Implementing Urdu Grammar as Open Source Software
Extended Abstract

Muhammad Humayoun
Department of Mathematics
University of Savoie, France
mhuma@univ-savoie.fr

Harald Hammarström, Aarne Ranta
Department of Computer Science
Chalmers University of Technology, Sweden
{harald2,aarne}@cs.chalmers.se

Summary

We describe an implementation of Urdu language as a software API, and we deal with orthography, morphology and the extraction of the lexicon. We also present an implementation of a small fragment of Urdu syntax to demonstrate the reusability of our approach.

1. Introduction

Urdu is a challenging language because of, first, its Perso-Arabic script, second, its morphological system having inherent grammatical forms and vocabulary of Arabic, Persian and the native languages of South Asia and third, its pragmatically neutral constituent order (SOV - Subject Object Verb).

Today, the state of art technology to write grammars (morphology + syntax) is to use special-purpose languages based on finite-state technology. These languages are mostly based on regular expressions. In our opinion, these languages are still close to the machine code. Therefore, we emphasis on using a higher level language to capture the linguistic abstraction. Then that higher level code should be translated into finite state code by some tool if required.

2. Orthography and Unicode Infrastructure

We save the morphology component and the lexicon in ASCII characters, because of, (1) It could be viewed and manipulated easily on different platforms, and (2) As Urdu is closely related to Hindi and they both share the grammar (morphology + syntax + almost all phonology), therefore, it could be reused for Hindi in future, just by adding a lexicon and the transliteration scheme for Hindi.

However, to support Unicode Character set, a clear, strict and reversible transliteration scheme is defined both for the letters and the diacritic marks of Urdu script. The orthography component and Unicode infrastructure is implemented in Java by using a package ICU4J [3]. It accommodates Perso-Arabic script of Urdu as well as provides a GUI application and useful tools for the project.

3. Morphology and Lexicon

The morphology component is implemented in Functional Morphology (FM henceforth) [1], which is a toolkit for morphology development in Haskell. This component contains, first, a type system that covers the language abstraction of Urdu completely and second, an inflection engine that covers word-and-paradigm morphological rules for Urdu for all word classes. However, we do not deal with the words that would require determining across multi-token units (some loan derivational affixes), hence leaving them to be dealt at the syntax level.

A wide-coverage lexicon is a key part of any morphological implementation. Therefore, a corpus is built from the literature and news domain and the lexicon is automatically extracted from it by using a tool *extract* [2]. Later we manually re-check the extracted lexicon to make sure its correctness. However, we do not apply the missing diacritics on partly vocalized words. We show the results of the extraction experiment below:

Table 1: Results

	Total Words	Words with Diacritic
Corpus	1,520,000	23,696
Unique	63,700	6,633
Extracted lexicon	9,126	632
Clean lexicon	4,816	415

As shown above, the clean lexicon is approximately half (52.8%) of the extracted lexicon. We find that the incorrect entries are mostly due to, first, the spelling mistakes; second, the lack of spaces between words or extra spaces inside words; and third, the use of foreign

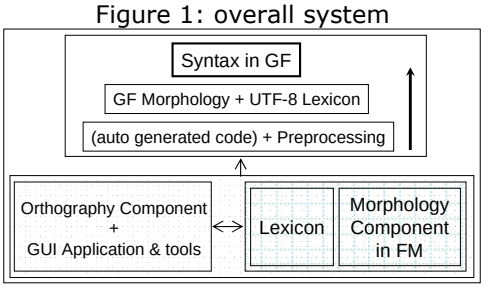
words; e.g. the use of Arabic and Persian text in Urdu, mostly in the religious, as well as in the slightly old literary text; where text is normally aided by the Quranic verses and the Persian poetry. Similarly, the text from news domain shows a big number of proper nouns and foreign words taken from English.

7. Syntax and a Complete Example

To show the usability and effectiveness of our approach, we implement a syntax component in Grammatical Framework [4] (GF henceforth) by reusing the above mentioned components. The morphology component and the lexicon can easily be ported into GF code by using the data export utility of FM. Later we apply some preprocessing and save the lexicon directly in UTF-8 format. We build a fragment of Urdu Syntax in GF as a separate component of the system on top of morphology.

The overall picture of the system is shown in the following diagram:

Figure 1: overall system



In GF, a grammar is a combination of two parts: The Abstract syntax and the Concrete syntax. We show the Abstract syntax of a function below from our implementation:

fun UsePresS: NP → VP → S;

While the concrete syntax for the same function is given below along with some explanation:

UsePresS np vp =
{s = np.s ! Nom ++ vp.s ! Present ! np.p ! np.n ! np.g}

It states that, the nominative form of noun phrase (which is a Pronoun + Postposition here) could be combined by the verb phrase (which is a Verb + Auxiliary here) to form a sentence and they must agree for their Person, Number and Gender parameters. e.g. (is ko kɪtɑbeɳ leni heɳ, اِس کو کتابیں لینی ہیں, He/she suppose to take the books). We show a complete example based on the above sentence below:

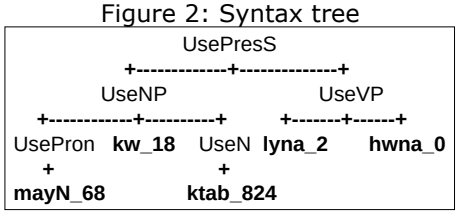
Transliteration: a(i)s kw ktabyN lyny hyN
Morphological analysis:

< اِس, a(i)s >
yih_66. یِہ +DemPron - Sg Obl - Pers3_Near
mayN_68. مَیں +PersPron - Sg Pers3_Near Obl-
< کو, kw >
kw_18. کو +PostP -
< کتابیں, ktabyN >
ktab_824. کتاب +N - Pl Nom - Fem
< لینی, lyny >
lyna_2. لینا +Verb - Inf_Fem -
< ہیں, hyN >
hwna_0. ہونا +Verb_Aux - Present Pers1 Pl Masc -
hwna_0. ہونا +Verb_Aux - Present Pers1 Pl Fem -
...

Syntactic parsing:

UsePresS
(UseNP (UsePron mayN_68) kw_18 (UseN ktab_824))
(UseVP lyna_2 hwna_0)

Figure 2: Syntax tree



9. Conclusion

This work is attractive for four main reasons, (1) for being a comprehensive implementation for Urdu (morphology) that covers the linguistic abstraction adequately, (2) for being elegant, extensible and reusable, (3) for its ease and speed of development and (4) its free availability as open-source software.

However, we do not provide a fully vocalized lexicon which is a fundamental limitation. Further, for the moment, for analysis of words, one cannot check if there exist any orthographically different versions of a word in the lexicon.

9. References

[1] M. Forsberg, A. Ranta. 2004. "Functional Morphology" *ICFP'04*, pp. 213-223, *Proceedings of the Ninth ACM SIGPLAN International Conference of Functional Programming*

[2] M. Forsberg, H. Hammarström, A. Ranta. 2006. "Lexicon Extraction from Raw, Text Data". *In: Salakoski, T. and Ginter, F. and Pyysalo, S. and Pahikkala, T. (eds.) Advances in Natural Language Processing*, FinTAL, Finland, August 23-25, 2006, pp. 488-499, SPRINGER, LNCS 4139.

[3] ICU4J 3.4. 2006. *International Components for Unicode for Java. Version 3.6*. http://icu.sourceforge.net

[4] A. Ranta. 2004. "Grammatical Framework: A Type-Theoretical Grammar Formalism". *Journal of Functional Programming*, 14(2):145-189.