TRmorph: A morphological analyzer for Turkish

Çağrı Çöltekin

Draft: August 4, 2013

This document describes the new/development version of TRmorph. As such, there may be some mismatches between what is documented here and how the analyzer behaves. This version is a complete overwrite of the previous version reported in Çöltekin 2010. If you are using the older version (you shouldn't), this document is probably useless for you.

1 Introduction

TRmorph is an open-source[1] finite-state morphological analyzer for Turkish. This document describes how to use the tools that comes with this package, as well as some implementation details that may be helpful for people who want to customize this open-source tool for their own needs. The complete source of the analyzer and a web-based demo can be accessed through `http://www.let.rug.nl/coltekin/trmorph`.

This document describes the current version of the analyzer. This version is a complete rewrite of the earlier version report in Çöltekin 2010. The earlier version of TRmorph was implemented using SFST (Schmid 2005), the current version is implemented with more popular finite state description languages *lexc* and *xfst* from Xerox (Beesley and Karttunen 2003), using Foma (Hulden 2009) as the main development tool.

The lexc/xfst implementation of TRmorph should compile with any lexc/xfst compiler without much additional effort. The only foma-specific notation used in the morphology description is about handling simple reduplication, which can also be handled with twolc rules, or compile-replace (Beesley and Karttunen 2003).[2]

[1] Current version of TRmorph is licensed under GNU Lesser General Public License. See README file in the TRmorph distribution for more information.

[2] TRmorph can be compiled with HFST (Lindén et al. 2009) without modification since HFST uses foma as the back end for parsing xfst files.

2 How to use it

2.1 Compilation from the source

To compile TRmorph from the source, you need a lexc/xfst compiler such as foma, a C preprocessor, GNU make and some standard UNIX utilities.

If all requirements are in place, to build analyzer/generator FST, you should type `make` in the main TRmorph distribution directory. The resulting binary automaton file will be `trmorph.fst`.

Trmorph comes with a set of other finite-state tools that are useful in various NLP tasks. Currently, tolls for the following tasks are distributed together with TRmorph.

- stemming/lemmatization
- morphological segmentation
- hyphenation
- guessing unknown words

To compile these tools, you should specify the FST you want to build as an argument to `make`, e.g., `make stemmer` will build an binary automaton called `stem.fst`. These additional tools are described in Section 6.

2.2 Customizing TRmorph

TRmorph is an open source utility. As a result, you are free to modify the source according to your needs. Source

code includes some useful comments on what/how/where things are done. Furthermore, TRmorph can be customized for some common choices during the compilation. These options are typically related to more relaxed analysis. For example whether to allow non-capitalized proper names, or analyze (and generate) text written in all capitals, or set the decimal and thousand separator in numbers. These options are set in the file `options.h`. The file contains documentation along with the existing options. This feature is under development (as of July 2013), new options are currently being added, and existing options may not fully work as intended yet.

Another common need for customizing a morphological analyzer is to add or modify the lexical entries. The lexicon structure and format of the lexical entries are described in Section 4.

2.3 Trying it out

Assuming you have built the binary `trmorph.fst` using foma, you can simply start foma and use xfst commands implemented in foma to analyze and generate the words. Here is an example session:

```
$ foma
foma[0]: regex @"trmorph.fst";
2.1 MB. 62236 states, 135237 arcs, Cyclic.
foma[1]: up okudum
oku<V><past><1s>
foma[1]: down oku<V><past><2s>
okudun
```

The first line is typed at the shell prompt to start foma. The second line reads the FST specified in `trmorph.fst` into the foma environment. The fourth line asks for the analysis of the verb *okudum* 'I read-PAST', and the fifth line is the output of the analysis. The sixth line asks for the generation of the analysis string produced earlier, modifying the agreement marker to second person singular agreement.

Note that part of the output is removed for readability. We should also note that this example presents one of the rare cases where the analysis is unambiguous. Turkish morphological analysis is an ambiguous process, and TRmorph does not try to avoid it during the analysis (see Section 5).[3]

Once you are convinced that the output may be useful for your purposes, you will probably want to use it for analyzing large amount of text. For batch analysis tasks foma's `flookup` utility is a better fit.

To use the analyzer with the HFST, you need to compile the source automaton with HFST tools.

3 The tagset

The description of the morphology in TRmorph mostly follows Göksel and Kerslake 2005. However, there are some divergences, and tags used in TRmorph analyses does not necessarily match with any of the tags used in any grammar book. This section describes the tags used in the current version of TRmorph. The aim of this section is to help users understand the output of the system. Occasional discussion of the morphological process is included, but this section documents neither the morphology of the language nor the way it is implemented in TRmorph. Our focus in this section is to describe the tags one finds in the analysis strings produced by the analyzer (or tags one needs to use for generation). The index at the end of the document also allows easy access to the points where a particular tag is defined or mentioned in this document.

A clarification of the notation for the surface forms is in order before starting the documentation of the tagset and related suffixes. Suffixes in Turkish often contain under-specified vowels and consonants that are resolved according to morphophonological rules, like vowel harmony. These vowels and consonants are indicated with capital letters listed below.

A is realized as either 'a' or 'e'.

I is realized as either 'ı', 'i', 'u' or 'ü'.

D is realized as either 'd' or 't'.

P is realized as either 'p' or 'b'.

K is realized as either 'k', 'ğ' or 'y'.

[3] For most purposes, the output of the morphological analyzer needs to be disambiguated. There are quite a few morphological disambiguators for Turkish reported in the literature, but, as yet, there are no disambiguators that work with TRmorph output.

C is realized as either ‘c’ or ‘ç’.

A letter in parentheses indicate a buffer consonant or vowel, that may be dropped in certain contexts.

3.1 Part-of-speech tags

All analysis strings include at least one POS tag. A word may get multiple POS tags, and even the same POS tag repeatedly. This process tracks the changes to the syntactic function of a word with suffixation. It follows the idea of *inflectional groups* (IGs, Oflazer, Hakkani-Tür, et al. 1999; Oflazer, Say, et al. 2003) used in the Turkish NLP literature. In a nutshell, the suffixation process may result in changes to the syntactic function of a word, and each of these functions may have their own ‘inflectional’ features. The POS tag of the whole word is the final POS tag in the sequence. However, the other intermediate IGs are also important, and, at least in typical dependency parsing, these IGs may participate in dependency relations with the other words in the sentence. Example (1) demonstrates this process with the analysis of of the word *evdekilerinki*.[4]

(1) ev⟨N⟩⟨loc⟩⟨ki⟩⟨Adj⟩⟨0⟩⟨N⟩⟨pl⟩⟨gen⟩⟨ki⟩⟨Adj⟩

The example analysis in (1) includes the following IGs.

1. The initial noun ‘ev’ with the locative maker.
2. Addition of the suffix -ki makes an adjective.
3. The adjective becomes a (pro)nominal with a zero derivation, which is inflected for plural and genitive case.
4. Yet another -ki is suffixed, and the word becomes an adjective again.[5]

By default, TRmorph does not mark IG boundaries explicitly. However, one can easily trace the IG changes following the POS tags. All POS tag names start with a capital letter, while other tags always start with a lowercase letter or number. The tag before a new POS tag is always preceded with the tag of the suffix that cause the derivation. When there is no surface suffix, a zero-derivation tag ⟨0⟩ is inserted before the POS tag.

The example in (1) is also interesting because of the fact the suffix -ki may result indefinitely long words (see Section 3.4).

All part-of speech tags used in TRmorph are listed in Table 1. Most POS tags are self explanatory, and does not require much explanation. The following part of speech tags are somewhat unusual and deserves some explanation.

⟨Exist⟩ is used for two words *var* ‘existent/present’ and *yok* ‘non-existent/absent’, where the latter is marked as ⟨Exist:neg⟩, indicating that it is the negative form (see Section 3.2, for the details of this notation). These words behave mostly like nouns in their predicate function (with zero copula), but marking them simply as nouns does blur their function.

⟨Not⟩ is used for *değil* ‘not’ only. Like *var* and *yok*, *değil* also behaves like nominal predicates. But again, marking it as noun or verb hides the fact that it has a special function.

Table 1: The list of part of speech tags in TRmorph.

Tag	Description
⟨Alpha⟩	Symbols of the alphabet
⟨Adj⟩	Adjective
⟨Adv⟩	Adverb
⟨Cnj⟩	Conjunction
⟨Det⟩	Determiner
⟨Exist⟩	The words *var* and *yok*
⟨Ij⟩	Interjection
⟨N⟩	Noun
⟨Not⟩	The word *değil*
⟨Num⟩	Number
⟨Onom⟩	Onomatopoeia
⟨Postp⟩	Postposition
⟨Prn⟩	Pronoun
⟨Punc⟩	Punctuation
⟨Q⟩	Question particle *mI*
⟨V⟩	Verb

[4] Rough translation: ‘the *ones* that belong to the *ones* in the house’, as in ‘the *books* that belong to the *people* in the house’.

[5] More likely reading of this example includes another zero derivation causing final POS to be again noun.

⟨Q⟩ is used for the question particle -mI. The question particle is written separately from the predicate it modifies. However, the preferred analysis of question particle in TRmorph is together with the predicate. This ensures that it follows the correct form of the predicate it is attached to, and vowel harmony is applied correctly. However, since we do not assume that the input is tokenized with this assumption, this form make sure that the input is analyzed with the cost of precision. The question particle is discussed further in Section **??**.

3.2 Subcategorization of lexemes

Besides the major major POS tags or word classes discussed above, TRmorph makes use of a set of subcategory tags to mark features that are part of a lexeme. Typically the subcategorization is applied to a root form in the lexicon, but some morphemes and POS tags after a derivation may also receive a subcategory tags. Subcategories defined here are features of a morpheme that do not have a surface realization. Representing these features using a different notation allows one to make this distinction, and the surface–analysis mapping becomes (almost) one-to-one. If a representation where all tags have a uniform notation is desired, the analyzer source can be modified accordingly, or easier, a simple regular expression based converter can be used.

The subcategories generally mark semantic differences, but they may also result in morphosyntactic differences. Lexical subcategorization in TRmorph output is marked using the syntax ⟨Cat:subcat$_1$:subcat$_2$:…⟩, where 'Cat' is a major category and 'subcat$_1$', 'subcat$_2$' and so on are sub categories. The order of subcategory tags are not important (although they are produced in a consistent order). A typical example of a subcategory is *proper* nouns, which are tagged as ⟨N:prop⟩.

The following lists subcategories used in TRmorph for all word classes that may be specified together with a subcategory.

Nouns Besides the tag ⟨N:prop⟩ marking proper names, abbreviated nouns are marked with the tag ⟨N:abbr⟩. For an abbreviated proper name, the tag is ⟨N:prop:abbr⟩.

Conjunctions are subcategorized as *coordinating*, *adverbial* or *subordinating* conjunctions, marked using tags ⟨Cnj:coo⟩, ⟨Cnj:adv⟩, ⟨Cnj:sub⟩ respectively.

The last one of these categories, ⟨Cnj:sub⟩, include only a limited set of conjunctions which come first in a subordinate clause. These words currently are *ki*, *eğer* and *şayet* (all borrowings from Persian). The other subordinating particles/words occur at the end of subordinate clauses, and they are marked as postpositions (⟨Postp⟩) described below. Furthermore, most of the subordination in Turkish is done through suffixation which is described in Section 3.14.

Pronouns Pronouns are further categorized as *personal*, *demonstrative* and *locative* pronouns, marked using ⟨Prn:pers⟩, ⟨Prn:dem⟩, ⟨Prn:locp⟩ respectively. Furthermore, the pronouns that form questions, like *kim* 'who', and *ne* 'what', are marked as ⟨Prn:qst⟩. Subcategory markers for both aspects can be present. For example *kim* 'who' would be marked as ⟨Prn:pers:qst⟩.

Besides the above subcategories, personal pronouns get person-number agreement markers. These markers can be useful in subject-predicate agreement as well as in other constructions (such as genitive-possessive construction involving pronouns). However, the agreement in Turkish is far from trivially determined (see Göksel and Kerslake 2005, pp.116–122). The markers ⟨Prn:pers:1s⟩, ⟨Prn:pers:2s⟩, ⟨Prn:pers:3s⟩, ⟨Prn:pers:1p⟩, ⟨Prn:pers:2p⟩ and ⟨Prn:pers:3p⟩ are tags used for the personal pronouns with person-number agreement. The agreement markers are further discussed in Section 3.11.

The reflexive pronoun *kendi(si)* is marked as ⟨Prn:pers:refl⟩.

Subcategorization of pronouns, particularly as personal pronouns, are sometimes not a clear decision. Subcategories of some pronouns are left unspecified even though they are often used as personal pronouns, and some pronoun marked as personal pronouns may refer to entities other than people.

Determiners are marked for definiteness. *Definite* determiners are marked ⟨Det:def⟩ and *indefinite* de-

terminers are marked ⟨Det:indef⟩. The question words that fill the same syntactic slot as determiners *ne kadar* 'how much' and *hangi* 'which' are tagged with ⟨Det:qst⟩.

Further subcategorization of determiners (for example quantifiers) can be implemented in the future.

Postpositions are always subcategorized in two dimensions. First subcategory is the syntactic category (POS) of the resulting postpositional phrase, either an *adjectival* or *adverbial* phrase, marked as ⟨Postp:adj⟩ and ⟨Postp:adv⟩ respectively. Note that unlike other POS tags, these subcategory markers start with a lowercase letter.

Postpositions choose their noun phrase complements. Besides the category of the resulting phrase, postpositions also include a tag specifying the requirement for the complement noun phrase. The tag marking required complement type is formed by a concise description of the requirement followed by the capital letter 'C'. The postpositions that require the complement to be in *ablative*, *dative* and *instrumental* cases are marked ⟨Postp:ablC⟩, ⟨Postp:datC⟩, and ⟨Postp:insC⟩ respectively. Postpositions that require non-case marked complement are tagged ⟨Postp:nomC⟩. The postpositions that require the noun phrase to be suffixed with either -Il or -sIz are marked with ⟨Postp:liC⟩.[6] Finally, postpositions that require numeric expressions as their complements are marked with ⟨Postp:numC⟩.

Numbers are tagged as ⟨Num:ara⟩ for Arabic numerals, and ⟨Num:rom⟩ for Roman numerals. Numbers that are spelled out are not marked with a subcategory marker (but still marked as ⟨Num⟩). Besides numbers, the question word *kaç* 'how many' is also tagged as a number with a sub tag specifying that it is a question word, resulting in ⟨Num:qst⟩.

Verbs are currently not subcategorized in TRmorph.

Subcategorizing verbs as *transitive* and *intransitive*, or marking all types (cases) of noun phrase complements a verb can take is planned and some early steps are underway as of this writing (July 2013).

[6] Like -(y)lA that we mark as ⟨ins⟩, these suffixes are typically considered derivational suffixes, however their use resemble case markers.

Adverbs are not currently subcategorized, except a few adverbial question words for which the tag ⟨Adv:qst⟩ is used.

Exist The tag ⟨Exist⟩ exists only for two words *var* 'existent/present' and *yok* 'non-existent/absent'. Since *yok* is the negative of *var*, it is tagged as negative: ⟨Exist:neg⟩.

Some verbs, nouns, adjectives, adverbs and conjunctions are formed by more than one written words. Some of these are adjacent words, like the adverb *apar topar* 'hurriedly', but some may be split like the conjunction *ya*, as in *ya evdedir ya iş yerinde* 's/he is either at home or the office'. Furthermore, some of individual 'words' in such constructions cannot be used by themselves, like *topar* above. If the non-split multi-word expressions are input to the analyzer together, they are analyzed like other words of the same class. However, if they are input word-by-word, a sub tag ⟨:partial⟩ is added to the main POS tag. For example *apar* and *topar* are tagged as ⟨Adv:partial⟩ and *ya* is tagged as ⟨Cnj:partial⟩ (more precisely ⟨Cnj:coo:partial⟩). Currently, the tags ⟨N:partial⟩, ⟨Adj:partial⟩ and ⟨V:partial⟩ are used for parts of nouns, adjectives and verbs respectively.

3.3 Nominal morphology and noun inflections

Nouns, pronouns, adjectives and adverbs in Turkish form the larger class of nominals. Most adverbs and adjectives can function as nouns (or pronouns). For example *mavi* 'blue' may mean 'the blue one', or *hızlı* 'fast' may mean 'the fast one'. In TRmorph this is handled by allowing any adjective or adverb to become an noun with a *zero derivation*. A zero derivation is always marked with the tag ⟨0⟩ followed by the new POS tag, in this case ⟨N⟩.

Nouns can be suffixed with the plural suffix, one of the possessive suffixes and one of the case suffixes. All of these inflections are optional. When not marked with any of these suffixes, the default is singular, no possessive marking, an no case marking (or nominal), respectively. When these suffixes co-occur, they have to occur in the order listed, shown in Figure 1. The full list of noun inflections are presented in Table 2.

If there is a plural marker, analysis string after the ⟨N⟩ will include the tag ⟨pl⟩. TRmorph does not mark for

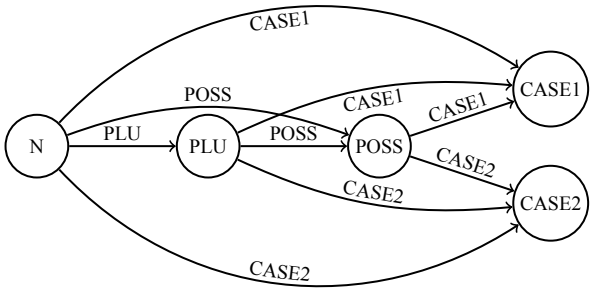

Figure 1: Automata depicting noun inflections. The edge CASE1 represents the locative and ablative suffixes, CASE2 represents all other case-like suffixes. The reason for the differentiation is due to the fact that the state CASE1 can be followed by the suffix -ki.

Table 2: Noun inflections.

	Function	surface	tag
	Plural	-lAr	⟨pl⟩
Possessive	First person singular	-(I)m	⟨p1s⟩
	Second person singular	-(I)n	⟨p2s⟩
	Third person singular	-(s)I	⟨p3s⟩
	First person plural	-(I)mIz	⟨p1p⟩
	Second person plural	-(I)nIz	⟨p2p⟩
	Third person plural	-lArI	⟨p3p⟩
Case	Accusative	-(y)I	⟨acc⟩
	Dative	-(y)A	⟨dat⟩
	Ablative	-DAn	⟨abl⟩
	Locative	-DA	⟨loc⟩
	Genitive	-(n)In	⟨gen⟩
	Instrumental/commutative	-(y)lA	⟨ins⟩

singular. If a noun is not marked for plural, it is assumed to be singular.

Possessive markers follow either the nominal stem, or the plural marker. Besides marking for possession, these suffixes have a number of other functions, such as deriving pronouns when attached to adjectivals (e.g., determiners or adjectives).

The first five suffixes in Table 2 are commonly recognized cases in Turkish. The *instrumental/commutative* marker also behaves like case suffixes. There are two more suffixes, namely -lI and -sIz that can occupy the same slot, which are marked with tags ⟨li⟩ and ⟨siz⟩ respectively.

The -(s)I suffix, listed as ⟨p3s⟩ in Table 2, is highly ambiguous. One of its many functions that may be confused with the possessive suffix is forming noun compounds. One of types of noun compounds in Turkish is formed by a bare noun and a (head) noun with suffix -(s)I, for example *at arabas*ı 'horse carriage'.[7] TRmorph marks this usage of the suffix -(s)I as ⟨ncomp⟩. In this use, this suffix always causes ambiguities. Besides the fact that a noun suffixed with -(s)I can either be marked for possession or as the head of a noun compound, since one of the two -(s)I suffixes following each other is deleted from the surface form, it can also be both (a noun compound marked for possession).

The case (or case-like) suffixes change the role of the noun (or the noun phrase headed by the noun) in the sentence. For example a locative marked noun phrase may function as an adverb (*saat dokuz*da *görüşürüz*) or an adjective (*yedi yaşın*da *çocuk*). However, following the common practice in the literature we do not attempt to mark possible POS changes after case-like markers.

3.4 The suffix -ki

The suffix -ki, tagged as ⟨ki⟩, attaches to locative or ablative marked nouns. The resulting word functions as an adjective or a pronoun. In both cases, TRmorph marks the transition to an adjective. For example, *evdeki* is analyzed as 'ev⟨N⟩⟨loc⟩⟨ki⟩⟨Adj⟩'. Since all adjectives are allowed to become a noun through a zero derivation, the pronoun reading is intended to be represented by this change. For example, the intended analysis for *evdeki kitap* 'the book in the house' is 'ev⟨N⟩⟨loc⟩⟨ki⟩⟨Adj⟩', while analysis for *evdeki uyuyor* 'the one/person in the house is sleeping' appends '⟨0⟩⟨N⟩' at the end of the analysis string.

[7] Even though one can assume that this use is somewhat related to possession, it is not strictly possessive marking (the horse does not own the carriage). Furthermore, although the surface suffix is skipped after a compound marker, a compound of this form may also be in possessive form (e.g., 'someone's horse carriage').

The (pro)noun formed by -ki can further be suffixed with other nominal suffixes. Although the number of iterations using -ki rarely exceed 2 in practice, there is no principled limit. As a result, length of a Turkish word is unbounded in-principle.

3.5 Tags related to nominal predicates

Any nominal in Turkish may become a predicate with one of the copular suffixes -(y)DI, -(y)mIş, -(y)sA or -(y). These suffixes correspond to *past*, *evidential*, *conditional*, and *present* predicates involving the copula 'be'. The copular markers has to precede one of the verbal person agreement markers. For example *öğrenciy*di*k* 'we were students', *öğrenciy*miş*ler* 'they were [evidentially] students', *öğrenciy*se*n* 'if you are/were a student', *öğrenciyim* 'i'm a student'. Since the third person singular agreement suffix is null on the surface and the buffer -(y)- does not surface in this case, any nominal without additional copular or person suffixes serve as a nominal predicate with present copula and third person singular agreement. Additionally, since a predicate with third person singular agreement also agrees with a third person plural subject, we additionally mark such a noun as having present copula and third person plural agreement (for example, *babam öğretmen, annem ve ablam doktor* 'my father is a teacher, my mother and older sister are doctors').

TRmorph handles this process by allowing any noun to first became a verb by a zero derivation, and then marking it with the appropriate copula and the person agreement marker. The tags for copula are ⟨cpl:pres⟩, ⟨cpl:past⟩, ⟨cpl:evid⟩ and ⟨cpl:cond⟩ for *present*, *past*, *evidential* and *conditional* copula respectively. Last three tags are also possible after a verb with a tense/aspect/modality suffix, and is discussed further in Section 3.12. Example analyses for the examples discussed above would be as follows:

öğrenciydik	⟨N⟩⟨0⟩⟨V⟩⟨cpl:past⟩⟨1p⟩
öğrenciymişler	⟨N⟩⟨0⟩⟨V⟩⟨cpl:evid⟩⟨3p⟩
öğrenciysen	⟨N⟩⟨0⟩⟨V⟩⟨cpl:cond⟩⟨2s⟩
öğrenciyim	⟨N⟩⟨0⟩⟨V⟩⟨cpl:pres⟩⟨1s⟩
öğretmen	⟨N⟩⟨0⟩⟨V⟩⟨cpl:pres⟩⟨3s⟩
doktor	⟨N⟩⟨0⟩⟨V⟩⟨cpl:pres⟩⟨3p⟩

Besides copular suffixes, the suffix -(y)ken (making adverbials from verbs, discussed in Section 3.14) may occupy the same slot as the copular suffixes, although its use is more restricted.

The nominal predicate with a copula and person agreement may be followed by the marker Göksel and Kerslake 2005 call 'generalizing modality marker', the suffix -DIr. It is particularly common with ⟨3s⟩ as it disambiguates between the noun and the predicate reading. The tag for this marker in TRmorph is ⟨dir⟩.

3.6 Number inflections

The suffix -(ş)Ar, tagged ⟨dist⟩, attached to numbers form *distributive* numerals. Besides the numbers (written as numerals or spelled out), question word *kaç* 'how many' may also get this suffix, and tagged with ⟨dist⟩.

The ordinal numerals are formed using the suffix –(I)ncI, and tagged as ⟨ord⟩. Ordinals are also specified by a 'dot' after Arabic or Roman numerals. TRmorph currently does not handle this notation.

Percent sign before a numeral is treated like a prefix, and tagged as ⟨perc⟩.

3.7 Verbal voice suffixes

Turkish verbs can be suffixed with one or more of the voice suffixes *reflexive*, *reciprocal*, *causative* and *passive*. The tags used for these functions are ⟨rfl⟩, ⟨rcp⟩, ⟨caus⟩ and ⟨pass⟩, respectively. The first two are rather unproductive while causative and passive forms are productive. Furthermore, causative suffix can be used repetitively.[8] With some verbs, use of double causative suffix yields the same semantics as a single causative suffix. TRmorph does not treat these cases separately. If surface string has double causative suffixes, the analysis will include two ⟨caus⟩ tags, regardless of its semantics.

Despite the fact that most grammar books list voice suffixes under inflectional morphology, TRmorph treats them as derivations, i.e., a ⟨V⟩ tag follows the voice related tags.

3.8 Compound verbs

A verbal stem (possibly including voice suffixes) may be followed by a set of suffixes that form compound

[8] Again, although this is limited in practice, there is no principled limit on the number of causative suffixes that one can string one after another.

Table 3: Suffixes that make compound verbs.

Suffix	Tag	Expresses
-(y)Abil	⟨abil⟩	ability
-(y)Iver	⟨iver⟩	immediacy
-(y)Agel	⟨agel⟩	habitual/long term
-(y)Adur	⟨adur⟩	repetition/continuity
-(y)Ayaz	⟨ayaz⟩	almost
-(y)Akal	⟨akal⟩	stop/freeze in action
-(y)Agör	⟨agor⟩	somewhat like ⟨iver⟩

verbs. These suffixes are related to some stand-alone verbs. These suffixes are listed in Table 3.

The first three suffixes in this Table 3 are relatively productive, the others are rare or their use are mostly lexicalized. Although not frequent in use, more than one these suffixes may attach to the same stem, for example *çıkıverebilir* 'he/sh/it may possibly come out/show up' analyzed as 'çık⟨V⟩⟨iver⟩⟨abil⟩⟨aor⟩⟨3s⟩'.

The form of ⟨abil⟩ in a negative verb is -(y)A, and unlike the rest of the suffixes listed in Table 3 it follows the negative marker.

Currently, these suffixes are not marked as derivations (they do not cause an IG change).

3.9 Negative marker

Negation of a verbal predicate is indicated with the suffix -mA, and marked simply as ⟨neg⟩. For nominal predicates do not get this suffix, instead the particle *değil* is used.

3.10 Tense/aspect/modality markers

A verb with a set of suffixes described above either becomes a finite verb by taking one of the tense, aspect and modality (TAM) markers followed by a person-number agreement suffix, or it can be subject to subordination and becomes nominalized.

The list of TAM suffixes, the corresponding tags and brief descriptions are given in Table 4.

Table 4: Tense/aspect/modality markers. The usage of suffix -(y)A to express conditional aspect is informal, and rather restricted. Aorist suffix is highly irregular. The choice of -Ar and -Ir depends on the stem. The -z form occurs only after negative marker, and it is not realized on the surface if it precedes first person agreement suffixes.

Tag	Suffix	Description
⟨evid⟩	-mIş	evidential past (perfective)
⟨fut⟩	-(y)AcAk	future
⟨obl⟩	-mAlI	obligative
⟨impf⟩	-mAktA	imperfective
⟨cont⟩	-(I)yor	imperfective
⟨past⟩	-DI	past (perfective)
⟨cond⟩	-sA,-(y)A	conditional
⟨opt⟩	-(y)A	optative
⟨imp⟩	-	imperative
⟨aor⟩	-Ar,-Ir,-z,-	aorist

3.11 Person and number agreement

After TAM markers a finite verb requires one of the person and number agreement markers. The surface form of the person-number agreement markers change depending on the suffixes they follow. Table 5 lists the person agreement markers and their surface form according the TAM of the verb they attach to.

3.12 Copular markers and -DIr

The copular suffixes discussed in Section 3.5 can also be attached to a verb after a TAM marker, typically forming complex tenses. These suffixes are -(y)DI, -(y)mIş and -(y)sA, tagged as ⟨cpl:past⟩, ⟨cpl:evid⟩ and ⟨cpl:cond⟩, respectively.

The conditional copula -(y)sA can co-occur with other copular markers. When there is a copular suffix, person-number agreement suffixes normally attach after the first copula. However the third person plural suffix may be after the TAM marker or second copular suffix as well.

Similar to the nominal predicates with a copula, copular suffixes may be followed the 'generalizing modality marker' -DIr tagged as ⟨dir⟩.

Table 5: Verbal person agreement markers. The first character of the person agreement tags is a number indicating the person (*1*st, *2*nd or *3*rd), and second one indicates the number (*s*ingular or *p*lural). The suffixes listed in the column marked 'TAM1' follow the TAM markers ⟨evid⟩,⟨fut⟩,⟨obl⟩,⟨impf⟩ and ⟨cont⟩ as well as the evidential copula ⟨cpl:evid⟩ and nominal predicates. The same set of suffixes also follow positive verbs with ⟨aor⟩ without a negative marker. The suffixes on the column marked 'TAM2' are used after ⟨past⟩ and ⟨cond⟩ as well as the corresponding copular markers ⟨cpl:past⟩ and ⟨cpl:cond⟩.

Tag	TAM1	TAM2	optative	imperative
⟨1s⟩	-(y)Im	-m	-(y)Im	*
⟨2s⟩	-sIn	-n	-sIn	-
⟨3s⟩	-	-	-	-sIn
⟨1p⟩	-(y)Iz	-K	-lIm	*
⟨2p⟩	-sInIz	-nIz	-sInIz	-(y)In,-(y)InIz
⟨3p⟩	-lAr	-lAr	-lAr	-sInlAr,-

3.13 The question particle

Question particle -mI, tagged as ⟨Q⟩, is normally written separately. However, it has an intimate relationship between the verb or the nominal predicate it attaches to. First, a few exceptions aside, it is attached to a tensed verb without a person agreement. In this case, the person agreement and the suffixes that may follow must be attached to the question particle. In this particular case, the verb will often be analyzed wrongly as having the agreement marker ⟨3s⟩ or ⟨3p⟩, since a predicate with null person agreement suffix may agree with third person singular or plural subjects. Second, the question particle follows the vowel harmony rules, and the underspecified vowel on -mI is realized based on the last vowel of the verb. As a result the question particle can only be analyzed (and generated) with precision only together with the word it is attached to.

If tokenized together with the predicate, TRmorph will swallow the space in between the predicate and the -mI and analyze it altogether. In this case the lowercase tag ⟨q⟩ is used. Furthermore, it is a common spelling mistakes to write the question particle together with the related word. TRmorph can be instructed to to accept this common mistake during compile time, in which case the tag will again be ⟨q⟩.

3.14 Subordination

A set of suffixes attached to an 'untensed' verb, a verb without any TAM markers, result in the phrase headed by the verb to become a subordinate clause. TRmorph follows the description in Göksel and Kerslake 2005, and makes the distinction between three different forms of subordination. First, a set of suffixes produce *verbal noun*s from a non-finite verb. The resulting words function as the head of the noun phrases, and with some limitation they can receive all nominal inflections. The second group forms *participles*, which form relative clauses. Participles can also take nominal inflections with few restrictions. The last group, *converbs*, form adverbials and they are more restricted in terms of the morphemes attached to them. The suffixes that form forms different types of subordinating suffixes overlap significantly. As a result, producing ambiguous analyses.

TRmorph uses the tag structure ⟨type:subtype⟩ for marking subordinating suffixes. The first part, type, is one of vn, part and cv for verbal nouns, participles and converbs, respectively. The second, subtype, part indicate a further distinction of the function of the suffix, a relevant linguistic abbreviation, but sometimes a version of the surface form of the suffix. The tags used for all three types of subordinating suffixes are listed in Table 6.

Since verbal nouns, participles and converbs derive nominal, adjectival and adverbial phrases, respectively, POS tags, ⟨N⟩, ⟨Adj⟩ and ⟨Adv⟩, follow these tags.

Some of the suffixes have multiple functions and may derive more than one type of subordinate clauses. Furthermore, TRmorph will produce some spurious ambiguity because of the fact that any adjective, hence a word suffixed with an participle, is allowed to become a noun with a zero derivation.

The list in Table 6 follows Göksel and Kerslake 2005. The main exception is the suffixes listed by Göksel and Kerslake 2005 as converbial suffixes that require a postposition. Since the postposition in these cases will signal the adverbial function of postpositional phrase, TRmorph

Table 6: Subordinating suffixes and tags used for subordinating suffixes.

Tag	Suffix
⟨vn:inf⟩	-mA
⟨vn:inf⟩	-mAK
⟨vn:yis⟩	-(y)Iş
⟨vn:past⟩	-DIk
⟨vn:fut⟩	-(y)AcAk
⟨vn:res⟩	-(y)An
⟨part:past⟩	-DIk
⟨part:fut⟩	-(y)AcAk
⟨part:pres⟩	-(y)An
⟨cv:ip⟩	-(y)Ip
⟨cv:meksizin⟩	-mAksIzIn
⟨cv:ince⟩	-(y)IncA
⟨cv:erek⟩	-(y)ArAk
⟨cv:eli⟩	-(y)AlI
⟨cv:dikce⟩	-DIkCA
⟨cv:esiye⟩	-(y)AsIyA
⟨cv:den⟩	-dAn
⟨cv:den⟩	-zdAn
⟨cv:cesine⟩	-CAsInA
⟨cv:ya⟩	-(y)A
⟨cv:ken⟩	-(y)ken

does not mark the complement of the postposition as a converb.

Most of these suffixes attach to an untensed verb. Except, the suffix -(y)ken which behaves much like the copular suffixes discussed above. Furthermore, the -(y)A in its subordinating function is typically used together with reduplication, e.g., *koşa koşa* 'run-(y)A run-(y)A = hurriedly', but also occurs in words like *diye*, where it does not need reduplication.[9]

Besides the subordinating suffixes (participles) discussed above, some of the TAM markers, namely ⟨aor⟩, ⟨evid⟩, ⟨fut⟩, ⟨makta⟩ and less commonly ⟨cont⟩, derive adjectivals resembling relative clauses. TRmorph handles this by analyzing any verb with one of these TAM markers without further suffixes (e.g., agreement markers) as an adjective. For example, the word *görülmüş* in *görülmüş mektup* 'see-PASV-EVID letter = the letter that was seen' is analyzed as 'gör⟨V⟩⟨pass⟩⟨V⟩⟨evid⟩⟨Adj⟩'.

[9] One may also analyze *diye* as a postposition, as it's use as subordinator is semantically unlike the others uses of -(y)A.

3.15 Productive derivational morphemes

Almost all the tags and relevant morphological process above are described as part of inflectional morphology in most grammar books. The suffixes described here are the ones that are traditionally considered derivational suffixes. Some of these suffixes, for example -lI and -sIz discussed earlier, may attach to word forms that are already inflected by other suffixes. Others normally attach only to the stem and produce another stem.

Of these suffixes, the noun–verb derivation suffix -lA causes a large number of ambiguous analyses since it is part of many other suffixes. These, for example, include the plural suffix -lAr whose remainder -r also matches a verbal suffix (aorist). Hence, including -lA in the analysis causes an increase in the analyses of any plural noun. Currently, TRmorph analyzes -lA only after onomatopoeia. The rest of the verbs derived from nouns using this suffix are lexically specified.

TRmorph does not limit the number of derivational suffixes that can be stringed one after another other, even though multiple derivations of this sort is a lot more restricted.

Besides the sources of possible erroneous over-analyses listed above, the derivational morphology specification in TRmorph over-generates in some cases. In particular, any form of the diminutive suffix is allowed to attach to any noun, although most nouns are used only one of the diminutive suffixes. The ambiguity and overgeneration are discussed in Section 5.

4 The lexicon

TODO.

5 Ambiguity and overgeneration

This section discusses the ambiguous analyses in TRmorph, and also touches upon a related but different prob-

Table 7: Derivational morphemes analyzed by TRmorph. The column 'Derivation' lists the POS changes using a two letter symbols. The first letter is the original POS, and the second one is the POS after the suffixation. Here, N means noun, J means adjective, A means adverb, V means verb and O is onomatopoeia.

Tag	Suffix	Derivation
⟨li⟩	-lI	NA NJ
⟨siz⟩	-sIz	NA NJ
⟨lik⟩	-lIk	NN JN AN
⟨dim⟩	-CIk	NN
	-cAk	
	-(I)cAk	
	-cAğIz	
⟨ci⟩	-CI	NN NJ
⟨ca⟩	-CA	NA AA JJ MJ
⟨yici⟩	-(y)IcI	VJ
⟨cil⟩	-CIl	NJ
⟨gil⟩	-gil	NN
⟨lan⟩	-lAn	JV
⟨las⟩	-lAş	NV JV
⟨yis⟩	-yIş	VN
⟨esi⟩	-(y)AsI	VJ
⟨sal⟩	-sAl	VJ
⟨la⟩	-lA	NV OV

lem, overgeneration.

The morphological analysis of Turkish text is an inherently ambiguous process. However, the design choices made in a morphological analyzer affects the number of ambiguous analyses produced per word. TRmorph, by design, does not try to reduce the number of ambiguous analyses. In general, TRmorph produces more ambiguous analyses than the others (mainly based on Oflazer 1994) reported in the literature.

The following is a list of cases where one finds ambiguous morphological analyses in TRmorph. Some of these cases are not specific to TRmorph, and for example, noted by Oflazer and Tür 1997 as well. This list may be useful for the users who may wish to disambiguate the output of the analysis using rule-based methods, or it may also be useful in the process of designing statistical disambiguators.

1. Ambiguous root forms, for example *yüz* can be analyzed as:
   (a) yüz⟨N⟩ 'face'
   (b) yüz⟨Num⟩ 'hundred'
   (c) yüz⟨V⟩⟨imp⟩⟨2s⟩ 'swim'
2. A root form is the same as a shorter root and one or more suffixes, for example *buna* can be analyzed as
   (a) bu⟨Prn:dem⟩⟨dat⟩ 'this-DAT'
   (b) buna⟨V⟩⟨imp⟩⟨2s⟩ 'become senile-IMP'
   (c) bun⟨N⟩⟨dat⟩ 'trouble-DAT'

   Note that the root 'bun' is a very rare/regional word, and the imperative verb reading is also very unlikely. However the best option for the analyzer is to produce all these analyses, and let the later stages disambiguate between them.
3. The surface form of a suffix is a combination of two other suffixes. For example, the word *evleri* can be
   (a) ev-leri 'ev⟨N⟩⟨p3p⟩ = their house'
   (b) ev-ler-i 'ev⟨N⟩⟨pl⟩⟨acc⟩ = houses-ACC'

   Furthermore, the same word can also be analyzed as
   (a) 'ev⟨N⟩⟨pl⟩⟨p3s⟩'
   (b) 'ev⟨N⟩⟨pl⟩⟨p3p⟩'
   (c) 'ev⟨N⟩⟨ncomp⟩⟨p3p⟩'
   (d) 'ev⟨N⟩⟨ncomp⟩⟨pl⟩'
   (e) 'ev⟨N⟩⟨ncomp⟩⟨pl⟩⟨p3p⟩'
   (f) 'ev⟨N⟩⟨ncomp⟩⟨pl⟩⟨p3s⟩'
   (g) 'ev⟨N⟩⟨ncomp⟩⟨pl⟩⟨p3p⟩'

   The reason for these analyses has to do with the sources of ambiguity explained in items 6 and 7.
4. An analysis with multiple morphemes is also a (derived) lexicalized form. For example the word *konuşma* can be analyzed as
   (a) konuşma⟨N⟩ 'speech'

(b) konuş⟨V⟩⟨vn:inf⟩⟨N⟩ infinitive 'to speak', e.g., as in *konuşmamızı isemiyorlar* 'The do not want us *to speak*'

(c) konuş⟨V⟩⟨neg⟩⟨imp⟩⟨2s⟩ 'speak-NEG-IMP = don't talk'

5. different affixes surfacing the same way, *evin* can be

   (a) ev-(n)In 'ev⟨N⟩⟨gen⟩ =of the house'

   (b) ev-(I)n 'ev⟨N⟩⟨p2s⟩ =your house'

6. The same surface suffix has multiple functions. For example, the word *doktorlar* can be,

   (a) doktor⟨N⟩⟨pl⟩ 'doctors'

   (b) doktor⟨N⟩⟨0⟩⟨V⟩⟨cpl:pres⟩⟨3p⟩ 'they are doctors'

7. Some suffixes are not realized on the surface in the neighborhood of some other suffixes. These are generally, but not always, the suffixes having the same or similar surface forms. For example, *evleri* (the example in item 3) may be analyzed as

   (a) ev⟨N⟩⟨p3p⟩ as in *Annem ve babamın evleri Istanbul'da* 'My parents' house is in Istanbul'

   (b) ev⟨N⟩⟨pl⟩⟨p3p⟩ as in *Annem ve babamın bütün evleri deniz manzaralı* 'All houses of my parents have a see view'.

   since in case of ⟨pl⟩ (-lAr) and ⟨p3p⟩ (-lArI) are combined, the plural suffix -lAr does not realized on the surface.[10]

   This particular source causes an extremely large number of ambiguous analyses because the multi functional suffix -(s)I is omitted in case it precedes (or follows) another -(s)I, but also a -lArI, -lI, -lIk, -sIz, -CI or -CIk. Since some of these suffixes may follow each other, and -(s)I itself has multiple functions, a word like *bağım-sız-lık-çı-lığ-ı-nı* causes a combinatorial expansion of ambiguous analyses because of the fact that at every suffix boundary marked with a dash in the example there may be a -(s)I suffix being deleted. This is further amplified by the fact that -(s)I may express ⟨ncomp⟩ or ⟨p3s⟩ and any of the resulting words may also have a null suffix expressing third person singular or plural agreement on a nominal predicate.[11] Most of these analyses will be semantically not plausible. However, there is no clear way of ruling them out at the analysis stage. The following illustrates the problem with a more tangible example, using the word *arabasız* which can be analyzed as one of the following (and more).

   (c) araba⟨N⟩⟨siz⟩⟨Adj⟩ 'without a car'

   (d) araba⟨N⟩⟨p3s⟩⟨siz⟩⟨Adv⟩ 'without a his/her car'

   (e) araba⟨N⟩⟨ncomp⟩⟨siz⟩⟨Adv⟩, e.g., in *at arabasız* 'without a horse carriage'

   (f) araba⟨N⟩⟨ncomp⟩⟨p3s⟩⟨siz⟩⟨Adv⟩, e.g., in *at arabasız* 'without his/her horse carriage'

Besides the ambiguity described above, overgeneration is another problem that one faces when the FST is used for generating surface forms. Unlike analysis, generation is almost always deterministic in Turkish. Nevertheless, there are a few cases where TRmorph produces multiple surface strings for a single analysis string. The following provides a (likely incomplete) list of cases where TRmorph is expected to overgenerate, i.e., either produce multiple (correct) surface strings for the same input, or produce incorrect surface strings in generation mode.

1. One of the clear cases where overgeneration occurs is the diminutive, ⟨dim⟩. The diminutive suffix in Turkish is one of -CIk, -cAk, -(I)cAk, -cAğIz. TRmorph allows attaching any of these suffixes to any noun. This is unlikely to cause problems during the analysis. However, it will certainly produce incorrect surface forms.

2. The ⟨p3s⟩ suffix -(s)I may also be used for marking third person plural possessive (⟨p3p⟩). For example *ev-i* in *Ali ve Ayşe'nin evi* 'The house of Ali and Ayşe' should be tagged as ⟨p3p⟩. On the other hand, the suffix -lArI is also used to express ⟨p3p⟩. As a result any analysis string with the symbol ⟨p3p⟩ will generate both surface options.

[10] One can also explain this as ⟨p3p⟩ being realized as -I in this particular context.

[11] Most straightforward reading of the word is dative form of the noun phrase can roughly be translated as 'his/her state of being a supporter of independence'. With this root, The total number of analyses is 25560.

3. A similar case of overgeneration is with the null agreement suffix which should generally be tagged as ⟨3s⟩. However, such a predicate may also agree with a ⟨3p⟩ subject. Consequently, a null-agreement suffix on a predicate is tagged as both ⟨3s⟩ and ⟨3p⟩. Since ⟨3p⟩ can also be expressed with the suffix -lAr, a analysis string with ⟨3p⟩ also generates multiple surface forms.

4. Another known case of overgeneration is related to the relaxed analysis of alternative spellings or common misspellings. In the simplest case, every word will be generated once capitalized and once all lowercase. If 'all capitals' option is enabled, another surface form which is in all capital letters will be produced.

5. Similarly, if the analyzer is instructed to accept the proper noun suffixes without an apostrophe, in the generation mode the surface form with and without apostrophe will be included. As a result, some of the options may need to be tuned if the FST is to be used for generation.

6. Some symbols, like apostrophe have multiple representations in Unicode definition. As a result any word that require an apostrophe will result in surface form for each alternative symbol.

7. After a small set of borrowings like *cami* 'mosque', the 's' in the suffix -(s)I is deleted according to official rules. However, this seems to be out of fashion in current use, and use of 's' (even in text) is more common that its deletion. Since TRmorph accepts both surface strings, this will cause generating multiple strings.

   There are also a few other cases where some (sizable number of) speakers diverge from the canonical forms. An example is the redundant use of genitive suffix after a pronoun, before the suffix -(y)lA, e.g., the surface form of 'sen⟨Prn:pers:1s⟩⟨ins⟩' should be *sen-in-le* where the suffix -in is redundant. Some speakers tend not to use -in int such constructions. TRmorph accepts both use, hence the generation will be ambiguous.

8. Some borrowed words include a few vowels with circumflex, namely *â*, *û* and *î*. Except for a few words where use of circumflex help disambiguation between different words, these vowels have been replaced by their non-circumflexed version in modern use. TRmorph allows this replacement even if the lexical form of the word should include a circumflex.[12] This also results in overgeneration, since any analysis string with a circumflexed vowel will have a surface form with and without circumflex.

6 Other tools

6.1 Stemming and lemmatization

In morphologically complex languages like Turkish, proper stemming requires analyzing the given word and stripping off the analysis symbols such that only the stem remains.

Although one can do this easily by filtering analyzer output, TRmorph includes a simple wrapper automaton for convenience. The automaton is defined in the file `stemmer.fst`. You need to type `make stemmer` to produce the binary `stem.fst`. This binary file can be used the way analyzer is used. Given a surface word, this automaton will produce the lexical form as the analysis string.

Optionally, one can keep the first tag, which is the syntactic category of the stem. Note that stemmer takes the lexical form as the 'stem', even if the lexical form has derivational suffixes immediately following the root form. Another compile time option related to stemmer causes the verbs to be suffixed with correct form of infinitive marker -mAk. This form of the verbs are what the dictionaries use as head words. Both options can be set in the file `options.h`.

Note that ambiguity is less of a problem for the stemmer. However, in examples like *buna* discussed on page 11, there will be multiple stem forms produced (*bu*, *bun* and *buna* in this case).

[12] One can also allow circumflexed vowels to be used for their non-circumflexed counterparts in the lexicon. This is useful if one needs to analyze somewhat older text. Enabling this option will also cause overgeneration.

6.2 Unknown word guesser

TRmorph includes a rudimentary guesser for guessing unknown words. To produce the automaton for this function, you should type `make guesser`, which would produce the file `guess.fst`. The usage of the automaton is again similar to the others. The surface strings of the FST is the (unknown) words, while analysis level is either the full analysis strings with possibly unknown root words that may lead to the surface form, or only the root word and its part of speech tag.

The guesser uses the same machinery as the analyzer, except the lexicon is replaced with a FSA that accepts a somewhat restricted set of strings as potential words. Since unknown words will likely include affixes, one may have a better chance of determining the root form of the word, and in most cases the class of the root word.

Depending on its application, the guesser be restricted further according to features of the words that can be coded into a finite state lexicon. For example, one may check whether the words fit into the syllable structure of the language, but this may miss the words of foreign origin, which are likely candidates for being unknown words. Currently only general restriction the guesser include the minimum and maximum root-word length that can be set in the file `options.h`.

The guesser may also be adjusted to return full analysis string(s) or only the root form followed by the POS tag. Again, these options can be set in `options.h`. Other customizations can be achieved by adjusting the file `guesser.lexc`.

The guesser is a standalone FST, to use it in combination with the analyzer, two automata can be combined with *priority union* such that guesser is only invoked if the analyzer fails. This can be achieved either as a simple wrapper xfst file, or if you are using foma's `flookup` utility specifying both FST files on the command line like `flookup -a trmorph.fst guesser.fst`.

6.3 Morphological segmentation

Morphological segmentation is the task of finding morpheme boundaries on the surface strings. TRmorph distribution includes an automaton description for segmenting the words into their morphemes. To build the segmenter you need to type `make segmenter` and the resulting binary will be called `segment.fst`.

TRmorph marks the root and morpheme boundaries on the surface string to aid morpho-phonological rules. These boundaries are deleted from the surface string in the normal analyzer FST. The segmentation FST relies on this and the following trick for segmenting a given word to its surface morphemes: The given input string is first analyzed with the regular analyzer FST. Then the analysis strings are passed to a slightly modified FST in generation mode, which does not delete the boundary markers from the surface string.

It should be noted that the surface morpheme boundaries are not always determined uniquely. It is especially difficult to decide whether some buffer vowels or consonants belong to the morpheme preceding or following them. TRmorph consistently attaches these buffer letters to the morpheme that follow the boundary.

Because of the way it is implemented currently, the segmenter output needs to be post processed to obtain the desired result. The segmenter will produce multiple identical segmented strings, and there will also be some incorrect segmentations due to overgeneration discussed in Section 5. The output should be post-processed to remove multiple identical segmentations. The incorrect segmentations due to overgeneration can be eliminated by comparing the segmented string with the original one. An example post processing script is provided as `scripts/segment-filter.py`.

6.4 Hyphenation and syllabification

Hyphens in Turkish are inserted at the syllable boundaries. Because of the regular syllable structure and transparency of the orthography, this process does not require any dictionary lookup, or morphological analysis. Since the hyphenation problem is easy to solve with a FST, a stand alone FST defined in xfst language included in the TRmorph distribution.

To build the hyphenation FST you need to type `make hyphenate` and the resulting binary will be called `hyphenate.fst`.

The surface string of the FST is Turkish words (or strings resembling words) and analysis string is the words where a hyphen '-' is inserted between the syllables, or at the points where one can insert a hyphen.

References

Beesley, Kenneth R and Lauri Karttunen (2003). "Finite-state morphology: Xerox tools and techniques". In: *CSLI, Stanford*.

Çöltekin, Çağrı (2010). "A Freely Available Morphological Analyzer for Turkish". In: *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC2010)*. Valetta, Malta.

Göksel, Aslı and Celia Kerslake (2005). *Turkish: A Comprehensive Grammar*. London: Routledge.

Hulden, Mans (2009). "Foma: a finite-state compiler and library". In: *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics: Demonstrations Session*. Association for Computational Linguistics, pp. 29–32.

Lindén, Krister, Miikka Silfverberg, and Tommi Pirinen (2009). "HFST Tools for Morphology–An Efficient Open-Source Package for Construction of Morphological Analyzers". In: *State of the Art in Computational Morphology*. Ed. by Cerstin Mahlow and Michael Piotrowski. Communications in Computer andInformationScience. Springer, pp. 28–47. ISBN: 978-3-642-04130-3.

Oflazer, Kemal (1994). "Two-level description of Turkish morphology". In: *Literary and Linguistic Computing* 9 (2).

Oflazer, Kemal, Dilek Hakkani-Tür, and Gökhan Tür (1999). "Design for a Turkish Treebank". In: *Proceedings of the Workshop on Linguistically Interpreted Corpora, at EACL'99*. Bergen, Norway.

Oflazer, Kemal, Bilge Say, Dilek Zeynep Hakkani-Tür, and Gökhan Tür (2003). "Building a Turkish treebank". In: Text, Speech and Language Technology. Springer, pp. 261–277. ISBN: 9781402013348.

Oflazer, Kemal and Gökhan Tür (1997). "Morphological Disambiguation by Voting Constraints". In: *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pp. 222–229.

Schmid, Helmut (2005). "A programming language for finite state transducers". In: *Proceedings of the 5th International Workshop on Finite State Methods in Natural Language Processing (FSMNLP 2005)*. Helsinki, pp. 308–309.

Index