Developing Morph-Analyzer for Urdu Using Apertium: Some issues

Shahid Mushataq Bhat

Shahid.bhat3@gmail.com

Linguistic Data Consortium for Indian Languages (LDCIL)

CIIL, Mysore

Content:

- Introduction
- *Morphological features of Urdu*
- Apertium (LT-toolbox): Some background
- Computing Noun-morphology using LT Toolbox
- Split-Orthography of Urdu
- Conclusion

Introduction:

- Automatic morphological analysis is the fundamental task in NLP that can be employed in enhancing the accuracy of POS-taggers, Chunkers, Parsers and Information retrieval systems.
- Computational morphology models the internal structure of words i-e the way; words are built out of minimal units called morphemes.
- Most of natural languages construct words by concatenating morphemes together in strict orders. Such Concatenative morphotactics is highly productive, particularly, in agglutinative languages like Tamil, Kannada, Manipuri, etc but in some languages like Hebrew and Arabic (Semitic languages) infixation is the main morphological operation (instead of concatenation), constituting Non-Concatenative (Templatic or Root & Pattern) morphology.

Continues ……

- Beyond this Concatenative and Non-Concatenative polarity, Urdu nouns (unlike nouns of other Indian Languages) show the interplay of both types of morphologies. So, morphological structure of Urdu like Tagalog (a language of Philippines) can't be computed adequately unless dual nature of its morphology is not taken into account.
- "The morphotactic limitations of the traditional implementations are the direct result of relying solely on the Concatenative operations in morphological descriptions" (Beesley & Karttunen)

Continues ……

- Corpus based analysis of about 5000 words and native speakers intuitions have revealed the fact that about 50% Noun-paradigms (inflectional) belong to Non-Concatinative morphology but about 50% of Noun-paradigms (Inflexional), almost all Inflexional Verb-morphology and whole derivational morphology of Urdu is concatinative in nature and can be properly handled by LT-Toolbox, though derivational-morphology poses tokenization problems due to split-orthography.

Continues ……

- LT Toolbox is lacking features for dealing with stem-internal variation, diphthong simplification, and compounding (F.M.Tyers, L. Wiechetek & T.Trosterud, 2009).
- Therefore, Morphological analyzer of Hindi like other Indian languages can be easily developed by using LT-toolbox, as their morphology is purely concatinative in nature but Morphological analyzer of Urdu can't be developed easily due to the non-concatinative nature of its many noun paradigms and the split-othography (tokenization problem).
- Such deviation from the nature of other Indian languages is due to the fact that Urdu is replete with the borrowings of Persio-Arabic loan words.

Morphological features of Urdu

- Urdu belongs to the Indo-Iranian branch of the Indo-European Language Family. It is morphologically rich language with highly productive inflectional and derivational morphology.
- Urdu nouns show agreement for number, gender and case. They also show diminutive and vocative affixation. Moreover, the nouns show derivational changes into adjectives and nouns.
- Adjectives show similar agreement changes for number, gender and case.
- Urdu verbs inflect to show agreement for number, gender, honorificity and case. In addition to these factors, it also has different inflections for infinitive, past, non-past, habitual and imperative forms. All these forms (about 20 in total) for a regular verb are duplicated for transitive and causative forms, thus giving a total of more than 60 inflected variations.

Continues.....

- The paradigms which are common to Urdu-Hindi like “laD’kA, laD’ki, kitAba, etc” can be easily defined in LT Toolbox but Non-Concatinative Paradigms (NCPs) where words show stem-internal variation can’t be defined as such.
- Nouns belonging to NCPs don’t show any variation when oblique forms are to be derived from the direct ones. Therefore, in NCPs there is no marked difference between direct and oblique forms (e.g. sg.dir = sg.obl and pl.dir = pl.obl).
- However, many Nouns can produce plural forms both concatenatively as well as non-concatenatively. e.g. “shaqala” has both “shakalEN” as well as “AshqAla” its plural forms.

Continues.....

- waqata
  awqAta
- farada
  afrAda
- qsama
  aqsAma
- qadama
  iqdAma
- madada
  imdAda
- maraza
  imrAza
- mazmUn
  mazAmIn
- khatUn
  khawAtIn
- shetAn
  sheyAtIn
- mazhab
  mazAhib
- sharat
  sharAit

Apertium or LT-toolbox: Some background

- Apertium or LT Toolbox is an open-source platform for creating rule-based machine translation (MT) system (ArmentanoOller et al. 2006; CorbíBellot et al. 2005). It was developed through a number of projects like “Open-Source Machine Translation for the Languages of Spain” and “EurOpenTrad: Open-Source Advanced Machine Translation for the European Integration of the Languages of Spain” by the funding of Spanish Ministry of Industry, Commerce and Tourism.
- It was initially designed for closely-related Romance languages pairs (such as Spanish-Catalan, Spanish–Galician, Spanish–Portuguese, Czech–Slovak, Swedish–Danish etc.), but has also been adapted to work better for less related languages.

Continues……

- It consists of following series of pipelined lexical processing modules: *Deformatter*, *Morphological analyzer*, *Categorical disambiguater*, *Structural and Lexical Transfer module*, *Morphological generator*, *Post generator* and Reformatter.
- It was decided that LT Toolbox can be used to develop morphological analyzers for Indian languages including those languages that have heavily borrowed from Arabic and Persian (e.g. Urdu, Kashmiri, etc). The only effort that has to be made is to add language specific data to the readymade tool.

Continues……

- Since, the tool will support Roman characters only; a transliteration scheme has to be followed that transliterates the scripts of Indian languages into Roman script, a well defined Transliteration Scheme is prerequisite for computing morphology using LT Toolkit .
- But it should preserve the writing conventions like intra-word spacing in multi-Token words .
- Using Transliteration to handle split-orthography (Tokenization Problem) would be simply evading the problem rather than solving it.

Computing Noun-morphology using LT Toolbox

- Computing noun-morphology of Urdu using LT Toolbox is in itself an incomplete task as only concatenative morphology seems to be computable using this tool and the techniques presently in hand. However, the process involves the following steps:

Step-1. All the symbols **<sdefs>** (Characters, Categories, Subcategories, Attributes and Attribute values) that will be used in the scheme are to be defined in the in the XML code of Linux compatible LT Toolbox as shown below:

Continues.....

```
<?xml version="1.0"?
<dictionary>

<alphabet>abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-
    '</alphabet>
<sdefs>
<sdef n="cat:N" c="Noun"/>
<sdef n="subcat:NC" c="Common noun"/>
<sdef n="subcat:NP" c="Proper noun"/>
<sdef n="subcat:NV" c="Verbal noun"/>
<sdef n="subcat:NST" c="Spatio-Temporal noun"/>
<sdef n="gender:m" c="masculine"/>
<sdef n="gender:f" c="feminine"/>
<sdef n="gender:o" c="feminine"/>
<sdef n="number:sg" c="singular"/>
<sdef n="number:pl" c="plural"/>
<sdef n="case:d" c="direct"/>
<sdef n="case:o" c="oblique"/>
</sdefs>
```

Continues……

Step-2. All the possible paradigms (<**pardefs**>) of a language are to be defined as shown below. But only the Concatenative morphological paradigms can be defined. The non-Concatenative paradigms can't be defined which is a main bottleneck of using LT Toolbox.

- In this step, a transliterated word is splitted into the left-unchangeable string (base or lemma) and the right-changeable string, separated by a slash. The changeable string which is on right side of the slash (splitter) gets substituted while generating the all possible forms of the same word as shown below:

Continues.....

<pardefs>
<pardef n="ladk/A__nm">
<e> <l>A</l> <r>A<s n="cat:N"/><s n="subcat:NC"/><s n="gender:m"/><s n="number:sg"/><s n="case:d"/></r> </e>
<e> <l>E</l> <r>A<s n="cat:N"/><s n="subcat:NC"/><s n="gender:m"/><s n="number:sg"/><s n="case:o"/></r> </e>
<e> <l>E</l> <r>A<s n="cat:N"/><s n="subcat:NC"/><s n="gender:m"/><s n="number:pl"/><s n="case:d"/></r> </e>
<e> <l>OM</l> <r>A<s n="cat:N"/><s n="subcat:NC"/><s n="gender:m"/><s n="number:pl"/><s n="case:o"/></r> </e>
</pardef>

- <pardef n="baCC/ah__nm">
- <pardef n="ladk/I__nf">
- <pardef n="AsmAn/a__nm">
- <pardef n="Upar/a__no">
- <pardef n="kh'ayAl/a__nm">
- <pardef n="rAt/a__nf">

Continues.....

- Step-3. In this step lemmatization has to be done manually and computationally feasible (compromising linguistic feasibility) lemmas are to be entered in the dictionary along with the paradigm name (already defined), they follow.
- Sufficient number of dictionary entries is to be made in order to assure wider coverage of the morphological analyzer/generator.
- Computationally feasible lemmas and their paradigm name assignment is show below in the XML coded dictionary.

Continues.....

```
</pardefs>
<section id="main" type="standard">
<e lm="ladk"><i>ladk</i><par n="ladk/A__nm"/></e>
<e lm="kut"><i>kut</i><par n="ladk/A__nm"/></e>
<e lm="kang"><i>kang</i><par n="ladk/A__nm"/></e>

<e lm="shikaw"><i>shikaw</i><par n="baCC/ah__nm"/></e>
<e lm="CamaC"><i>CamaC</i><par n="baCC/ah__nm"/></e>
<e lm="baCC"><i>baCC</i><par n="baCC/ah__nm"/></e>

<e lm="ladk"><i>kang</i><par n="ladk/I__nf"/></e>
<e lm="murg"><i>murg</i><par n="ladk/I__nf"/></e>
<e lm="khidk"><i>khidk</i><par n="ladk/I__nf"/></e>

</section>
</dictionary>
```

Split-Othography : A Tokenization problem

- There are some sort of splitting tendencies in the written language in those languages that follow Persio-Arabic script due to which derivational morphemes are written as separate tokens.

For instance, adjectival morphemes are written separately from their bases as given below:

nigAr	e.g. mazmOn –nigAr	مضمون نگار
khAnah	e.g. kutub- khAnah	کتب خانہ
nAk	e.g. khof- nAk	خوف ناک
gAr	e.g. gunah- gAr	گنہ ـگار

Conclusion:

The above discussion reveals that developing a Finite-state-morphological-analyzer for Urdu using LT Toolbox is not as realistic task as it is for Hindi and other Indian languages. Handling infixation of Urdu needs entirely different approach. Since infixation is the property of Semitic languages (like Arabic, Hebrew) that use several tiers to compute morphology. For instance (McCarthy, 1981) uses four tiers, one for prefixes, one for roots, one for template (consonant pattern) and the last one for vocalization (vowel pattern).

Any Question or Comment:

Thank You