A Finite-State Kernel Architecture for Turkish Natural Language Processing

Zelal Güngördü and Kemal Oflazer
Department of Computer Engineering and Information Science
Bilkent University, Bilkent, Ankara, TR-06533, Turkey
{zelal,ko}@cs.bilkent.edu.tr

Abstract:

We present a finite-state kernel architecture for Turkish that performs certain presyntactic processing steps on a given sentence, such as morphological analysis, and recognition of lexicalized and nonlexicalized collocations, followed by morphological disambiguation by voting constraints [7]. The kernel has been implemented using the Xerox Finite State Tools. The approach to recognizing collocations presented here is of particular interest for languages with highly agglutinative morphology (of which Turkish is a very good example), since it only requires a single mechanism to deal with a potentially infinite number of variants of a single collocation in certain cases. Moreover, it may also help resolve morphological ambiguity to some degree.

1 Introduction

Turkish employs a rich collection of collocations, i.e., multi-word constructs that may be considered single syntactic/semantic entities. Recognizing such constructs at a presyntactic level of processing is of interest for both theoretical and practical reasons: It is clear, given the diversity of collocations in Turkish, that dealing with such constructs in syntax would require additional syntactic rules, thereby rendering the grammar rather cumbersome. That further implies that handling those constructs at a presyntactic level would considerably simplify the development of parsers for Turkish. In addition, such functionality may also help resolve morphological ambiguity in cases where one or more of the lexical forms in a collocation has/have various morphological interpretations which are highly unlikely in the context of that particular collocation. Consider, for example, the idiomatic expression in (1), where the second word has actually three ambiguous morphological parses, namely a dative noun, an optative verb, and an adjective. The nominal one is the only plausible parse in the present context so the remaining two can safely be ignored.

(1)

ip-e	sap-a	gel-me-yen	'implausible/unreasonable/illogical'
cord-DAT	1. stem-DAT	come-NEG-PART	
	2. diverge-OPT sapa		
	3. secluded		

We present here a finite-state kernel architecture for Turkish that recognizes such multi-word constructs at a presyntactic stage of processing, and also involves a finite-state implementation of the morphological disambiguation approach by voting constraints, proposed by Oflazer and Tür [7]. The kernel has been implemented using the Xerox Finite State Tools.

Apart from the large number of different collocation forms in Turkish, there is also the fact that certain collocations may, in theory, have an infinite number variants, due to the nature of Turkish morphology. Let us take, for instance, the support verb construct *yürürlüğe koymak 'to put in force'*, and provide only a couple of examples for the possible forms in which this collocation may occur:

(2)

a.	yürürlüğ-e effect-DAT	koy-du-m put-PAST-1SG	'I have put ... in force'
b.	yürürlüğ-e effect-DAT	koy-ma-dık-lar-ım put-NEG-PART-PLU-1sPoss	'those I haven't put in force'
c.	yürürlüğ-e effect-DAT	koy-ul-an-lar-dan put-PASS-PART-PLU-ABL	'from those that have been put in force'

Notice that the first word always occurs as a dative noun, whereas the second one can take any number of inflectional and/or derivational suffixes as an ordinary verb according to the morphotactics of Turkish. The approach presented here enables one to deal with all potential variants of a single collocation like that in a trivially unified way.

The organization of the paper is as follows. After an overview, in Section 2, of different forms of collocations one may observe in Turkish, Section 3 outlines the basics of recognizing collocations using a finite-state framework. Section 4 then presents the kernel architecture we have developed for Turkish, discussing first how different components relate to each other within the system, followed by a description of the functionality of each individual component separately. Finally, Section 5 states our conclusions.

2 Collocations in Turkish

This section provides an overview of different forms of collocations employed by Turkish. One can classify such constructs in Turkish in the following two categories depending on whether or not they occur productively:

1. Lexicalized Collocations: These include adverbial expressions (e.g., *hiç olmazsa 'at least'*, *öte yandan 'on the other hand'*), adjectival constructs (e.g., *açık göz 'open-eyed'*, *saçma sapan 'incongruous'*, *sıkı fıkı 'intimate'*), and support verb constructs, usually with non-compositional lexical semantics, (e.g., *ele geçmek 'to be captured'*, *kazan kaldırmak 'to revolt'*, *yürürlüğe koymak 'to put in force'*).

2. Nonlexicalized Collocations: These include multi-word constructs in one of a number of productive forms, exemplified as follows:

a) adverbial constructs of the following forms: i) (manner adverbials) duplicated third person singular verbal forms in optative mood, e.g., *koşa koşa 'by running'*, ii) (temporal adverbials) third person singular aorist verbal forms with root duplications and sense negation, e.g., *gider gitmez 'as soon as ... go/goes'*; and iii) (again temporal adverbials) duplicated verbal (in simple past tense) and derived adverbial (with the *'-yAlI'* suffix) forms with the same verbal root, e.g., *yaptın yapalı 'since you have done'*.

b) nominal constructs in the form of duplicated derived adverbial (with the *'-yIp'* suffix) and nominal (with, for example, *'-mA'*, *'-dIK'*, *'-yIş'*) forms with the same verbal root, only with sense negation, e.g., *yapıp yapmamak 'whether or not to do'*, *gidip gitmediğim 'whether or not I have gone'*.

c) emphatic adjectival forms involving an adjective duplication with an intervening question clitic, e.g., *küçük mü küçük 'very small'*.

d) manner adverbials formed with adjective or noun duplications, e.g., *güzel güzel 'beautifully'*, *il il 'province by province'*.

It should be noted that this list is by no means meant to be comprehensive, yet it provides quite an extensive set of collocation forms one would observe in Turkish.

3 Recognition of Collocations

This section summarizes a transducer based approach to recognizing collocations. We represent a sentence with n tokens $w_1, w_2, \ldots, w_n$ as an acyclic (identity) transducer with $n+1$ states $s_0, s_1, \ldots, s_n$ – s_n being the final state – where states denote the token boundaries, and a set of transitions labeled $p_{i,1}, p_{i,2}, ..., p_{i,j}$ between a pair of consecutive states s_{i-1} and s_i represent the j ambiguous morphological parses of the ith token w_i in the sentence. Figure 1, for instance, shows the transducer for a sentence with three tokens, where the first token is assumed to have three ambiguous morphological parses ($p_{1,1}, p_{1,2}, p_{1,3}$), the second one only one parse ($p_{2,1}$), and the third one two parses ($p_{3,1}, p_{3,2}$).

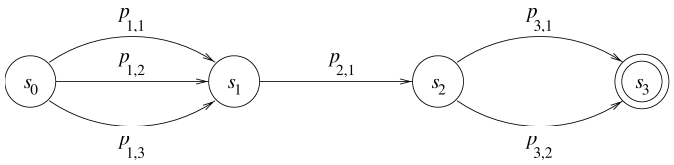

Figure 1: The sentence transducer for a sentence with three tokens which have three, one, and two morphological parses, respectively.

All paths of the transducer are checked against the set of collocations in hand. Each collocation is again represented by a finite-state transducer that updates any matching part of a sentence path by replacing that part by a single transition labeled by the parse assigned to the collocation in question. Parts of a sentence path that do not match a collocation are left unchanged by the transducer for that collocation. Assuming, for instance, that the transition sequence labeled $[p_{1,1}, p_{2,1}]$ between the states s_0 and s_2 in Figure 1 (i.e., the first morphological parse for w_1 followed by the only parse for w_2) matches a collocation represented by the parse p_c, that sequence is replaced by a single transition labeled p_c between the states s_0 and s_2, as shown in Figure 2. Notice that the remaining paths between s_0 and s_2 (i.e., $[p_{1,2}, p_{2,1}]$ and $[p_{1,3}, p_{2,1}]$) are left unchanged in this case. If however one is certain that those remaining paths are highly unlikely in the present context (see, for example, (1) in Section 1), then it is possible to leave them totally out, as in Figure 3, thereby resolving the ambiguity due to the first token w_1 in the sentence.

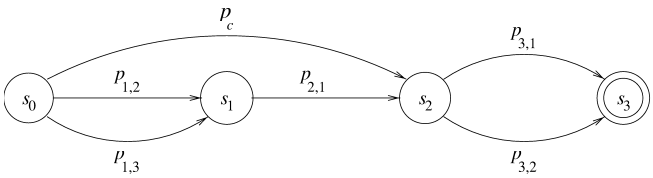

Figure 2: The sentence transducer in Figure 1 once the collocation (with the parse p_c) matching the sequence $[p_{1,1}, p_{2,1}]$ is recognized.

The check to see whether a given collocation occurs in an input sentence is implemented via the composition operation over finite-state transducers, i.e., via composing the finite-state transducer corresponding to the input sentence with the one implementing the replacement operation for the collocation in question. That further enables one, due to the nature of the composition operation, to

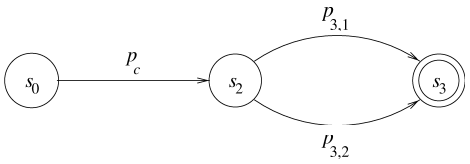

Figure 3: The simplified sentence transducer after unlikely paths in Figure 2 are removed.

compose all collocation transducers *off-line*, thereby obtaining a single collocation transducer, which can then be composed with any given input sentence transducer in just one step.

4 The System Architecture

We have implemented the approach outlined above using the Xerox Finite State Tools (xfst). Figure 4 shows the architecture of our system (which should be considered in a bottom-up fashion), with 'o' denoting the composition operator, and each '→' pointing to the outcome of the composition indicated just below it. When an input transducer for a sequence of words (representing only the lexical items) is composed with the morphological analysis transducer, one gets a sentence transducer as defined in Section 3 (cf. Figure 1), with all possible morphological parses of the words in the input sequence. That transducer is then composed with a collocation transducer, which has been compiled *off-line* as the composition of all collocation transducers (for both lexicalized and nonlexicalized collocations), also including a vote initialization transducer inserting a vote component, initialized to 0, in each transition label of the sentence transducer once all collocation checks are completed. Thus, the output is a modified version of the initial sentence transducer, possibly with certain transitions replaced by transitions corresponding to any collocations recognized in the input sentence, and also with an initial vote of 0 assigned to every transition label. Finally, that output transducer is composed with a constraint transducer, which again has been compiled *off-line* as the composition of all morphological disambiguation constraint transducers, that updates the transition votes corresponding to certain morphological parses in the sentence transducer according to the individual voting constraint rules involved in the composition.

Having provided the reader with the general picture, let us now discuss each individual component in Figure 4 in more detail.

Morphological Analysis Transducer (MAT)

That component is a transducer corresponding to the regular expression in (3),[1] where `M` is assumed to correspond to a transducer that maps a given token to all its possible morphological parses, and `0`, `.x.` and `+` denote the empty string (epsilon), and the crossproduct and iteration operators, respectively.

(3) `[ M [ 0 .x. [" "]+ ] ]+`

At this point tokenization is not handled separately and tokens are assumed to be separated by white space characters. The MAT essentially maps a sequence of tokens each followed by any number of token boundary spaces to all sequences (combinations) of possible morphological parses of the tokens in the input (once the composition with the input sentence takes place), also getting rid of the boundary spaces in the output.

[1] We present our regular expressions using the Xerox regular expression language (for further information on that language see http://www.xrce.xerox.com/research/mltt/fst/home.html).

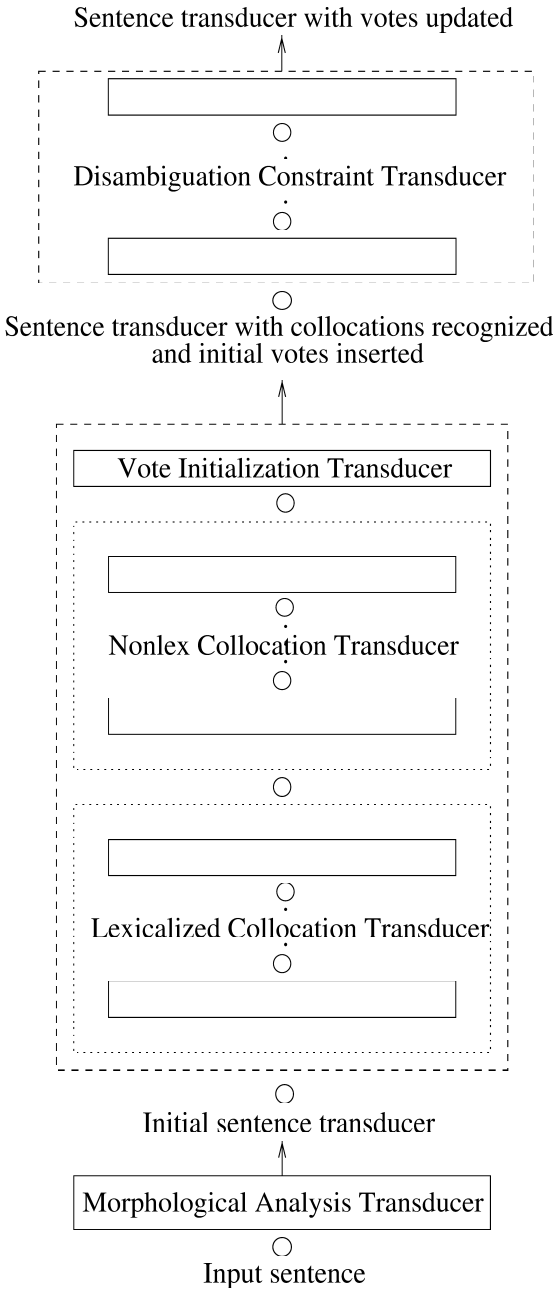

Figure 4: The system architecture of the finite-state kernel for Turkish

Currently the MAT module comprises a single transducer for performing the morphological analysis of Turkish words. This full scale analyzer [5] implemented using Xerox *twolc* and *lexc* tools ([3], [1]), maps a Turkish word into all its possible morphological interpretations represented as sequential feature-value encoding all derivational and inflectional information. For instance, the word *koyun* is analyzed as,

```
1. koyu+[u]n  (your dark (thing))
   [[CAT=ADJ][ROOT=koyu][CONV=NOUN=NONE][AGR=3SG][POSS=2SG][CASE=NOM]]
2. koyun   (sheep)
   [[CAT=NOUN][ROOT=koyun][AGR=3SG][POSS=NONE][CASE=NOM]]
3. koy+[n]un  (of the bay)
   [[CAT=NOUN][ROOT=koy][AGR=3SG][POSS=NONE][CASE=GEN]]
4. koy+un     (your bay)
   [[CAT=NOUN][ROOT=koy][AGR=3SG][POSS=2SG][CASE=NOM]]
5. koy+[y]un  (put!)
   [[CAT=VERB][ROOT=koy][SENSE=POS][TAM1=IMP][AGR=2PL]]
```

while *geldiğimdekiler* (those things (present) at the time I came) would be analyzed as

```
1. [[CAT=VERB][ROOT=gel][SENSE=POS]
    [CONV=NOUN=DIK][AGR=3SG][POSS=1SG][CASE=LOC]
    [CONV=DET]
    [CONV=NOUN=NONE][AGR=3PL][POSS=NONE][CASE=NOM]]
```

We have also built a second analyzer based on the first one, but with only a single noun root lexicon which matches an arbitrary sequence of Turkish characters. This analyzer will hypothesize interpretations of unknown character sequences subject to Turkish morphographemic and morphotactic constraints. Our previous work with this approach has shown that it is very effective for unknown words in Turkish [6]. But, this second analyzer leaves out an important class of unknown words, those with a foreign language root word written using its orthography in the foreign language, but then inflected using Turkish suffixation based on it *pronunciation*, (e.g., Bordeaux'dakiler, where the French root word ends with a vowel in pronunciation, but not in orthography.) Such use is prevalent in technical literature, e.g., in computer technology newspapers. A morphological analyzer dealing with this class of unknown words, using a unified pronunciation encoding of foreign (mainly English and French) words, is under development. Normally, for proper morphological analysis of tokens occurring in Turkish text, one should use a transducer that combines all three analyzers above. However taking the union of these transducers is not practical, since the input language of the latter two would have to be constrained to prevent overgeneration, and such constraining leads to very large transducers. Thus, one should use these transducers at run time using a simple scheme whereby the unknown word transducer is consulted only if the morphological analyzer fails, and the foreign word analyzer is consulted if the former two fail. Composition in this context would then be simulated by transducer lookups followed by concatenations. In the current system, we only have the morphological analyzer integrated and have postponed the integration of the other two to a later phase.

Lexicalized Collocation Transducer (LCT)

The LCT is a transducer which has been compiled as the composition of a number of transducers each handling an individual lexicalized collocation. Let us consider, as an example, the collocation in (4), an adverbial construct made up of the pronoun *ne* followed by the postposition *denli*.

(4)	ne	denli	'to what degree'
	what	to ... degree	

(4) is dealt with by the transducer corresponding to the following regular expression:

(5)
```
"[" "[CAT=ADVERB] "[ROOT="  n e %- d e n l i "]" "]"
        <-
 [
"[" "[CAT=PRONOUN]" "[ROOT=" n e "]" "[TYPE=QUES]" "[AGR=3SG]" "[POSS=NONE]" "[CASE=NOM]" "]"
"[" "[CAT=ADVERB]   "[ROOT=" d e n l i "]" "]"
];
```

(5) uses the replace operator `<-` (Karttunen [2]) to replace any instance of a sequence of morphological parses in the lower language belonging to the two tokens making up the collocation in (4) (i.e., the part below the `<-`) by the parse assigned to that collocation as a whole, in the upper language (i.e., the part over the `<-`).

It should be noted that the first word *ne* in (4) has actually two further parses (in addition to the pronominal parse contributing to the collocation in (4)), namely an adjectival parse and an adverbial one, both of which can safely be ignored in the context of that particular collocation. As discussed in Section 3, to achieve that one needs to cut off the paths in a given sentence transducer containing any transitions labeled by either of those two parses followed by a transition labeled by the parse for the second word in (4). We implement this idea using a transducer which corresponds to the regular expression in (6), where `*` and `|` denote the Kleene star and union operators, respectively, and $\sim$ denotes complementation.

(6)
```
~[ FEAT-ST*
  "[" [ "[CAT=ADJ]" "][ROOT=" n e  |
        "[CAT=ADVERB]" "[ROOT=" n e  ] "]" "]"
  "[" "[CAT=ADVERB]" "[ROOT=" d e n l i "]" "]"
   FEAT-ST* ]
```

Here `FEAT-ST` denotes the regular expression `"[" CAT-F "[ROOT=" S+ "]" REST-F* "]"` where `CAT-F` is the union of all possible syntactic category values in Turkish, and S is the union of all Turkish characters. `REST-F` denotes the union of all features occurring in morphological parses in Turkish except for the category and root features. This transducer when composed with the sentence transducer after the addition of the lexicalized collocation arc, will remove the spurious parses due to the adjectival and adverbial readings of *ne*.

Note that the collocation in (4) is rather easy to handle, since it only occurs in the language in that particular form. However, as we have mentioned in Section 1, there are certain other lexicalized collocations in Turkish where the last word of the collocation may, in theory, have an infinite number forms, with any number of inflectional and/or derivational suffixes affixed to its root according to the morphotactics of Turkish. Examples include the adjectival and verbal lexicalized collocations mentioned in Section 2. Let us consider here the support verb construct *yürürlüğe koymak 'to put in force'*, for which we have provided a number of possible variants earlier in Section 1. It turns out one needs a slightly more complicated mechanism than (5) to deal with such collocations. For instance, all three examples in (2) (and a potentially infinite number of other variants of the same collocation) are recognized by the transducer corresponding to the following regular expression (`FEAT-ST` and `REST-F` here denote the same regular expressions defined above):

(7)
```
{ [] <- %{ , [] <- %} }
  .o.
[
  FEAT-ST
 |
  [
    %{
      [ 0 .x. [ "[" "[CAT=NOUN]" "[ROOT=" y U r U r l U k  "]" "[AGR=3SG]"
                  "[POSS=NONE]" "[CASE=DAT]"  "]"  ] ]
      [ [ "[" "[CAT=VERB]" "[ROOT=" y U r U r l U G e %- k o y ]
          .x.
        [ "[" "[CAT=VERB]" "[ROOT=" k o y ]                    ] "]" REST-F* "]"
    %}
  ]
]*
  .o.
[ %{ ... %}  <-@
  [
    "[" "[CAT=NOUN]" "[ROOT=" y U r U r l U k  "]" "[AGR=3SG]" "[POSS=NONE]" "[CASE=DAT]" "]"
    "[" "[CAT=VERB]" "[ROOT=" k o y "]"  REST-F*  "]"
  ]
];
```

This transducer is actually the composition of three separate transducers with the following functions:

1. The bottom transducer makes use of the longest match bracket operator `<-@` of Karttunen [2] to bracket (using { and }), in the upper language, any sequence of transitions in the lower language labeled by the morphological parse of the first word of the collocation followed by a parse with the same verbal root of the second word, which is followed by `REST-F*`. Effectively, the second word of a sequence bracketed by the bottom transducer will have the verbal root *koy 'put'* with any number of inflectional and/or derivational features following that root.

2. The middle transducer then either passes through (without any effect) the unbracketed sections of the input (indicated by the first part of the disjunct, `FEAT-ST`), or replaces any instance of the collocation in the lower language (within the brackets { and }) by the parse assigned to that collocation, in the upper language, in the following way: First, the morphological parse for the first word in the collocation is totally left out in the upper language. Then, the verbal root *koy 'put'* is replaced by the verbal root *yürürlüğe koy 'put in force'*. And finally, any other morphological features following the root *koy* in the input are passed through unchanged (indicated by `REST-F*`), thereby preserving, in the output, any vital inflectional and/or derivational information concerning the second word of the collocation.

3. In the end, the top transducer simply gets rid of any brackets in the lower language using the parallel replacement operator of Karttunen [2].

Nonlexicalized Collocation Transducer (NCT)

The NCT is compiled as the composition of six separate transducers each dealing with one of the nonlexicalized collocation specifications mentioned in Section 2 (with one exception as discussed below). The four transducers for the three cases in (a) and the one in (b) are rather similar, and exemplified below in (9) by the one for the (aii) case, that is, temporal adverbials in the form of third person singular aorist verbal forms with root duplications and sense negation, as in the following construct:

(8)

gel-ir	gel-me-z	'as soon as ... come/comes'
come-AOR	come-NEG-AOR	

(9)
```
{ [] <- %{ , [] <- %} }
  .o.
[
  FEAT-ST
 |
  [
    %{
       "["  [ "[CAT=ADVERB]" .x. CAT-F ]  "[ROOT=" S+ "]" REST-F*
            [ 0 .x.  [ "[SENSE=POS]" "[TAM1=AOR]" "[AGR=3SG]" "]"
                       "[" CAT-F "[ROOT=" S+ "]" REST-F*  "[SENSE=NEG]" "[TAM1=AOR]" "[AGR=3SG]" ]
            [ [ "[TYPE=TEMP]"  "[M-TYPE=AS-SOON-AS]" ] .x. 0 ]  "]"
    %}
  ]
]*
  .o.
[
  %{ ... %}  <-@
  [
    "[" CAT-F  "[ROOT=" S+ "]"  REST-F*  "[SENSE=POS]"  "[TAM1=AOR]"  "[AGR=3SG]"  "]"
    "[" CAT-F  "[ROOT=" S+ "]"  REST-F*  "[SENSE=NEG]"  "[TAM1=AOR]"  "[AGR=3SG]"  "]"
  ]
];
```

The transducer corresponding to (9) is again the composition of three separate transducers, just as in the case of (7) above, working in the following way:

1. The bottom transducer brackets any portion of the input matching the collocation in hand, that is two consecutive verbal parses with sense negation, but the same mood and agreement (optative and third person singular, respectively). Notice that we do not actually check whether those two parses indeed have the same root, since such a check could not be done using finite-state techniques.[2] Considering that there is no other case in Turkish where two words with the above-mentioned morphological features may occur consecutively (at least not without a comma in between, and even then it is highly unlikely), we assume the roots would simply be the same in such a case. Note also that the roots are not constrained to be verbal (hence `CAT-F` rather than a verbal category constraint, `"[CAT=VERB]"`), since those two words may well be derived verbal forms, as in (10), where the verb *taşlaş* has been derived from the nominal root *taş* with the suffix *'-laş'*:

   (10)

taş-laş-ır	taş-laş-ma-z	'as soon as ... become/becomes of stone'
stone-CONV-AOR	stone-CONV-NEG-AOR	

   That the two words in a sequence bracketed by this trandsucer are verbal is guaranteed by the fact that their parses are constrained to end with the tense/aspect/mood and agreement features.

2. The middle transducer either passes through the unbracketed sections of the input, or replaces any instance of the collocation in the lower language by the parse assigned to that collocation, in the upper language, in the following way: The category value of the first word is replaced by an adverbial category value, and the root is passed through to the upper language. After that, any features up to the last three (i.e., the sense, tense/aspect/mood and agreement features) are again passed through, but the last three are simply ignored. Note that the agreement of both forms in such a construct will always be third person singular, independent of the agreement of the subject the construct takes, hence can simply be ignored here.[3] On the other hand, any voice

[2] It would require recognizing strings of the form `wx wy`, which is not context-free.

[3] However, in the case of the temporal adverbials mentioned in (aiii) in Section 2, for example, the agreement of the first verb is essential for the subject-verb agreement (to be checked by the syntactic processor), and hence is passed through to the upper language in the transducer dealing with that class of collocations.

features (coming before the sense feature) could be vital for the syntactic processor to determine the valence features of the construct as a whole,[4] and so should be passed through to the upper language, as is the case in (9). Consider, for instance, the three collocations in (11), all with the same transitive verbal root *yap*. The transitive root in (11a) is not followed by any voice suffixes so the verb has (default) active voice. The same root in (11b) is followed by a passive suffix, thereby resulting in an intransitive verb. And in (11c) it is followed by a causative suffix, resulting in a ditransitive verb.

(11) a. yap-ar yap-ma-z 'as soon as ... do/does'
do-AOR do-NEG-AOR

b. yap-ıl-ır yap-ıl-ma-z 'as soon as ... have/has been done'
do-PASS-AOR do-PASS-NEG-AOR

c. yap-tır-ır yap-tır-ma-z 'as soon as ... have/has ... do'
do-CAUS-AOR do-CAUS-NEG-AOR

In the end, the middle transducer in (9) also includes, in the parse for the collocation, a temporal type feature and a minor type feature conveying the meaning of the collocation.

3. Finally, the top transducer gets rid of any brackets in the lower language using the parallel replacement operator.

The transducer for the emphatic adjectival forms involving an adjective duplication with an intervening question clitic (cf. item (c) in Section 2) works in a similar fashion, only this time searching the input for sequences of two adjectives with an intervening question clitic. Again, we do not check whether the adjectives in such a constuct indeed have the same root. Yet we do make sure that the second adjective is not followed by another question clitic, to avoid misidentifying coordinating interrogative structures missing a comma (although it should in fact be there) as instances of this collocation.

Finally, coming to the last item (d) in Section 2, adjective duplications are handled in a way similar to the lexicalized collocation (4) on page 6. There is one transducer (similar to the one corresponding to the regular expression in (5)) for every adjective that may actually occur in such a collocation, searching the input for any duplications of that adjective. Note however that doing the same thing for noun duplications would not be that practical, since the noun lexicon is considerably larger in size. Therefore, manner adverbials formed with noun duplications are not handled by our system.

Disambiguation Constraint Transducer (DCT)

The DCT has been compiled as the composition of a number of separate morphological disambiguation transducers, which mainly encode the voting constraints for morphological disambiguation in Turkish proposed by Oflazer and Tür [7]. The main idea there is that sequences of morphological parses which satisfy various local contextual criteria are recognized and voted on by constraint rules. The disambiguation of parses are performed at the very end after all constraint rules have been applied, by finding the path with the largest tallied vote. Oflazer and Tür [8] provide a finite state transducer implementation of this approach for English. Here we apply the same approach to Turkish. For instance, in the phrase below, the first and the last tokens have other readings (total 2 and 4, respectively), but since the parses given in (12) make up a possessive noun phrase, paths on which these parses both exist are voted with a high vote.

(12) bölüm-ün (yeni) öğrenci-ler-i 'the department's (new) students'
department-GEN new student-PLU-3sPoss

[4] Knecht ([4]) defines passivization/causativization in Turkish as an operation that decreases/increases the valence of a verb by one with the affixation of a passive/causative suffix to that verb.

Voting is implemented by introducing a vote field to each arc label and then manipulating it by transducers, one transducer for each voting constraint rule. For example, the regular expression in (13) increments by 100 the votes of all parses taking part in a possessive noun phrase such as the one in (12). Here FEAT-ST, CAT-F, S, and REST-F below denote the same regular expressions defined earlier. POSS-F is defined as the union of all possessive feature values. CONVADJ-F is the union of all adjective derivation features, and TYPEADJ-F is the union of all subtype feature values that may be assigned to an adjective. AGR-F and CASE-F denote the union of all agreement and case features, respectively. And finally, VOTE denotes a regular expression of the form "<" ["+" | "-"] DIGIT+ ">" with DIGIT being the union of all decimal digit symbols.

(13)
```
{ [] <- %{ , [] <- %} }
  .o.
[
  [ "[" FEAT-ST  VOTE  "]" ]
 |
  [
    %{
       [ "[" FEAT-ST  ADD100 "]" ]+
    %}
  ]
]*
  .o.
[
  %{ ... %}  <-@
  [
    "[" "[" CAT-F  "[ROOT=" S+ "]"  REST-F*  "[AGR=3SG]"  POSS-F  "[CASE=GEN]"  "]"
         VOTE "]"
    [ "[" "[" [ [ "[CAT=ADJ]" "[ROOT=" S+ "]" ] |
                [  CAT-F     "[ROOT=" S+ "]" REST-F*  CONVADJ-F ] ]
              (TYPEADJ-F) "]"   VOTE "]" ]*
    "[" "[" CAT-F  "[ROOT=" S+ "]"  REST-F*  AGR-F  "[POSS=3SG]"  CASE-F  "]"
         VOTE "]"
  ]
]
  .o.
[ "[" FEAT-ST  VOTE "]" ]*
```

The transducer corresponding to (13) is the composition of four different transducers:

1. The bottom transducer constrains the input to sequences of a morphological parse followed by a vote component, bracketed by [and].

2. The second one from the bottom brackets in the input any sequences of a third person singular noun with genitive case followed by a noun bearing the third person singular possessive suffix and any number of intervening (simple or derived) adjectives between those two nouns.

3. The third one then either passes through the unbracketed sections of the input, or increments by 100 the vote fields of all the words within a pair of brackets. Here ADD100 is a simple transducer that manipulates the vote field digits so that the upper side vote field represents a number that is 100 greater than the lower side number.

4. Finally, the top transducer gets rid of any brackets in the lower language using the parallel replacement operator.

5 Conclusions

We have presented the highlights of a finite state kernel architecture for use in Turkish natural language processing applications. The kernel handles morphological analysis, coalescing of lexicalized and nonlexicalized collocations and imposition of local contextual constraints for disambiguation. Our approach to handling collocations enables us to capture all morphological variants of a single collocation in a trivially unified fashion, which may also be of interest for other languages with similar morphological and lexical phenomena. Despite the fact that most nonlexicalized collocation patterns in Turkish require non-finite state mechanisms (some requiring power even beyond context-free) for recognition, we use certain simplifying assumptions in a cautious way that enable us to deal with such cases using finite state techniques.

6 Acknowledgements

We thank Lauri Karttunen of Xerox Research Centre Europe, for providing us with the Xerox Finite State Tools. This research was also supported by a NATO Science for Stability Project Grant, TU-LANGUAGE.

References

[1] Lauri Karttunen,. "Finite-state lexicon compiler,". Technical Report, XEROX Palo Alto Research Center, April 1993.

[2] Lauri Karttunen, "Directed replacement," in *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, 1996.

[3] Lauri Karttunen and Kenneth. R. Beesley,. "Two-level rule compiler,". Technical Report, XEROX Palo Alto Research Center, 1992.

[4] Laura E. Knecht,. *Subject and Object in Turkish.* PhD thesis, Department of Linguistics and Philosophy, Massachusetts Institute of Technology, Cambridge, Mass., 1986.

[5] Kemal Oflazer, "Two-level description of Turkish morphology," in *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics (A full version appears in* Literary and Linguistic Computing, *Vol.9 No.2, 1994.)*, Utrecht, The Netherlands, April 1993.

[6] Kemal Oflazer and Gökhan Tür, "Combining hand-crafted rules and unsupervised learning in constraint-based morphological disambiguation," in Eric Brill and Kenneth Church, eds., *Proceedings of the ACL-SIGDAT Conference on Empirical Methods in Natural Language Processing*, 1996.

[7] Kemal Oflazer and Gökhan Tür, "Morphological disambiguation by voting constraints," in *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, 1997.

[8] Kemal Oflazer and Gökhan Tür,. "Implementing voting constraints with finite state transucers,". Submitted to FSMNLP'98, January 1998.