm?id=972925`. 25

Max Silberztein, Tamás Váradi, and Marko Tadić. Open source multi-platform nooj for nlp. In *COLING (Demos)*, pages 401–408, 2012. 24

William A Woods. Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606, 1970. URL `http://dl.acm.org/citation.cfm?id=362773`. 25

Deniz Yüret and Ferhan Türe. Learning morphological disambiguation rules for turkish. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 328–334. Association for Computational Linguistics, 2006. URL `http://dl.acm.org/citation.cfm?id=1220877`. 26, 27

Daniel Zeman. Hard problems of tagset conversion. In *Proceedings of the Second International Conference on Global Interoperability for Language Resources*, pages 181–185, 2010. 97

Çağrı Çöltekin. A freely available morphological analyzer for turkish. In *LREC*, 2010. 2, 19, 28, 31, 49