BERKE ÖZENÇ

M.S. Thesis

2017

MORPHOLOGICAL ANALYSER FOR TURKISH

BERKE ÖZENÇ

IŞIK UNIVERSITY
2017

MORPHOLOGICAL ANALYSER FOR TURKISH

BERKE ÖZENÇ
B.S., Computer Engineering, IŞIK UNIVERSITY, 2017

Submitted to the Graduate School of Science and Engineering
in
Computer Engineering

IŞIK UNIVERSITY
2017

IŞIK UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

MORPHOLOGICAL ANALYSER FOR TURKISH

BERKE ÖZENÇ

APPROVED BY:

Prof. Dr. Ercan Solak (Thesis Supervisor)	Işık University	_____
Assoc. Prof. Olcay T. Yıldız	Işık University	_____
Assoc. Prof. M. Oğuzhan Külekçi	Istanbul Technical University	_____

APPROVAL DATE: ..../..../....

MORPHOLOGICAL ANALYSER FOR TURKISH

Abstract

Natural Language Processing is one one the fields of work in computer science and specializes in text summarization, machine translation and many various topics. Morphology is one of the Natural Language Processing features which analyses the words with its suffixes. A words meaning can change according to the suffix that it takes. Turkish is an agglutinative language with rich morphological structure and set of suffixes. This features of Turkish result in complex morphology structure.

In this study, we present an analyser for Modern Anatolian Turkish which has high coverage on suffixes and morphological rules of Turkish. Two-Level transformation method which is convenient to design morphology of a language, consists our base of approach. We used HFST which is a Finite State Transducer implementation, as our implementation technique. The analyser covers all morphological and phonetic rules that exist in Turkish and contains a lexicon which consist of today's Turkish words. The analyser is publicly available and can be used on http://ddil.isikun.edu.tr/mortur.

Keywords: Turkish, Morphology, Analyser, Two-Level Approach, FST, Finite State Transducer, HFST, Natural Language Processing, NLP

TÜRKÇE İÇİN MORFOLOJİK ANALİZÖR

Özet

Doğal Dil İşleme, bilsayar bilimindeki çalışma alanlarından biridir ve özetleme, makine çevirisi gibi bir çok alanda özelleşmektedir. Morfoloji, Doğal Dil İşlemede kullanılan özelliklerden biridir ve bir kelimeyi ekleriyle birlikte analiz eder. Bir kelimenin anlamı aldığı eklere göre değişebilir. Türkçe, zengin morfolojik yapıları ve zengin ek kümesi olan eklemeli bir dildir. Turkçenin bu özelliği, kompleks morfolojik yapıları ortaya çıkartır.

Bu çalışmada Modern Anadolu Türkçesi için bir analizör sunuyoruz. Bir dile ait morfolojik yapıyı tasarlamak için uygun olan Çift katmanlı dönüşüm metodu, analizörü hazırlarken kullandığımız yaklaşımın temelini oluşturmaktadır. Analizörün kodlanması için, bir sonlu durum dönüştürücüsü uygulaması olan HFST kullandık. Analizörümüz, Türkçe'de var olan tüm morfolojik ve fonetik kuralları kapsamaktadır ve güncel Türkçe kelimelerinden oluşan bir sözlük bulundurmaktadır. Analizörümüz, halka açık olarak, http://ddil.isikun.edu.tr/mortur adresinden kullanılabilir.

Anahtar kelimeler: Türkçe, Morfoloji, Analizör, Çift Katmanlı Yaklaşım, FST, Sonlu Durum Dönüştürücüsü, HFST, Doğal Dil İşleme, DDİ, NLP

Table of Contents

Abstract ii

Özet iii

List of Tables vi

List of Figures vii

1 Introduction 1

2 Literature Survey 2

3 Approach 4
- 3.1 Morphotactics . . . 4
- 3.2 Phonology . . . 5

4 Turkish 6
- 4.1 Lexicon . . . 6
- 4.2 Nominal Inflection . . . 7
- 4.3 Verbal Inflection . . . 8
  - 4.3.1 Voice . . . 9
  - 4.3.2 Ability Polarity Probability (APP) . . . 11
  - 4.3.3 Tense Aspect Modality Person . . . 13
- 4.4 Nominal Predicate . . . 19
- 4.5 Other Classes . . . 21
- 4.6 Derivation . . . 22
- 4.7 Phonology . . . 26

5 Implementation 28
- 5.1 HFST Structure . . . 28
- 5.2 HFST Syntax . . . 30
- 5.3 MorTur Structure . . . 33

6 MorAz 34

7 Results and Discussion 36

8 Conclusion **40**

Reference **41**

List of Tables

4.1 Possessor morphemes for Figure 4.1. Empty cells are undefined. . 9
4.2 Case suffixes for Figure 4.1. 9
4.3 Tense categories and their semantics. 14
4.4 Copula categories and their semantics. 14
4.5 Person paradigms for all verb inflection. 15
4.6 Person paradigms for all verb inflection. 15
4.7 The most productive derivational morpheme sets d_1, d_2, d_3 and d_4 used in Figure 4.15. 24

6.1 Person paradigms for all verb inflection in AT. 34

List of Figures

4.1 Slot-based nominal inflection 8
4.2 Voice paradigm 10
4.3 Verb classes representing different behavior for Causative and Passive suffixes for different verb roots. 11
4.4 Ability-polarity-probability sub-paradigm. Note the ambiguity caused by the identical surface forms of the paths APP-POL_3-PROB-$TAMP_1$ and APP-ABIL-POL_1-$TAMP_1$. 12
4.5 Group 1, behaviour of imperative 15
4.6 Group 2, behaviour of Optative 16
4.7 Group 3, behaviour of Conditional 17
4.8 Group 4, behaviour of Past 18
4.9 Group 5, behaviour of Past 18
4.10 Group 6 19
4.11 Group 7 20
4.12 Group 8 20
4.13 Nominal Predicate Pt1 21
4.14 Nominal Predicate Pt2 22
4.15 The derivation FST. 23
4.16 Whole FST of the Turkish Morphology 25

6.1 The nominal predicate for AT. 35

Chapter 1

Introduction

Morphology is an essential part of the Natural Language Processing (NLP). Almost all works in NLP starts with decomposition, analysis, of the word. By this, possible meaning of the word can be acquired. This requires a morphological analyser.

Turkish is an agglutinative language with rich morphological structures. It is most spoken one among all Turkic languages. For the clarity, Turkish in this work is termed Modern Anatolian Turkish. Its alphabet is based on Latin alphabet with addition of "ı", "ü", "ö", "ç", "ğ", "ş" and exclusion of "x", "q" and "w". Like all Turkic languages, Turkish has vowel harmony, consonant drops and changes as part of its phonology.

In this work, we provided a new description of Modern Anatolian Turkish morphology and implemented its morphological analyser (MorTur). In our implementation, we used Helsinki FST (HFST) [1]. We published the analyser as a website on http://ddil.isikun.edu.tr/mortur. It is free to use and its source is publicly available. The source code is accessible though the web page.

Chapter 2

Literature Survey

Finite State Transducer (FST) is a convenient tool for implementing morphological structure of an agglutinative language with a two-level transformation approach. Two-level transformation approach to morphology is explained in Chapter 3. There are several implementations of FST and all implementations are based on Xerox Finite State tool (XFST) [2]. XFST is a tool that provides finite state operations along with a regular expression compiler. XFST includes two types of operations as lookup (analyse) and generation. Helsinki Finite State Transducer is the FST implementation that we used to implement MorTur. Like other implementations, it is based on XFST so it inherits all features of XFST including analyse and generation modes. Details about HFST are given in Sections 5.1 and 5.2.

FST is popular among morphology researches because of its convenience for the structure. There are several works that use FST with two-level transformation approach for several languages. For English, there is [3] with a large lexicon. Large lexicon provides a large coverage for the analyser. Also, there is a tool to manage the lexion for further modifications. PC-KIMMO is used for the implementation. PC-KIMMO is a system that is specially developped for linguistics. The program is designed for generate or analyse the words. It requires morphological rules and a lexicon, set of root. For Japanese, there is [4]. Similarly, KIMMO is used for the implementation with provided Japanese morphological rules. Also, there is a study that includes Finnish, English, Japanese, Romanian, French, Swedish, Old Church Slavonic, Greek, Lappish, Arabic, Icelandic Languages [5]. Unlike a single language analyser, a language independent model is presented in their work.

Unlike other languages, there are not many morphological analyzers for other Turkic languages. For Turkmen, there is [6]. In this work, XFST is used with

two-level morphology approach. For Kazakh, there is [7]. The analyser implemented by using FOMA which is another implementtaıon of FST, wtih the two-level transformation approach. In addition to morphological analyser this work contains a morphological disambiguator. Since there might me more than one morphological analyses for a word, morphological disambiguator decides which analyses of the word is valid according to context. For Uighur, there is, [8]. Rather than a complete analyser, their work focuses on nouns. XFST is used for implementation.

Although, Modern Anatolian Turkish and Azerbaijani Turkish (AT) are close to each other, there are no Morphological analysers for AT as far as we know. There are significant differences between Turkish and AT. This prevents an easy adaptation of available Turkish analyser to AT. So, we implemented the very first AT morphological analyser (MorAz) with the same approach and implementation of FST. Details about MorAz are givenwa in Chapter 6.

There are several morphological analyzers for Turkish, [9], [10], [11]. [9] is implemented by Kemal Oflazer. It is the very first morphological analyzer that is done for Turkish. Two-level morphology is used as approach and PC-KIMMO environment is used for implementation. [10] is implemented by Çağrı Çöltekin. Two-level morphological approach is used and Stuttgart Finite State Transducer (SFST) is used for implementation. Like HFST, SFST is a FST implementation that is based on XFST. [11] is implemented by Muhammet Şahin, Umut Sulubacak and Gülşen Eryiğit. In their work, Flag Diaractics is used for implementation. Flag Diaractics is an extension of XFST. They used Flag Diaractics to handle morphological and phonetic exceptions. These analysers have some linguistically problematic or neglected parts. This is why we implemented a new and complete analyser in the first place. Differences between MorTur and other analysers are explained in Chapter 7.

Chapter 3

Approach

We used a Two-level representation to represent the morphology of Turkish. First of these two levels formally describes the morphotactics using a finite state transducer (FST). Second level consists of sets of rules which using modify the output of first level FST according to phonological changes. Every set of rule is related to a different phonological phenomenon. To implement the two-level structure, we used HFST tool. HFST is a FST implementation developed by Helsinki University. HFST consist of lexc and twol files, which correspond respectively to each level of two-level representation.

3.1 Morphotactics

A FST is a finite state machine whose transitions have a pair of input/output strings. A FST consists of state and these states connected to each other with transitions. Transitions modify the input string that is from origin state of the transition with a pre-determined string to produce the output string. By its structure, FST is convenient structure to represent morphotactics of an agglutinative language. Input and output string of a transition have special meanings when FST is used to represent a language like Turkish. In the FST representation of Turkish, every transition corresponds to a suffix. In these transitions, while output string is abstract morpheme which is symbolic representations of the suffix, the input string is the label that corresponds to the abstract meaning of the suffix.

(1) kalem<NOM><Num:Pl><Poss:1s>

(2) kalem-lAr-(I)m

(3) kalemlerim

In these examples, “kalemlerim” (my pens) is a string that is generated by FST. On (1), morphological class of the word and labels of suffixes are shown respectively, <NOM> shows that the root “kalem” (pen) belongs to the Nominal class, <Num:Pl> is abstract morpheme of “-lAr” and <Poss:1s> is abstract morphem of “-(I)m”. abstract morphemes consist of two parts seperated with “:”. First part denotes the key and second part denotes the value. On (2), suffixes are shown seperately, -lAr and -(I)m, this is also output of the first level. In this representation we used “-” as seperator for suffixes. This representation of suffixes called archmorphemes. In this example, “A” and “I” are archiphonemes and brackets around the “I” denotes that “I” is optional. (3) called surface form. It will be explained on following sub-section.

3.2 Phonology

In the second level, which implements phonological rules, sets of rules are applied to the output of the first level. As a result, surface form of the word is generated. These rules clean suffix seperator and, if necessarry, optional characters. Also, change archiphonemes into their correct versions according to big vowel harnomy of Turkish. The following example (3) is surface form of (2).

Among the three strings (1), (2) and (3), only (1) and (3) are considered as pure input and result. This has two work mode, generation and analysis. In generation mode, (1) is given as input and two-level structure produces the surface form as a result. In analysis mode, (3) surface form is given as input and (1) the analysis is produced.

Chapter 4

Turkish

Turkish is an agglutinative language. It has rich morphological structure for nominal and verbal words. Every word class has it's own inflectional class. Moreover, it is also possible to derive new words, with same or different word classes, from the root or stem. These derived words may undergo further inflection. After the inflection, they may derive again. It depends on the suffix that causes the derivation. All derivational suffixes and their functionalities will be explained in 4.6 section. Such rich inflection and derivation result in a single surface form having many distinct analyses.

4.1 Lexicon

In Turkish there are 4 main word classes; Verbs, Adverbs, Nouns and Adjectives. In all Turkic languages, distinction of Nouns and Adjectives is rather fuzzy. In a noun phrase having an adjective that modifies the noun, the noun can be dropped. When it happens, dropped noun is indicated by the context. For example,

(1) mavi bardak kırık
blue glass is broken
mavi kırık
blue is broken

When an adjective modifies a noun, it become the subject so that they can take place of the noun and act as noun in the absence of noun. This means an adjective can be inflected and derived as a noun. In addition to that, nouns also can be

used as adjectives. So we merged Nouns and Adjectives under a new category Nominal.

Normally, POS tags are determined by the context and morphology is independent from context. Thus It is not correct to represent a root with its POS tag in morphological analysis. So, we used morphological categories instead of POS tags. Nominal is one the categories and nouns, pronouns and adjectives are in this category. Other than Nominal, there are Adverb and Verb categories. Some nouns can be used as adverbs but there are also words that are used only as adverbs thus adverb category is necessary.

4.2 Nominal Inflection

Nominal inflection of Turkish has fixed order of suffixes. The order is

(2) Nominal root/stem-Number suffix-Possession suffix-Case suffix

When a suffix has obligatory part in the inflection, it is called as slot. In this structure, all suffixes are slots. So this can be called slot based nominal inflection.

Number suffix gives the word meaning of singularity or plurality. The slot can have Singular (Sg) or Plural (Pl) values. Surface form of this suffixes are in Figure 4.1. Possessor suffix gives the word meaning of being possessed. The slot has 7 values, one for each person type and also for the case no possessor. Surface form of possessor suffixes are in Table 4.1. Case suffix indicates the state of the word. The slot can have 7 values of Nominal (Nom), Accusative (Acc), Dative (Dat), Locative (Loc), Ablative (Abl), Genetive (Gen) and Instrumental (Ins). Surface form of Case suffixes are in table 4.2. The whole FST of nominal inflection is on Figure 4.1.

In slot based nominal inflection, all suffixes must be present. So Case states are the only final states in the FST. As an example, anaylsis of the word “araba” (car) is

(3) araba<NOM><Num:Sg><Poss:None><Case:Nom>

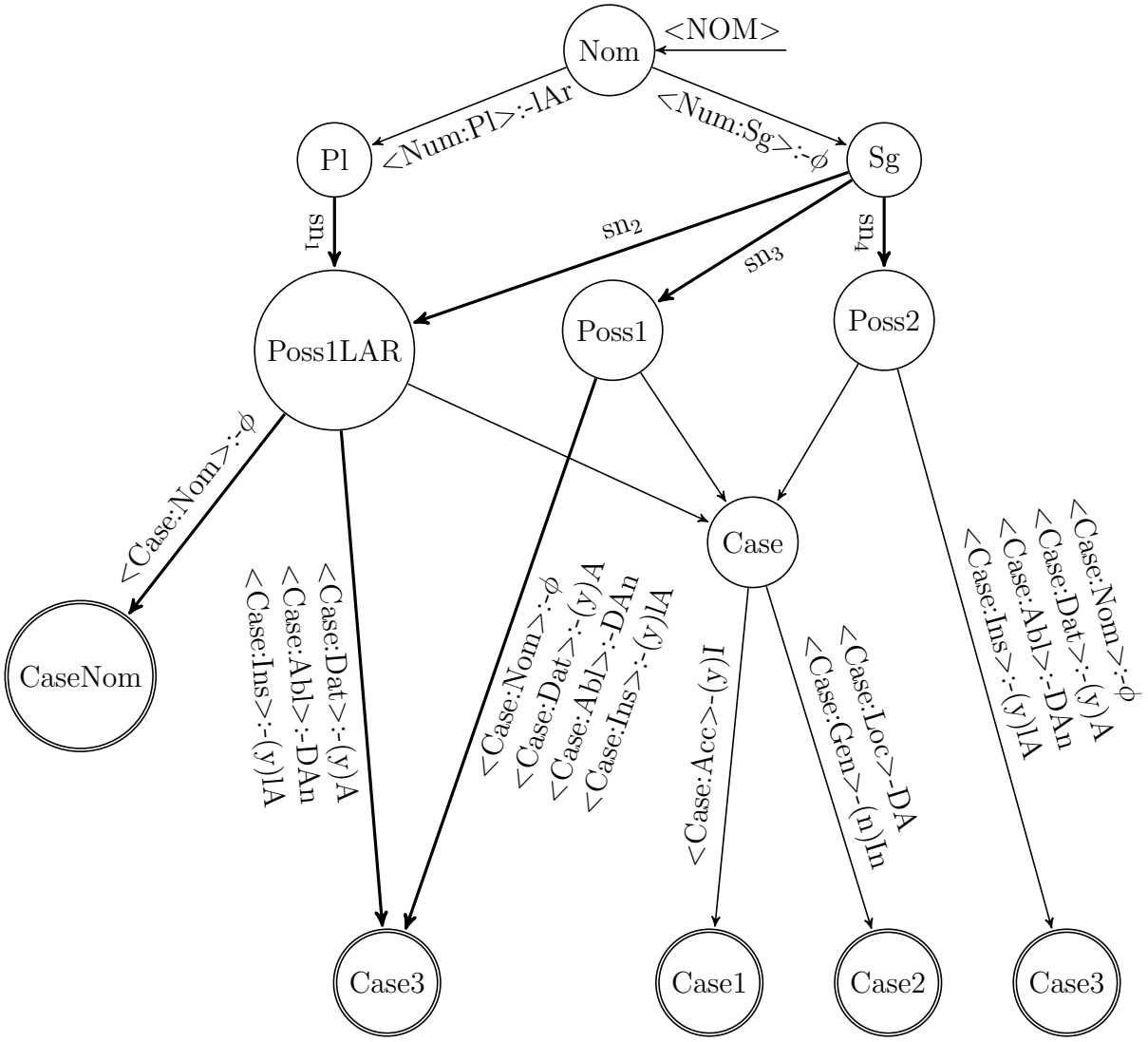

Figure 4.1: Slot-based nominal inflection

In Figure 4.1, Case state is just a dummy state to group some transactions. In some cases, nominal inflection can be interrupted by derivation. Details of derivation is in section 4.6. This Case state also helps to handle the derivation that occurs right after the possession suffix. sn's on the figure are custom suffix groups of possessor suffixes. We used sn to not to clutter the diagram. Suffixes of sn groups can be found on Table 4.1

4.3 Verbal Inflection

Verb inflection has more suffixes than nominal inflection. But unlike Nominal inflection, Verb inflection doesn't have slot based structure. The general structure

Table 4.1: Possessor morphemes for Figure 4.1. Empty cells are undefined.

Abstract morpheme	**First level output**			
	sn$_1$	**sn**$_2$	**sn**$_3$	**sn**$_4$
<Poss:No>	-ϕ		-ϕ	-ϕ
<Poss:1s>	-Im		-(I)m	-(I)m
<Poss:2s>	-In		-(I)n	-(I)n
<Poss:3s>	-I(n)		-(s)I(n)	
<Poss:1p>	-ImIz		-(I)mIz	
<Poss:2p>	-InIz		-(I)nIz	
<Poss:3p>	-I(n)	-lArI		

Table 4.2: Case suffixes for Figure 4.1.

Case	**Abstract morpheme**	**First level output**
Nominal	<Case:Nom>	-ϕ
Dative	<Case:Dat>	-(y)A
Locative	<Case:Loc>	-DA
Accusative	<Case:Acc>	-(y)I
Ablative	<Case:Abl>	-DAn
Genitive	<Case:Gen>	-(n)In
Instrumental	<Case:Ins>	-(y)lA

of verb inflection is

(4) Verb root/stem-Voice suffixes-Ability-Polarity-Probability-Tense Aspect Modality (TAM) Person suffixes

In this structure, Voice and TAM are group of suffixes. Full verbal paradigm is complex. For the clarity in explanation, it is partitioned into sub-paradigms. These sub-paradigms are, Voice, Ability-Polarity-Probability, TAM and Person. Parts are same as in the structure shown in (4). Details about sub-paradigms are given in the following subsections in the order they appear.

4.3.1 Voice

Voice group has 5 suffixes; Active, Reflexive, Reciprocal, Causative and Passive. Among 5 suffixes of Voice, only Active has zero surface form. Surface form of other Vocie suffixes can be found in figure 4.2. A root or stem can have multiple Voice suffixes according to following order

(5) Root/Stem-Reciprocal or Reflexive-Causative-Passive

As seen in the order, there is a restriction for Reciprocal and Reflexive. They can not exist at the same time.

For the Passive, there are two archmorphemes; -(I)l and -(I)n. Selection of Passive morpheme depends on the phonology. -(I)n morpheme comes after a word anding with l or a vowel.

For causative there are achmorphemes; -dIr, -(I)t, -(A)r and -(Ir). Selection of Causative morpheme also depends on phonology. !!!Eklere göre kurallara bak!!! In the figure 4.2, Causative states are summarized as one state for the sake of simplicity.

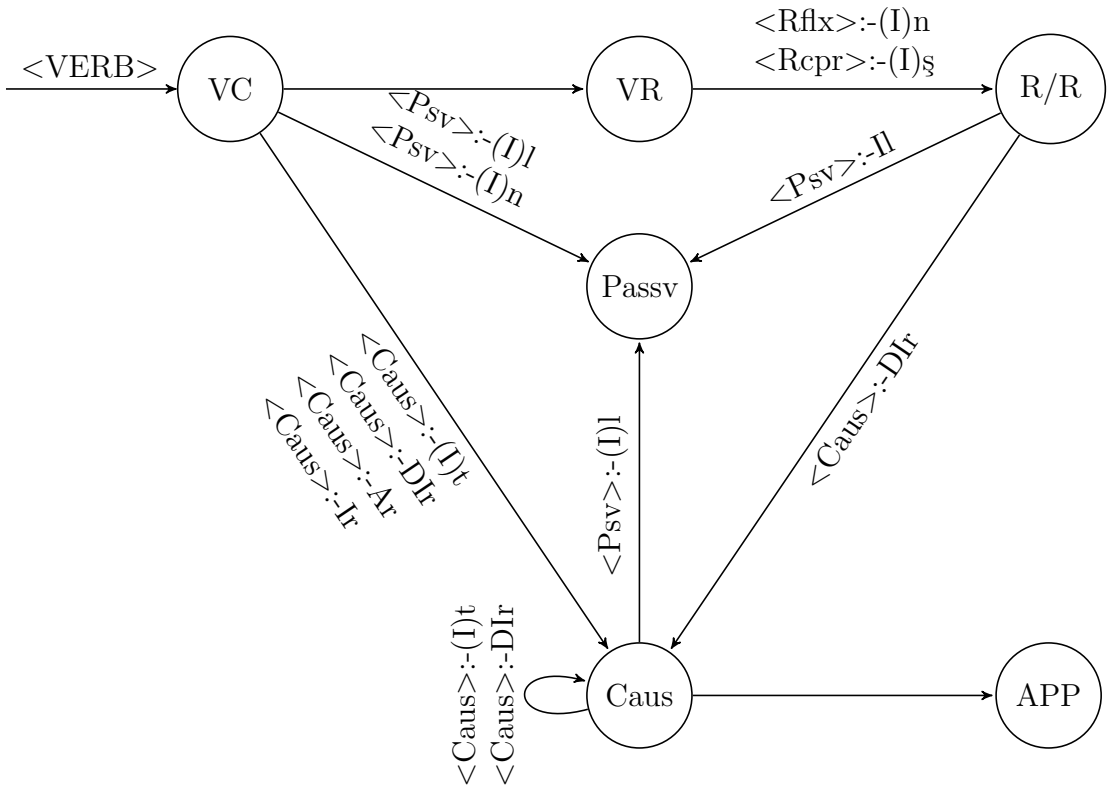

Figure 4.2: Voice paradigm

To implement this phonological suffix selection, we need to create a classing system with FST. Including the No Passive and No Causative situations, there are 5 Causatives and 3 Passives in total. Each Causative can come after each passive, so there are 15 Passive-Causative pairs. We labelled each pair as a verb class. Each class correspond a state that determines the Passive and Causative suffix

that roots will take in our approach. To make this classifying work, all verb root must be directed, this can also be called labelling, to corresponding state.

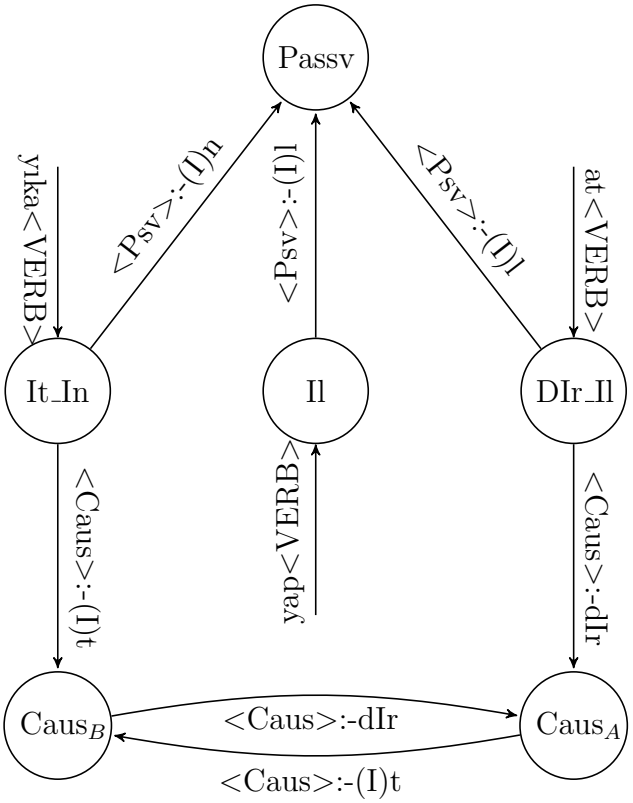

Figure 4.3: Verb classes representing different behavior for Causative and Passive suffixes for different verb roots.

In the figure 4.3, three class states are shown as examples with one input example for each. Caus states of this figure is the expanded version of the Caus state in figure 4.2.

4.3.2 Ability Polarity Probability (APP)

Order of these three suffixes is as following

(6) Ability-Polarity-Probability

The Figure 4.4 shows the FST of APP part. In here, APP state is just a transition state that exist to organize incoming transitions from Voice part to Ability Polarity Probability part. Tamp states are also has same functionality, in difference, Tamps states connects Ability Polarity Probability to Tense Aspect Modality

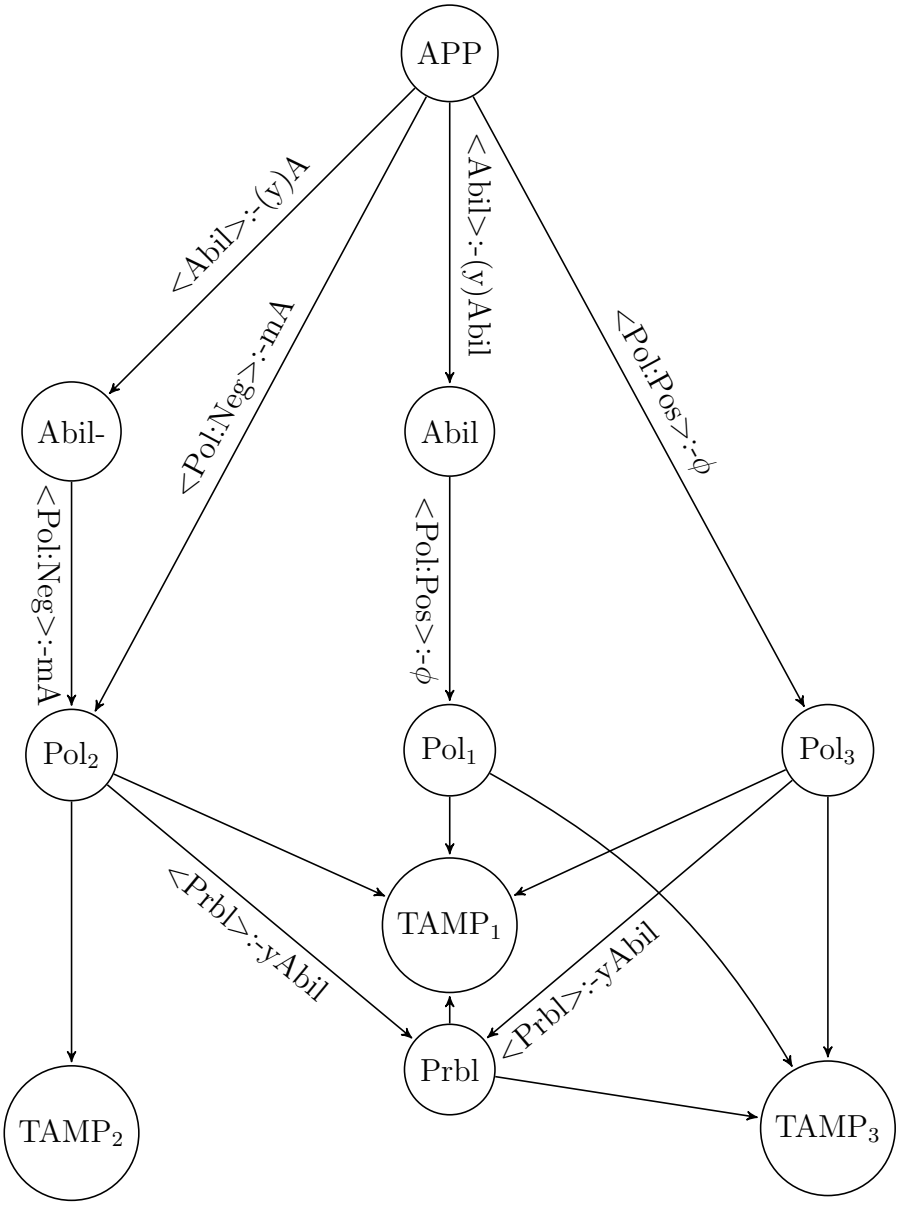

Figure 4.4: Ability-polarity-probability sub-paradigm. Note the ambiguity caused by the identical surface forms of the paths APP-POL_3-PROB-$TAMP_1$ and APP-ABIL-POL_1-$TAMP_1$.

Person and there is three different TAMP states with different purpose. Details of TAMP states will be explained in following section. For he APP part, Polarity has a slot in structure while Ability and Polarity are optional.

Although Ability and Probability has different meaning, 'be able to' and 'might' respectively, they have same surface form. So the surface forms prevent constructions like koş-abil-abil (might be able to run). To solve this problem, the meaning of both suffixes is given on a structure like "koş-abil" (may run). This Turkish

word means both 'might run' and 'be able to run' but not at the same time. This is an ambiguous situation and can only be solved with context. For the analyzer, there 2 analyses for the word koş-abil.

(7) koş<VERB><Active><Abil><Pol:Pos><Tns:Imp><Prsn:3s>

koş<VERB><Active><Pol:Pos><Prbl><Tns:Imp><Prsn:3s>

When the verb has negative meaning, the situation changes. Positive value of Polarity, <Pol:Pos>, suffix has zero surface form. Although it is slot based, its existence has no effect on surface level. But the Negative value of Polarity, <Pol:Neg> has a surface form "-mA". When "-mA' morpheme comes between Ability and Probability, both suffixes can be usedbut Ability change its surface form to "-(y)A". For example, analysis of the word "koş-a-ma-yabil" (might not be able to run) is

(8) koş<VERB><Active><Abil><Pol:Neg><Prbl><Tns:Imp><Prsn:3s>

The "-(y)A" furface form of Ability is only used when Polarity slot has the value of Negative, <Pol:Neg>. Analyse of the "koş-a-ma" is

(9) koş<VERB><Active><Abil><Pol:Neg><Tns:Imp><Prsn:3s>

4.3.3 Tense Aspect Modality Person

The general structure of this paradigm is

(10) Tense - Copula_1 - Person - Copula_2 - Question

In this structure, Tense and Person suffixes has slots, all others are optional. The structure in (10) is a general structure. In some cases, order of Copula suffixes and person suffix change. These cases depends on the Copula and Person values. You can find all possible cases in TAMP FST. The whole TAMP FST is too big so we divided it into three pieces for the convenience of explanation and understanding.

In verb inflection of Turkish, the time that action takes place is denoted by Tense and Copula. Among two, Tense has major part and Copula has supportive role. So, While Tense is obligatory, Copula is optional. There are 10 Tenses with 12 morphemes. These are, Aortive, Imperative, Optative, Conditional, Evidential, Present, Past, Necessitive, and Future tenses. Amog these, Future is divided into two as Future and Imperfective, and Past is divided into two as Past and Narrative. Imperfective and Narrative has slightly different meaning from their parental group. Exceptionally Aortive tense, <Tns:Aor>, has four morphemes. Tenses with their group and surface form are given in Table 4.3.

Table 4.3: Tense categories and their semantics.

Group	**Semantics**	**Abstract morpheme**	**First level output**
1	Aortive	<Tns:Aor>	-Ir, -Ar, -z, -ϕ
2	Imperative	<Tns:Imp>	-ϕ
3	Optative	<Tns:Opt>	-(y)A
4	Conditional	<Tns:Cond>	-sA
5	Present	<Tns:Pres>	-(I)yor
6	Past	<Tns:Past>	-DI
6	Narrative	<Tns:Narr>	-mIş
7	Neccessitive	<Tns:Necc>	-mAlI
8	Future	<Tns:Fut>	-(y)AcAk
8	Imperfective	<Tns:Iprf>	-mAktA

Beside the Tenses, there are 4 Copula groups and morphemes in total. These are, Aortive, Conditional, Narrative and Past. Copulas are divided into two levels. Although all copulas can exist in the First level, only Conditional and Narrative Copulas exist in second level. These Copulas are also can come after inflected nouns to build Nominal Predicates which will be explained in section 4.4. All Copulas with their group and surface form are given in Table 4.4.

Table 4.4: Copula categories and their semantics.

Semantics	**Abstract morpheme**	**First level output**
Aortive	<Cpl:Aor>	-DIr
Past	<Cpl:Past>	-(y)DI
Narrative	<Cpl:Narr>	-(y)mIş
Conditional	<Cpl:Cond>	-(y)sA

In our approach, we grouped tenses according to copula and person behaviours and prepared our FST according to these groups. In the following, all groups will

be explained individually. First group is the most simple group which contains only Imperative tense, <Tns:Imp> with zero morpheme. Imperative tense only accepts person suffixes of group 1. All groups of Person suffix can be found on Tables 4.5 and 4.6 . FST of first group is in Figure 4.5.

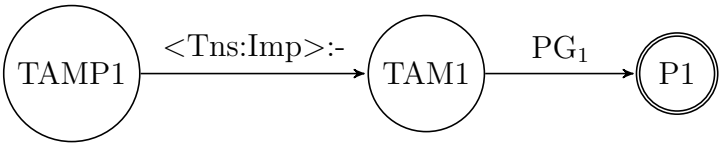

Figure 4.5: Group 1, behaviour of imperative

Table 4.5: Person paradigms for all verb inflection.

Abstract morpheme	**PG$_1$**	**PG$_2$**	**PG$_3$**	**PG$_4$**	**PG$_5$**	**PG$_6$**
<Prsn:1s>		-Im	-m		-(y)Im	-yIm
<Prsn:2s>	-ϕ	-sIn	-n		-sIn	-sIn
<Prsn:3s>	-sIn	-ϕ	-ϕ		-ϕ	-ϕ
<Prsn:1p>		-Iz	-k		-(y)Iz	-yIz
<Prsn:2p>	-(y)In	-sInIz	-nIz		-sInIz	-sInIz
<Prsn:2p>	-(y)InIz					
<Prsn:3p>	-sInlAr	-lAr	-lAr	-lAr	-lAr	-lAr

Table 4.6: Person paradigms for all verb inflection.

Abstract morpheme	**PG$_7$**	**PG$_8$**	**PG$_9$**	**PG$_{10}$**	**PG$_{11}$**
<Prsn:1s>	-yIm	-m			-yIm
<Prsn:2s>	-sIn		-sIn		-sIn
<Prsn:3s>	-ϕ		-ϕ	-ϕ	-ϕ
<Prsn:1p>	-lIm	-yIz			-yIz
<Prsn:2p>	-sInIz		-sInIz		-sInIz
<Prsn:3p>	-lAr		-lAr	-lAr	

The second group consist of only Optative Tense, <Tns:Opt> with surface form of "-(y)A". Optative tense can accept three copulas and person suffix changes according to existence of copula. Optative tense can get a question suffix. Like person states, question can be a final state. FST of this group is in Figure 4.6. Note that FST in Figure 4.6 shares some states with the FST in Figure 4.5.

The third group includes only Conditional tense, <Tns:Cond> with surface from "-sA". Like Optative tanse, it takes same two copulas of first level but person suffixes that comes right after the tense are belong to different group. Additionally, Conditional tense may have copula after the question suffix. Because of these

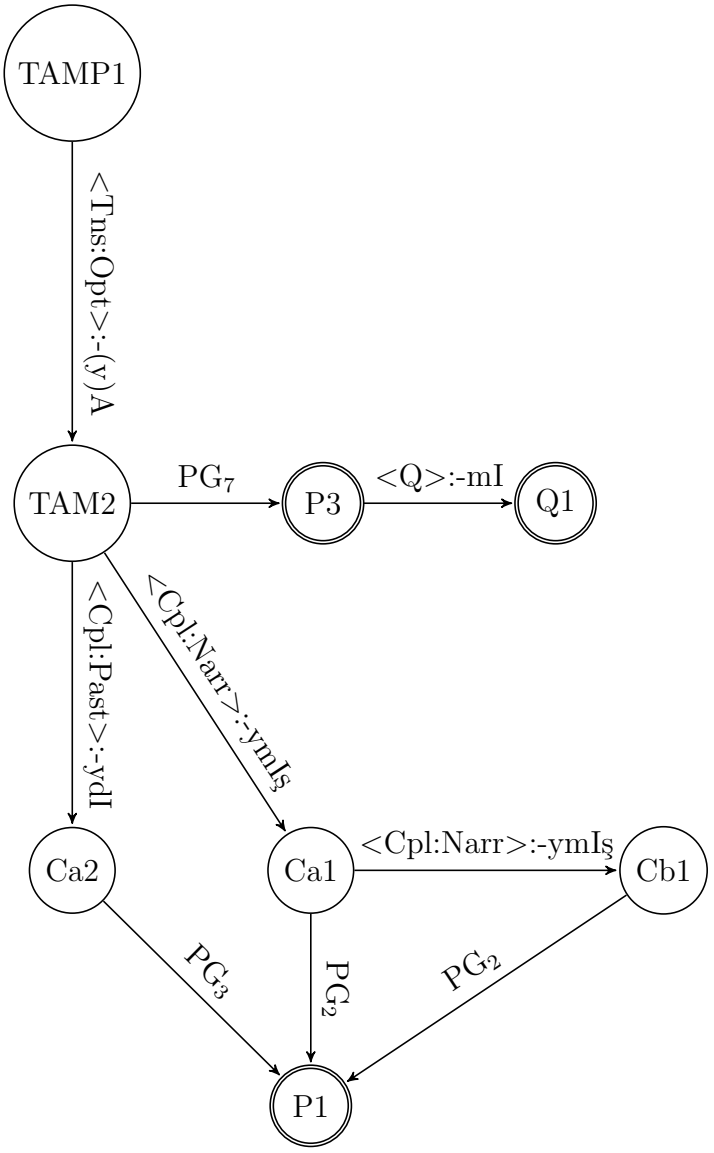

Figure 4.6: Group 2, behaviour of Optative

differences, conditional tense required a new group. Figure 4.7 shows the FST of third group. In this figure, to make the figure clear, states come after Ca1 and Ca2 are not shown because they are same as in the Figure 4.6. This reduction is applied on all tense group figures.

The fourth group consists of only Past Tense, <Tns:Past> with first level output "-DI". Past tense accepts two copulas and all person suffixes are in group three. FST of this part is in Figure 4.8.

The fifth group is created to handle exceptional behaviour of Aortive copula. So all tenses accept the copula, <Tns:Necc>, <Tns:Pres>, <Tns:Iprf>, <Tns:Fut>,

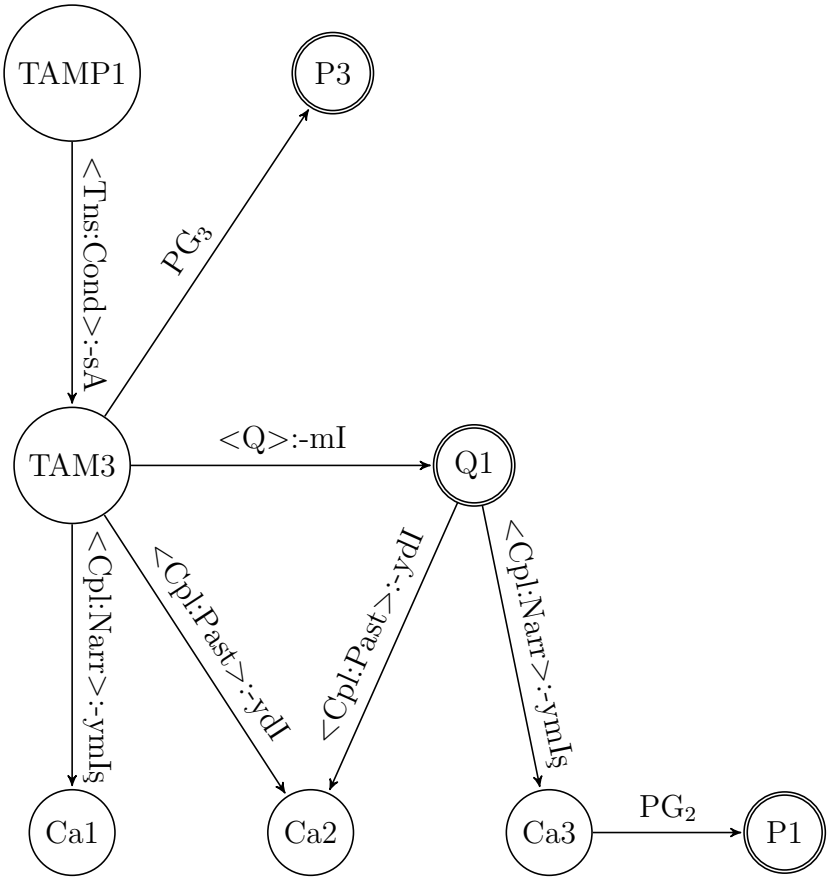

Figure 4.7: Group 3, behaviour of Conditional

<Tns:Narr>, are in this group. Normally, person suffixes come before Present copula with the exception of <Prsn:3p>. This suffix also can come after the copula. Figure 4.9 shows the FST of this part.

The sixth group contains <Tns:Necc>, <Tns:Pres>, <Tns:Iprf>, <Tns:Fut> and <Tns:Aor> tenses. This group handle all behaviours of these tenses excluding Present copula. It has slightly complex FST with all possible combinations of copula, person and question suffixes. In contrast with other groups, Group 6 originated from 2 different states TAMP1 and TAMP3. In total, there are 3 different TAMP states, TAMP1, TAMP2 and TAMP3. TAMP2 and TAMP3 only exist to handle unusual behaviour of Aortive tense. Unlike other tenses, Aortive tense changes surface form when it comes right after the <Pol:Neg> suffix. This condition creates the TAMP2 and TAMP3. Else, it just act as other tenses in group six with the surface forms "-Ir", "-Ar". Figure 4.10 shows the FST of sixth group.

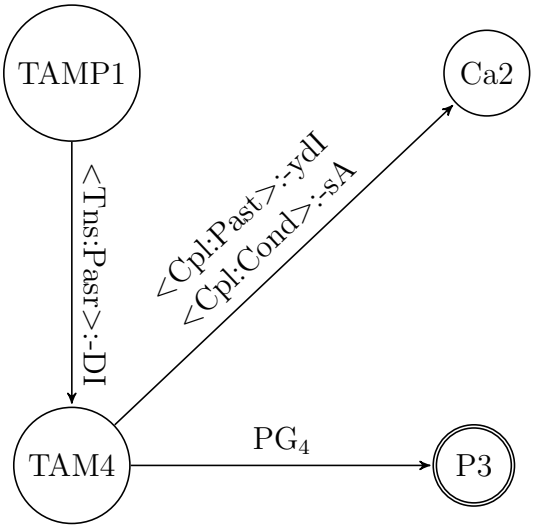

Figure 4.8: Group 4, behaviour of Past

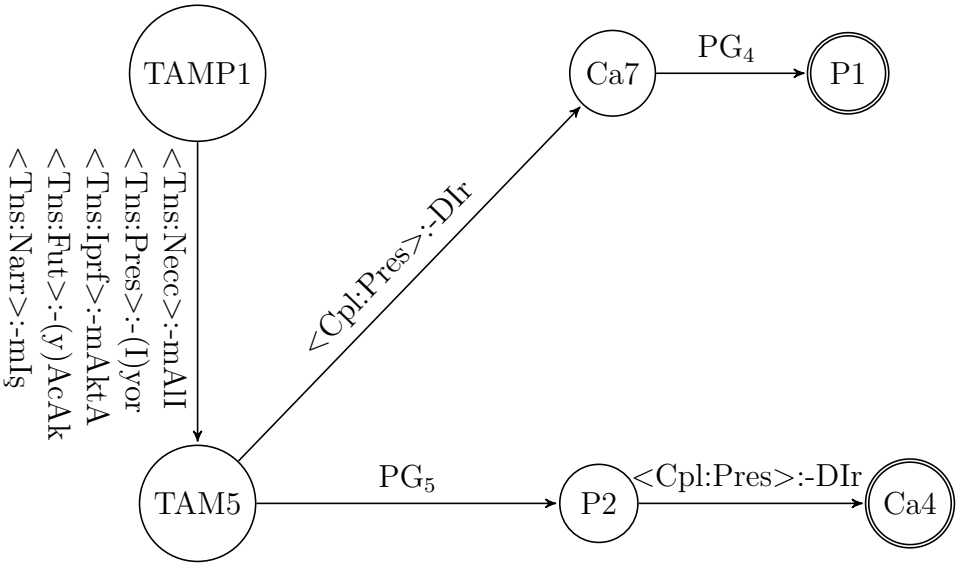

Figure 4.9: Group 5, behaviour of Past

The seventh group handles the behaviour of Narrative tense. FST of this part is in Figure 4.11

Aortive tense has four different surface forms of "-Ir", "-Ar", "-z" and zero morpheme. Aortive tense change its surface form into "-z" or zero morpheme when it comes right after the negative polarity suffix. Groups eight and nine handles these surface forms of the aortive suffix. We divided two forms into two groups because of the different behaviours. While zero morpheme accepts no suffix other than person first singular and plural, "-z" form gets copula and others. The Figure 4.12 shows the FST of these two groups.

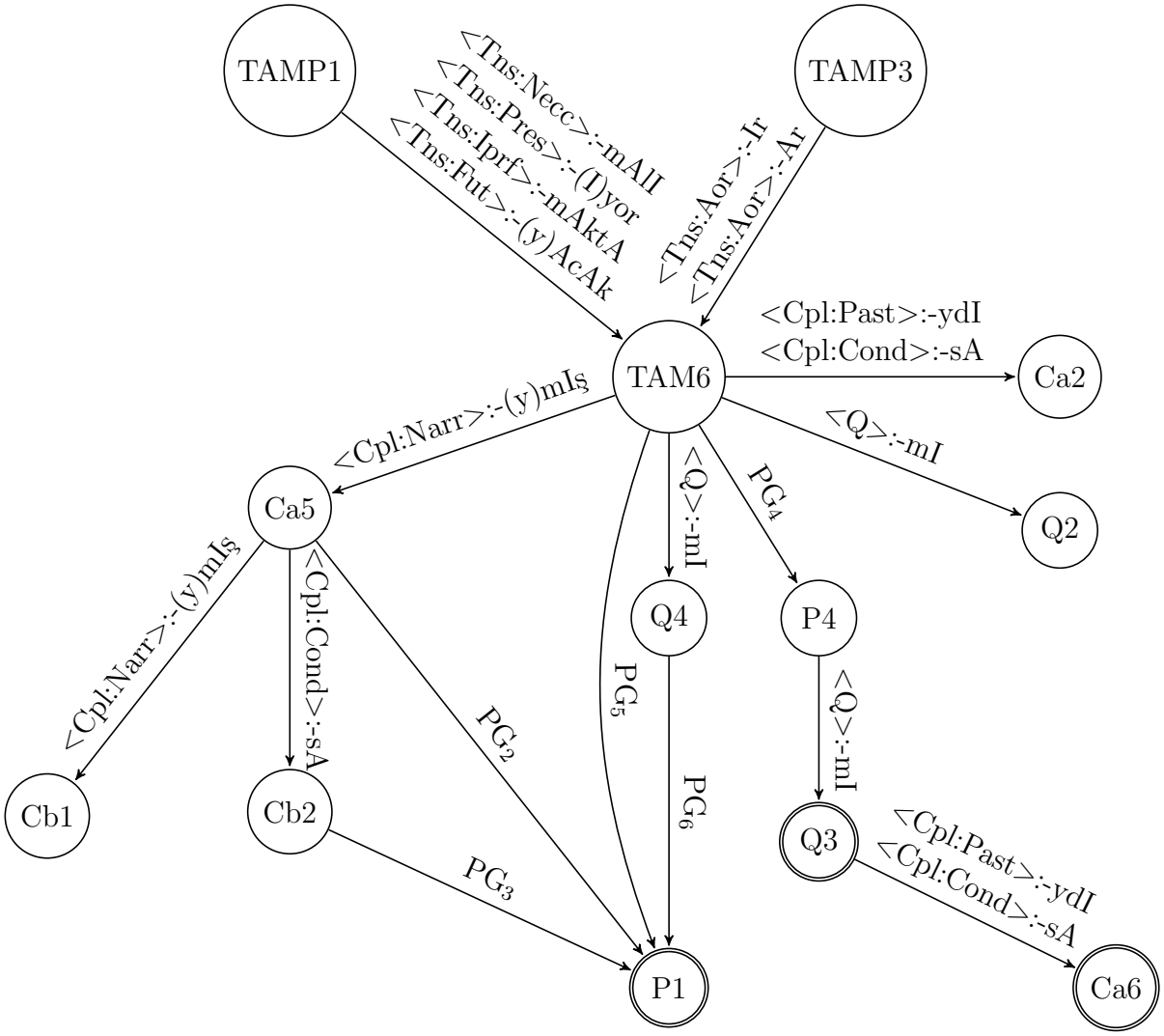

Figure 4.10: Group 6

4.4 Nominal Predicate

In Turkish sentences, inflected Nominals can serve as predicate in sentence. These sentences are called Nominal sentences. The following is a Nominal sentence example with the analysis of predicate word.

(11) Bu araba mavidir

(12) Bu araba mavi<NOM><Num:sg><Poss:None><Case:Nom><Cpl:Aor><Prsn:3s>

(13) This car is blue

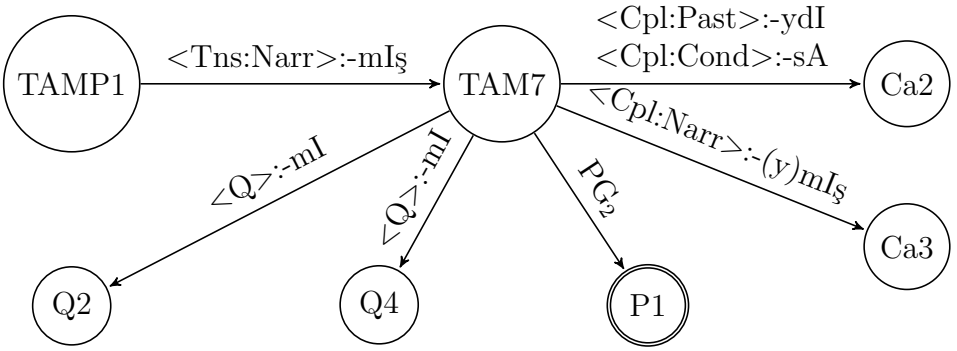

Figure 4.11: Group 7

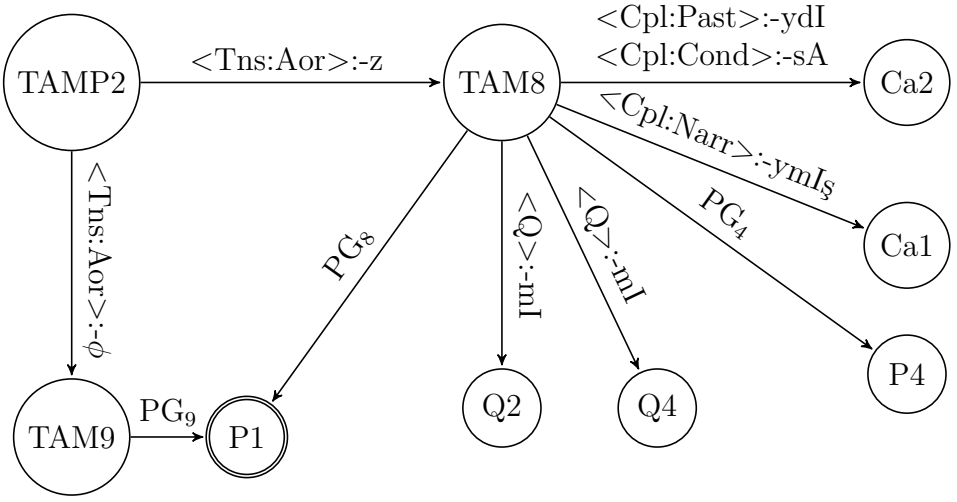

Figure 4.12: Group 8

Except Accusative Case, all Cases can be inflected to be predicate. To handle this, we separate the Accusative Case from others. When a nominal becomes predicate, it is treated like a verb in Section 4.3.3 with the exclusion of the Tenses. This means Copulas, Person suffixes and question suffix comes after predicate. We give the all possible structures in Figures 4.13 and 4.14. Nominal predicate FST has common states with verbal inflection FST.

The "-lAr" complication that occurs in Nominal Inflection also exist here. It appears when Present Copula, <Cpl:Pres>, with zero morpheme comes before the Third person plural, <Prsn:3p>, with surface form "-lAr" suffix. To handle this situation, we divide Predicate into two part as Pred1 and Pred2. Input of Pred2 is guaranteed to have any kind of "-lAr" in it. This is given in Nominal Inflection FST in Figure 4.1.

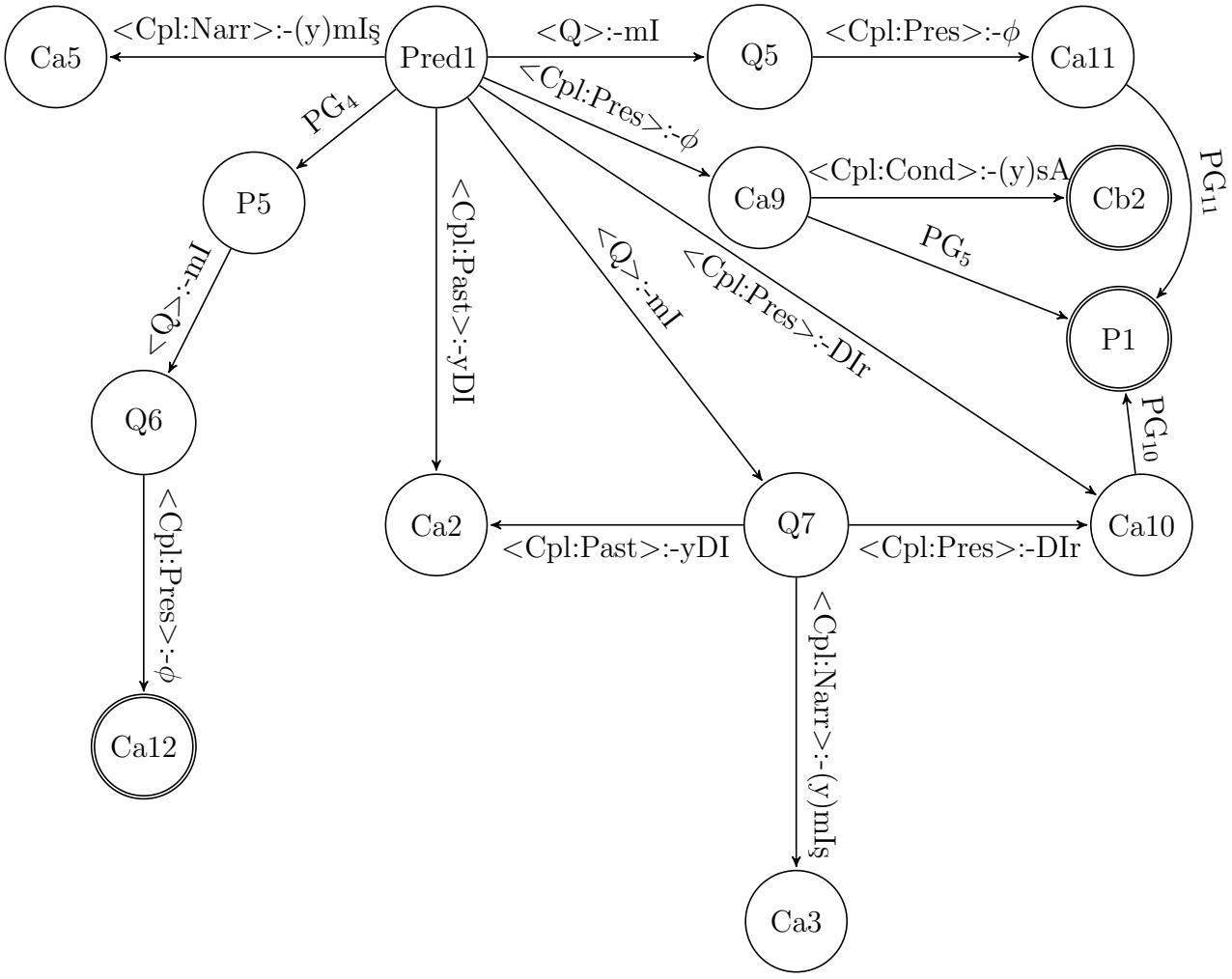

Figure 4.13: Nominal Predicate Pt1

4.5 Other Classes

Other than nominals and verbs, there are 5 more classes. These are: adverbs, punctuations, conjunctions, interjections and postpositions.

Among them, punctuations are not words but they should still be analysed. Punctuations are '.', ',', ':', ';', '!', '"', '(', ')', '"', '?'. Punctuations don't undergo any kind of inflection or derivation so it's FST is only consist of an single accepting state.

Like punctuations, postposition, conjunctives and interjections don't undergo any inflections and derivation in general. Postposition and interjections might be used as predicates.

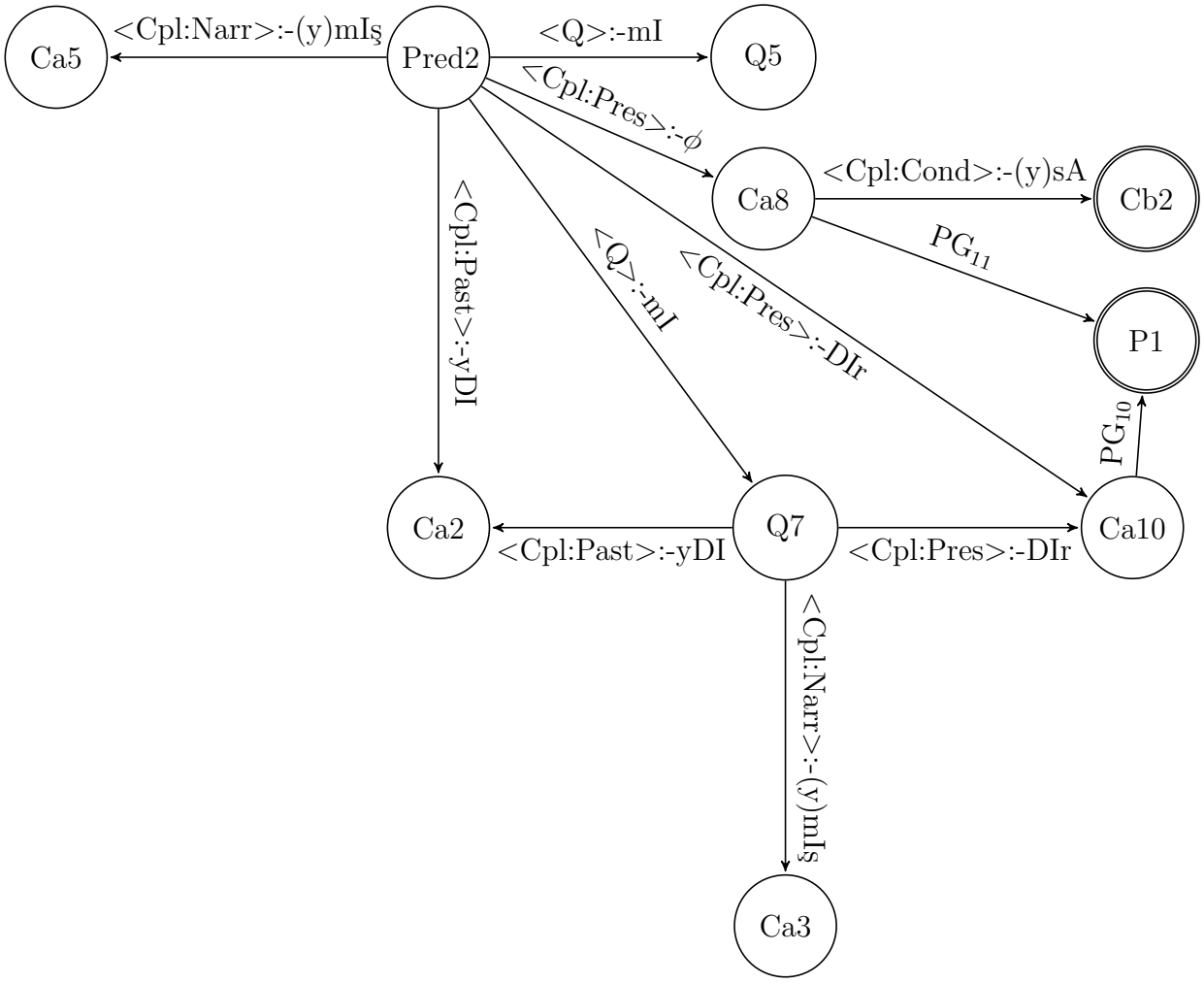

Figure 4.14: Nominal Predicate Pt2

Adverbs don't undergo any kind of inflection or derivation but still they can undergo Nominal Predicate inflection given in Figures 4.13 and 4.14. Also there are Nominals that can serve as adverb.

4.6 Derivation

Turkish has strong derivational structure. Almost every morphological class can derived into another one. Not all derivational suffixes are as productive as others. Non-productive suffixes are used as embedded into stem word. This approach is used in this project and derived words with non-productive suffixes are added into the lexicon. So, the derivation FST is based on only productive derivational suffixes. You can find the derivation FST on figure 4.15. As you can see, some

states of derivation FST, is part of inflectional FSTs. That is because of the derivation can occur only spesific parts of inflections.

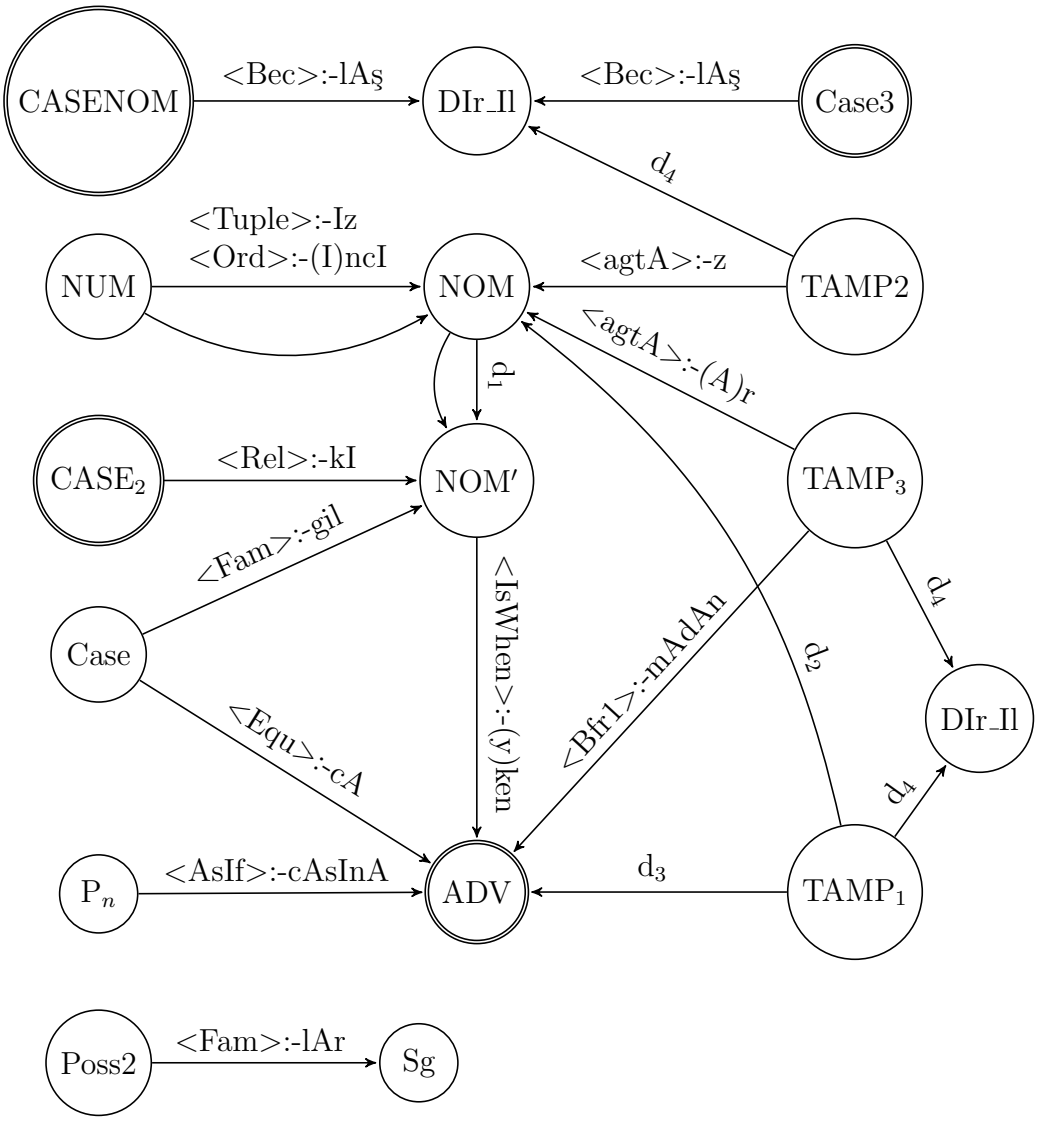

Figure 4.15: The derivation FST.

In nominal inflection, derivation may occur just before the inflection with several suffixes for example in Case states. In Section 4.4, Case is divided into two. Similar situation occurs here. Accusative Case doesn't undergo any derivational operation. We could use just two states but every case doesn't take every derivational suffix. To solve this, we divided Case states again. The Figure 4.1 shows all Case states and Figure 4.15 shows which Case state takes which derivational suffixes.

In Verbal Inflection, Derivation comes after the Probability in APP but there is one exception and that is the "-CAsInA" suffix. This derivation requires a tense

Table 4.7: The most productive derivational morpheme sets d_1, d_2, d_3 and d_4 used in Figure 4.15.

Set	**Semantics**	**Key**	**First level output**
d_1	Diminutive of N	<Dim1>	-CIk
d_1	Diminutive of N	<Dim2>	-CAğIz
d_1	Occupation of N making, selling	<Occup>	-CI
d_1	With N, having N	<With>	-lI
d_1	Without N	<Without>	-sIz
d_1	State of being N	<State>	-lIk
d_1	Like N	<Like1>	-(I)msI
d_1	Like N	<Like2>	-(I)mtrak
d_2	Agent who regularly does V	<AgtR>	-(y)IcI
d_2	Agent who did V	<AgtP>	-mIş
d_2	Agent who will do V	<AgtF>	-(y)AcAk
d_2	Agent who does V	<AgtA>	-(y)An
d_2	Agent who likely does V	<AgtL>	-(y)AsI
d_2	Agent who is doing V	<AgtI>	-(I)yor
d_2	Action V, past or aortive	<InfA>	-dIk
d_2	Action of V	<Inf1>	-mA
d_2	Action of V	<Inf2>	-mAk
d_2	Action of V	<Inf3>	-(y)Iş
d_2	Action of V, future	<InfF>	-(y)AcAk
d_3	While doing V	<While>	-(y)ArAk
d_3	After doing V	<After1>	-(y)IncA
d_3	Since doing V	<Since>	-(y)AlI
d_3	Until doing V	<Until>	-(y)IncAyA
d_3	After doing V	<After2>	-(y)p
d_4	Keep doing V	<KeepDo>	-(y)Adur
d_4	Easy doing V	<EasyDo>	-(y)Iver
d_4	Almost done V	<AlmostDo>	-(y)Ayaz

and a person suffix before it. The derivation FST in Figure 4.15, "-CAsInA" starts from a state named PN. This PN stands for all Person states like P1, P2 and etc.

To represent the derivation in analyses, the same format used for suffix and tag pairs is used. Additionally, new morphological class of the word is showed right after the derivation suffix.

(14) ev<NOM><Less><NOM>

In this example, you see the analysis of the word “evsiz” (homeless). The word is derived from the nominal root “ev” (home) to nominal stem “evsiz” (homeless) with the suffix of “-sIz”. In the analysis, morphological class of the root comes right after the root and morphological class of the derivation result comes right after the derivation suffix tag.

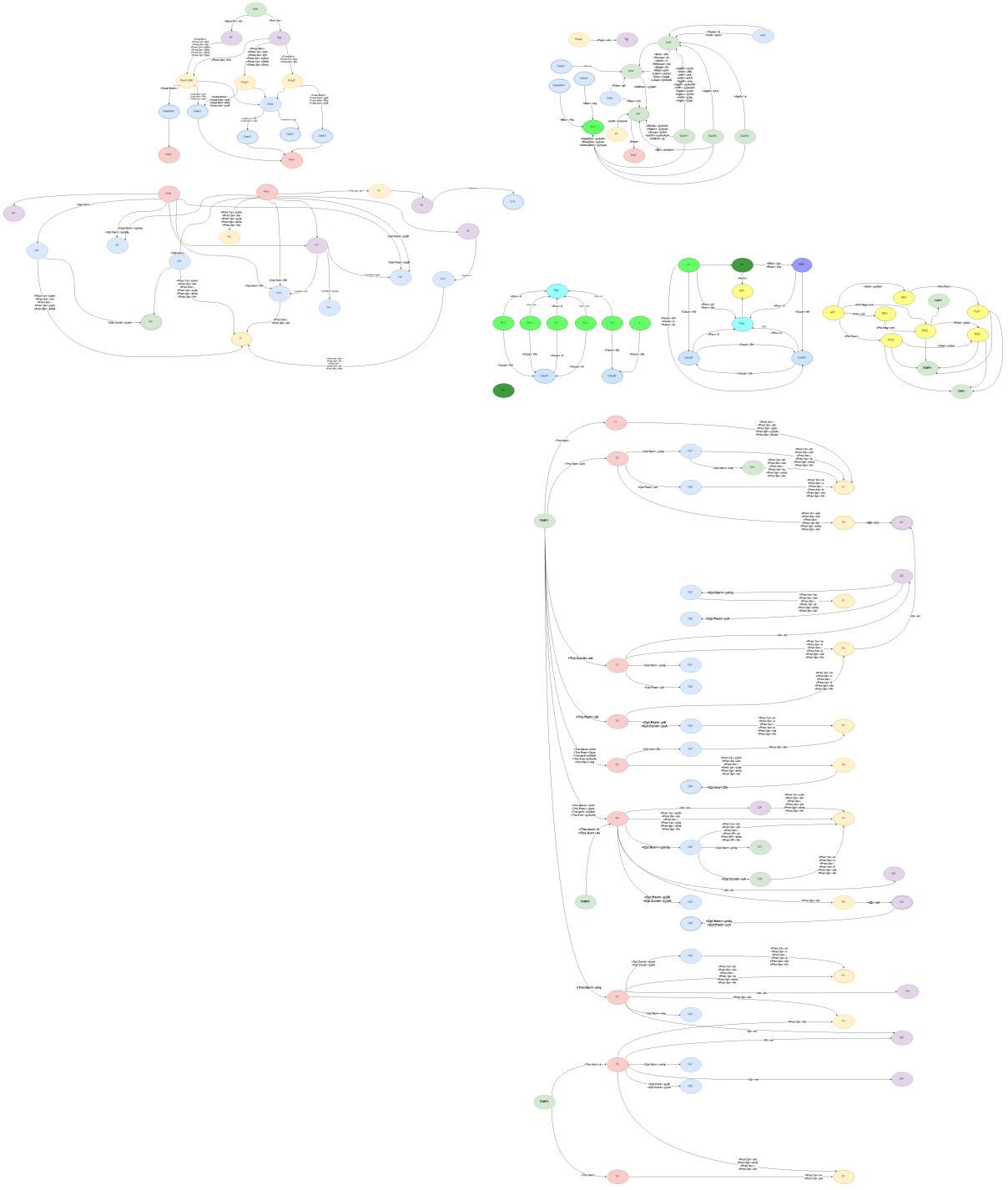

Figure 4.16: Whole FST of the Turkish Morphology

4.7 Phonology

There are several phonological rules in Turkish and almost all rules depend on the phonological context of the root or other, previous, suffixes.

In the first level output of several suffixes, “(y)” exists. In Chapter 3 this character named as optional y. In Turkish, two vowels of different suffixes can’t get together. This optional y exist to solve this problem, like other optional consonants. If a suffix that have optional consonant at the beginning comes after a suffix or root that ends with a vowel, the optional consonants is preserved. Otherwise it is dropped. For example Dative Case, <Case:Dat>. It’s surface form is “-(y)A”. If this suffix come at the end of a word like “araba”, car, it’s surface form become “-yA”. If it comes at the end of a word like “okul”, school, it’s surface form become “-A”. In some cases, it is guaranteed that “(y)” is going to preserved or dropped. For example when Future Tense, “-(y)AcAk”, is followed by Narrative Copula, “-(y)mIş”, it is certain that “(y)” is preserved because previous suffix has “k” at the end. So we changed the Narrative Copula’s surface form from “-(y)mIş” to “-ymIş”. With this, a phonological rule is implied on the morphology level.

Similar phonological rule exist for some certain vowels, like (I). In this situation the vowel is both optional and archphoneme. These two are independent from each other. For example, First Person Singular Possesivity suffix, <Poss:1s>, has surface form “-(I)m”. If this suffix follows a word like “masa”, table, the surface form become “-m”. If it follows a word like “telefon”, telephone, it become “-Im”.

Another rule is deciding what will an archphoneme change into. Or with it’s formal name, Big Vowel Harmony. There are two different archphoneme and both has different rule. For the archphoneme “A” the rule implies that back vowels are followed only by back vowels and fron vowels followed by only front vowels. So archphonemes change accorsing to previous vowels. “A” archphoneme can change into “a” or “e”. For the archphoneme “I”, being Flat or Round gets involved into situation in addition to Front and Back. The archphoneme “I” can change into “ı”, “i”, “u” or “ü”. Following two examples are given for this rule.

my pen, kalem-(I)m, kalem-im

to school, okul-(y)A, okul-a

Similar simple rule exist for the archphonemes C, D and K. If C or D comes after one of the consonants of “f”, “s”, “t”, “k”, “ç”, “ş”, “h”, “p”, C become “ç”

and D become “t”. Otherwise they remain as “c” and “d” respectively. For the archphoneme K, rule is different. If K is followed by any vowel, it becomes “ğ”. Otherwise it remains as “k”. Following three examples are given for this three phonological rules.

from tree, ağaç-DAn, ağaç-tan

I will do, yap-(y)AcAK-Im, yap-acağ-ım

book seller, kitap-CI, kitap-çı

There are some phonetic rules that occur in the word root. The first rule called doubling and occurs on some significant roots like “hak”, right. When this root gets a suffix starts with a vowel, last consonant of the root copies itself. For example, when “hak” gets <Poss:3s> suffix, it becomes “hakk-ı”, his/her right. In the second level of HFST, it is not possible to add a new character to the string, so implementing this rule directly is not possible. To solve this problem, we stored all roots that doubling occurs in their double consonant form and with an additional symbol, {DOUB}, that indicates the doubling. The rule deletes the symbol and if necessary, doubled consonant.

Another phonetic rule that occurs on root is penultimate drop. In this rule some specific roots, for example “akıl”, mind, drops their last “i”, “ı”, “ü” or “u” when it gets a suffix starts with a vowel. When the root “akıl” get the suffix ¡Poss:1s¿, it becomes “akl-ım”. There is no specific root pattern for this rule to occur, so it is completely lexicon dependent. To implement this rule, we marked necessary roots’ the last “i”, “ı”, “ü” or “u” with a special symbol for each and wrote a rule to drop or change these special symbols.

Chapter 5

Implementation

To implement the two level approach, we used HFST of Helsinki University. HFST stands for Helsinki Finite-State Transducer. HFST is designed for the implementation of morphological analysers. Also, it can be used for other systems that are based on transducers. HFST has a feature supporting weighted transducers which is not used in this project.

5.1 HFST Structure

A HFST structure consist of four different type of files which are ".lexc", ".twol", ".hfst" and ".ol" files. Code is written into ".lexc" and ".twol" files. In general, these two files are text files and can be modified by any text editor. Content of these files will be explained in Section 5.2.

".lexc" files correspond to the first level which is morphotactics. There can be more than one lexc file but they work as one. Lexc files must be compiled into a single structure. This is done by the command "hfst-lexc". This command gathers all lexc files into one file and creates a ".hfst" file which is the output of the compilation process. Following example shows usage of hfst-lexc command.

(1) hfst-lexc words.lexc ninfl.lexc -o lex.hfst

Twol files correspond to the second level which is phonological layer. Each twol file must contain at least one rule. Compilation of twol files are done by the command "hfst-twolc -R". A hfst file is created as result of compilation. There can be several twol files and compiled versions, hfsts, of them are combined to

create an intersection between them. With this, all rules are applied to second-level inputs as one file. This intersection is done by the command "hfst-compose-intersect". As a result of this command an hfst file is created. Following examples are given for "hfst-twolc -R" and "hfst-compose-intersect".

(2) hfst-twolc -R -i dropLetter.twol -o dropLetter.hfst

(3) hfst-compose-intersect -1 dropLetter.hfst -2 penult.hfst -o phon1.hfst

(4) hfst-compose-intersect -1 phon1.hfst -2 doub.hfst -o phon2.hfst

(5) hfst-compose-intersect -1 phon2.hfst -2 changeA.hfst -o phon3.hfst

(6) hfst-compose-intersect -1 lex.hfst -2 phon7.hfst -o tr.hfst

Order of the files is important while combining them. Especially, when there are some rules affecting other rules. In the examples (3), (4) and (5) shows 4 different HFST files combined in specific order. These files are part of MorAz. "dropLetter.hfst" consists of rules that deciding whether optional letter, like "(y)", will be preserved or be dropped. "changeA.hfst" consists of rules that deciding what form will the archphoneme "A" take. "dropLetter.hfst" file affects "changeA.hfst" because there is optional archphoneme "(A)". Before deciding its form, must decide to preserve or drop the archphoneme. To implement this, "dropLetter.hfst" file must come before "changeA.hfst" in the order.

"hfst-compose-intersect" command can also combine HFST of morphotactics and phonetics as shown in the example (6). This hfst file, tr.hfst in the example (6), is morphotactics and phonetic rules together. There are two more steps to create the analyzer.

First, this hfst file needs to be inverted. Due to HFST's two way structure, both analyzer and generator can be created. Generator is created from the main HFST file, "tr.hfst" in example (6). And analyser is created from its inverse. Inversion is done by the command "hfst-invert". A HFST file is created as result. Following example shows the usage of this command

(7) hfst-invert -i tr.hfst -o tr.inv.hfst

Lastly, to create runnable analyser and generator files, the “hfst-fst2fst -O” command is used. This command turns hfst file into a “.ol” file. This command is same for both generator and analyser. Following examples shows usage of this command.

(8) hfst-fst2fst -O -i tr.hfst -o generate.ol

(9) hfst-fst2fst -O -i tr.inv.hfst -o analyze.ol

These two files can be executed on terminal by the command “hfst-lookup”. When files executed, user input words are required. The result is shown right after the input and new input is waited. Following example shows the usage of this command.

hfst-lookup analyze.ol

5.2 HFST Syntax

A lexc file has basic syntax. On the very first line of the file “Multichar_Symbols” to indicated defined multichar symbols. It allows to create any kind of symbol to use later. “ % ” character is used as escape character and can be used in multichar symbols.

After declaration of multichar symbols is done, definition of states starts. Each state is called as Lexicon here. Definition of a state starts with the word “LEXICON” after that, name of the comes with a white space. This line indicates that, a state, lexicon, code is starting from here. A state code contains several transitions and ends with start of another lexicon. A transition contains two parts: a mapping between two strings, these strings both can be defined multichar symbols or ordinary characters, and the destination of the transition which is a state. Every transition line terminated with a “;”. mapping in the transition is represented by the character “:” and it is possible to use white space in mapping. Also, absence of mapping is possible. If the state is a final state than a transition of the state must include “#” which indicates the end. Following examples shows possible lexicon definition.

(10) LEXICON Nom’
%<number%:pl%>:%-l%{A%}r NumPl;
%<number%:sg%>:% NumSg;

(11) LEXICON Case2
#;
Pred1 ;
Case2Deriv ;

Syntax of twol files are a bit more complex. The code consist of three sections. First, alphabet comes. In here, all symbols, ordinary symbols and multichar symbols that we defined in the lexc files, are written. Alphabet section starts with the “Alphabet” and all symbols are separated by white space and can be written line by line. Alphabet section is terminated with “;”. Example (12) shows the alphabet that is defined in “changeA.hfst” file.

(12) Alphabet
A B C Ç D E F G Ğ H I İ J K L M N O Ö P R S Ş T U Ü V Y Z
a b c ç d e f g ğ h ı i j k l m n o ö p r s ş t u ü v y z
%{A%}
%{I%}
%{D%}
%{C%}
%{K%}
%{SPC%}
%-
;

After the alphabet, Sets section is defined. This section is optional and contains custom created sets of symbols. These sets are used in the rules. Syntax starts with “Sets” word. Every set has a name and list of symbols. These name and lists connected to each other with “=”. Every set section terminates with “;”. Example (13) shows the sets that are defined in “changeA.hfst” file.

(13) Sets

Vow = A E I İ O Ö U Ü

a e ı i o ö u ü ;

Cons = B C Ç D F G Ğ H J K L M N P R S Ş T V Y Z

b c ç d f g ğ h j k l m n p r s ş t v y z ;

Back = A I O U

a ı o u ;

Front = E İ Ö Ü

e i ö ü ;

beforeBackA = B C Ç D F G Ğ H J K L M N P R S Ş T V Y Z

b c ç d f g ğ h j k l m n p r s ş t v y z

A I O U

a ı o u

%{A%} %{I%} %{D%} %{C%} %{K%} %- ;

beforeFrontA = B C Ç D F G Ğ H J K L M N P R S Ş T V Y Z

b c ç d f g ğ h j k l m n p r s ş t v y z

E İ Ö Ü

e i ö ü

%{A%} %{I%} %{D%} %{C%} %{K%} %- ;

Last part is rules part. Rules of the twol file are defined here. After writing "Rules", every rule is defined with a name, name is written between quotation marks, and rule itself. Rule starts with the mapping of symbols. This mapping indicated with ":" again. After that "<=>" is written. This symbol connects the mapping and the regular expression part of the rule. Regular expression is written to catch the conditions when mapping occurs. Location of the mapping is determined with underscore character in the regular expression. Rule line is terminated with ";". Example (14) shows the rules that are defined in "changeA.hfst" file.

(14) Rules

"A to a"

%{A%}:a <=> Back: [Cons:] [%:] [beforeBackA:] _ ;

"A to e"

%{A%}:e <=> Front: [Cons:] [%:] [beforeFrontA:] _ ;

5.3 MorTur Structure

MorTur consist of 6 Lexc files and 8 twol files. These 14 files and an additional make file are enough to create a runnable version of the analyser and generator. All commands required to compile and create the program are executed by make-file. This is the hfst part of the program and it is described in sections 5.1 and 5.2 with the examples from the MorTur's source code.

The analyser can run on terminal if there is no custom created interface. Using it on the terminal is not convenient so we designed a web page as an interface for the analyser. Web page is reachable from the hppt://ddil.isikun.edu.tr/mortur. It is a simple interface with one input and one output pane. One or more words can be given as input. In multi-word input situation, words need to be separated by new line character. The result is shown in same order as the input.

On the background, Django server runs the analyser. The server also does some post processing on analyser result. As we described in section 3.2, we handled some phonetic rule by editing the roots in the lexicon. In hfst, analyser result includes exact form of the root in the lexicon. So in default output, a marked root in the lexicon is seen as marked. Server code does a post-processing on outputs to change any marked roots into their original form.

Chapter 6

MorAz

During the implementation of MorTur, we implemented another analyser which is for Azerbaijani Turkish, [12]. Analyser is publicly available on http://ddil.isikun.edu.tr/moraz/, [13]. Azerbaijani Turkish is one of the Turkic languages and it is similar to Anatolian Turkish. By this simultaneous implementation, we could clearly see the differences and similarities between these two Turkic languages.

In broad perspective, they are similar. This similarity can seen by comparing the FST diagrams of both languages. For example Nominal inflections are nearly identical. The significant difference is on suffixes. There are suffixes of same class but different surface forms. Naturally, dialect difference cause this. An example, is the <Prsn:1s> suffix of verbal inflection. In Azerbaijani Turkish it's surface form is "-(y)Am". Also, paradigms in person suffixes of AT are quite different from Turkish Person suffixes of AT are given in Table 6.1.

Table 6.1: Person paradigms for all verb inflection in AT.

Abstract morpheme	$\mathbf{p}_1$	$\mathbf{p}_2$	$\mathbf{p}_3$	$\mathbf{p}_4$	$\mathbf{p}_5$	$\mathbf{p}_6$	$\mathbf{p}_7$	$\mathbf{p}_8$
<Prsn:1s>	-Am	-m	-(y)Im	-m		-Am	-(y)Am	
<Prsn:2s>	-sAn	-n	-ϕ	-sAn	-sAn		sAn	
<Prsn:3s>	-ϕ	-ϕ	-sIn	-ϕ	-ϕ			-ϕ
<Prsn:1p>	-IQ	-Q	-(y)AQ	-Q		-IQ	-(y)IQ	
<Prsn:2p>	-sInIz	-nIz	(y)In	-sInIz	-sInIz		sInIz	
<Prsn:3p>	-lAr	-lAr	-sInlAr	-lAr	-lAr			-lAr

Other than dialectal difference, there is another deeper difference. Complexity of languages. From east to west, while the language evolved, morphologies seem to have become more complex. For example, the ways of handling Nominal predicate

of Azerbaijani Turkish is simpler than Anatolian Turkish. In Azerbaijani Nominal Predicates, Copulas and Persons have fixed location. By this, the problem caused by "-lAr" and <Cpl:Pres> is handled or never existed. Figure 6.1 shows the Nominal Predicate FST of AT to indicate difference clearly.

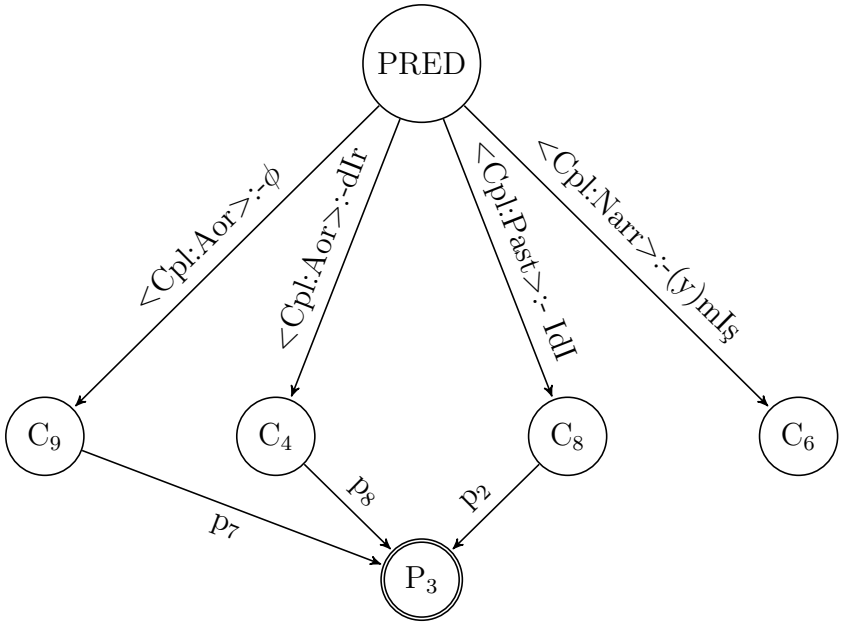

Figure 6.1: The nominal predicate for AT.

Chapter 7

Results and Discussion

There are two different implementation of morphological analyser of Turkish. TrMorp [10] and Morphological Analyzer of ITU NLP Group (ITU-MA) [11]. They are both publicly available. Both have some linguistic problems. We implemented MorTur hopefully free from these problems.

Both of the analysers are implemented using FSTs. Stuttgart Finite State Transducer (SFST) is used to implement TrMoph and flag diacritics which are an extension of Xerox Finite State Transducer is used to implement morphological analyser of ITU NLP. Below we provide examples of linguistic problems for the TrMorph and ITU analyser.

Family suffix is a significant derivational of Turkish morphology. It has two abstract morphemes as "-gil" and "-lAr" and gives the meaning of plurality in a special way. For example, the word "annemgil" or "annemler" mean "my mother and my siblings if there any, my father or just whoever is with my mother". Both of the analysers neglect this suffix in different ways. TrMorph recognizes only the abstract morpheme "-gil", so "lAr" is neglected. ITU-MA pretends like there is no family suffix. Analyses of ITU-MA does not include family suffix. Following examples shows analyses of the words "annemler" and "annemgil" by both two analyzers.

(1) Analyses of "annemgil" by TrMorph
anne<N><p1s><gil><N>
anne<N><p1s><gil><N><0><V>
anne<N><p1s><gil><N><0><V><cpl:pres><3p>
anne<N><p1s><gil><N><0><V><cpl:pres><3s>

(2) Analyses of “annemler” by TrMorph

anne<N><p1s><0><V><cpl:pres><3p>
anne<N><p1s><pl><0><V><cpl:pres><3p>
anne<N><p1s><pl>
anne<N><p1s><pl><0><V>
anne<N><p1s><pl><0><V><cpl:pres><3p>
anne<N><p1s><pl><0><V><cpl:pres><3s>

(3) Analyses of “annemgil” by ITU NLP

annemgil+Guess+Noun+A3sg+Pnon+Nom

(4) Analyses of “annemler” by ITU NLP

anne+Noun+A3pl+P1sg+Nom

In our implementation, we considered Family suffix in all of its form. Following examples show the analyses of “annemler” and “annemgil” by MorTur.

(5) Analyses of “annemgil” by MorTur

anne<NOM><Num:Sg><Poss:1s><Fam><NOM>
<Num:Sg><Poss:No><Case:Nom>
anne<NOM><Num:Sg><Poss:1s><Fam><NOM>
<Num:Sg><Poss:No><Case:Nom><PRED><Cpl:Aor><Prsn:3s>

(6) Analyses of “annemler” by MorTur

anne<NOM><Num:Sg><Poss:1s><Case:Nom>
<PRED><Cpl:Aor><Prsn:3p>
anne<NOM><Num:Sg><Poss:1s><Fam><NOM>
<Num:Sg><Poss:No><Case:Nom>
anne<NOM><Num:Sg><Poss:1s><Fam><NOM>
<Num:Sg><Poss:No><Case:Nom><PRED><Cpl:Aor><Prsn:3s>

Another problem is absence of Probability suffix. In the section 4.3.2 we explained the Ability and Probability suffixes. Analyses of these analysers don’t contain Probability suffix. In TrMorph, Probability suffix is not separated from Ability suffix. In the ITU NLP analyser, any suffix that corresponds to the Probability

suffix does not exist. Following examples shows analyses of “gelmeyebilirim” and “gelemeyebilirim” in both two analysers.

(7) Analyses of “gelmeyebilirim” in TrMorph

gel<V><neg><abil><V><aor><1s>
gel<V><neg><abil><V><aor><Adj><0><N><0><V>
<cpl:pres><1s>
gel<V><neg><abil><V><aor><Adj><0><N><p1s>
gel<V><neg><abil><V><aor><Adj><0><N><p1s><0><V>
gel<V><neg><abil><V><aor><Adj><0><N><p1s><0><V>
<cpl:pres><3p>
gel<V><neg><abil><V><aor><Adj><0><N><p1s><0><V>
<cpl:pres><3s>
gel<V><neg><abil><V><aor><Adj><0><V><cpl:pres><1s>
gel<V><neg><abil><V><aor><Adj><p1s><Prn>
gel<V><neg><abil><V><aor><Adj><p1s><Prn><0><V>
gel<V><neg><abil><V><aor><Adj><p1s><Prn><0><V>
<cpl:pres><3p>
gel<V><neg><abil><V><aor><Adj><p1s><Prn><0><V>
<cpl:pres><3s>

(8) Analyses of “gelemeyebilirim” in TrMorph

gel<V><abil><V><neg><abil><V><aor><Adj><0><N>
<0><V><cpl:pres><1s>
gel<V><abil><V><neg><abil><V><aor><Adj><0><N>
<p1s>
gel<V><abil><V><neg><abil><V><aor><Adj><0><N>
<p1s><0><V>
gel<V><abil><V><neg><abil><V><aor><Adj><0><N>
<p1s><0><V><cpl:pres><3p>
gel<V><abil><V><neg><abil><V><aor><Adj><0><N>
<p1s><0><V><cpl:pres><3s>
gel<V><abil><V><neg><abil><V><aor><Adj><0><V>
<cpl:pres><1s>
gel<V><abil><V><neg><abil><V><aor><Adj><p1s><Prn>
gel<V><abil><V><neg><abil><V><aor><Adj><p1s><Prn>
<0><V>

gel<V><abil><V><neg><abil><V><aor><Adj><p1s><Prn><0><V><cpl:pres><3p>
gel<V><abil><V><neg><abil><V><aor><Adj><p1s><Prn><0><V><cpl:pres><3s>
gel<V><abil><V><neg><abil><V><aor><1s>

(9) Analyses of "gelmeyebilirim" in ITU NLP
gel+Verb+Able+Neg+Aor+A1sg

(10) Analyses of "gelemeyebilirim" in ITU NLP
gel+Verb+Able+Neg+Aor+A1sg

In our implementation, despite having same surface form, Probability suffix must be separated from Possibility because of their different meaning and different locations in the structure. In our implementation, we considered Probability suffix in all ways. Following examples shows the analyses of "gelmeyebilirim" and "gelemeyebilirim" by MorTur.

(11) Analyses of "gelmeyebilirim" in MorTur
gel<VERB><Actv><Pol:Neg><Prbl><Tns:Aor><Prsn:1s>

(12) Analyses of "gelemeyebilirim" in MorTur
gel<VERB><Actv><Abil><Pol:Neg><Prbl><Tns:Aor><Prsn:1s>

Chapter 8

Conclusion

In this paper we present an approach and an implementation for morphological analysis of Turkish language. The approach is based on Finite State Transducers and two-level transformation. We used HFST for the implementation of the analyser. Other publicly available analysers have linguistic problems. We implement our analyser to be free from these problems so there can be an Turkish Morphological Analyser that is complete and publicly available. As result, an analyser is implemented. To make it more convenient to use, a web site is designed and published. Now it is publicly available on http://ddil.isikun.edu.tr/mortur. As future work, we plan to provide the analyser as pluggable library to be used in any desired application.

References

[1] K. Lindén, E. Axelson, S. Hardwick, M. Silfverberg, and T. Pirinen, "HFST–framework for compiling and applying morphologies," pp. 67–85, 2011.

[2] K. R. Beesley and L. Karttunen, *Finite State Morphology*. Stanford, CA: CSLI Publications,, 2003.

[3] D. Karp, Y. Schabes, M. Zaidel, and D. Egedi, "A freely available wide coverage morphological analyzer for english," in *Proceedings of the 14th conference on Computational linguistics-Volume 3*. Association for Computational Linguistics, 1992, pp. 950–955.

[4] Y. S. Alam, "A two-level morphological analysis of japanese," in *Texas Linguistic Forum*, vol. 22, no. 229-252, 1983, p. 14Although.

[5] L. Karttunen, R. M. Kaplan, and A. Zaenen, "Two-level morphology with composition," in *Proceedings of the 14th conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, 1992, pp. 141–148.

[6] A. C. Tantug, E. Adali, and K. Oflazer, "Computer Analysis of the Turkmen Language Morphology." *FinTAL*, vol. 4139, pp. 186–193, 2006.

[7] G. Kessikbayeva and I. Cicekli, "A Rule Based Morphological Analyzer and a Morphological Disambiguator for Kazakh Language," *Linguistics and Literature Studies*, vol. 4, no. 1, pp. 96–104, 2016.

[8] M. Orhun, A. C. Tantug, and E. Adali, "Rule based analysis of the uyghur nouns." *Int. J. of Asian Lang. Proc.*, vol. 19, no. 1, pp. 33–44, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/jclc/jclc19.html#OrhunTA09

[9] K. Oflazer, "Two-level description of turkish morphology," *Literary and Linguistic Computing,*, vol. 9, no. 2, pp. 137–148, 1994.

[10] c. Çöltekin, "A freely available morphological analyzer for Turkish," in *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC 2010)*, 2010, pp. 820–827. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2010/summaries/109.html

[11] M. Şahin, U. Sulubacak, and G. Eryiğit, "Redefinition of turkish morphology using flag diacritics," in *Proceedings of The Tenth Symposium on Natural Language Processing (SNLP-2013)*, Phuket, Thailand, October 2013.

[12] R. Ehsani, B. Özenç, and E. Solak, "A fst description of noun and verb morphology of azarbaijani turkish," pp. 62–68, 2017. [Online]. Available: http://www.aclweb.org/anthology/W17-4008

[13] R. Ehsani, B. Özenç, and E. Solak, "Moraz: Morphological analyzer for azarbaijani turkish," http://ddil.isikun.edu.tr/moraz, 2017, accessed: 2017-09-30.