Proceedings of LREC 2010 : Seventh International Conference on Language Resources and Evaluation, May 17-23, 2010, Valletta, Malta / Nicoletta Calzolari ... (Hrsg.). - European Language Resources Association (ELRA), 2010. - S. 2921-2927. - ISBN 2-9517408-6-7

Transliterating Urdu for a Broad-Coverage Urdu/Hindi LFG Grammar

Muhammad Kamran Malik‡, Tafseer Ahmed†, Sebastian Sulger†, Tina Bögel†, Atif Gulzar‡, Ghulam Raza†, Sarmad Hussain‡, Miriam Butt†

†Universität Konstanz, ‡CRULP FAST NUCES
†Konstanz, Germany; ‡Lahore, Pakistan
sebastian.sulger@uni-konstanz.de

Abstract

In this paper, we present a system for transliterating the Arabic-based script of Urdu to a Roman transliteration scheme. The system is integrated into a larger system consisting of a morphology module, implemented via finite state technologies, and a computational LFG grammar of Urdu that was developed with the grammar development platform XLE (Crouch et al. 2008). Our long-term goal is to handle Hindi alongside Urdu; the two languages are very similar with respect to syntax and lexicon and hence, one grammar can be used to cover both languages. However, they are not similar concerning the script – Hindi is written in Devanagari, while Urdu uses an Arabic-based script. By abstracting away to a common Roman transliteration scheme in the respective transliterators, our system can be enabled to handle both languages in parallel. In this paper, we discuss the pipeline architecture of the Urdu-Roman transliterator, mention several linguistic and orthographic issues and present the integration of the transliterator into the LFG parsing system.

1. Introduction

This paper introduces a Roman transliterator for the Urdu Arabic-based script, which is used as part of a broad-coverage grammar for the South Asian language Urdu being developed within the ParGram (Parallel Grammar) project (Butt et al., 1999; Butt et al., 2002). Very few resources exist for Urdu and part of the project is to build a broad-coverage finite-state morphological analyzer for Urdu and to connect it up with the grammar via the morphology-syntax interface, defined by Kaplan et al. (2004) for Lexical-Functional Grammar (LFG) (Dalrymple, 2001).

Hindi, although being very similar to Urdu, is written in a different script: Devanagari. Since the goal of the Urdu Grammar project is to eventually be able to process both Urdu and Hindi text, our approach is to abstract away from both of the scripts to a common Roman transliteration scheme. This will allow us to use just one morphological analyzer and one LFG grammar for both languages. As a first step, we have developed a transliterator for Urdu via a cascaded set of tools written in C++. This paper presents the toolset as well as the integration of the transliterator into the pipeline consisting of a morphology module and an LFG grammar, using the LFG grammar development platform XLE (Crouch et al., 2008).

2. Particularities of the Script

The Urdu script uses an extended Arabic character set. It uses letters for consonants and aerabs (diacritics) for vowels. The combination of these realize a relatively rich phonemic inventory. The use of aerabs, however, is not very common in written Urdu, which gives rise to ambiguity and makes it complicated for text-to-speech systems to correctly interpret the string. To avoid problems due to ambiguity, one module of the transliterator guesses the correct vowels using a word form lexicon (section 3.2).

Urdu letters/characters can in general be mapped from graphemes to phonemes in a regular, one-to-one fashion, so that a simple rule-based model can be developed (Hussain, 2004). There are four types of characters in Urdu:

(1) simple consonant characters;
(2) dual (consonant and vocalic) behavior characters;
(3) vowel modifier character;
(4) consonant modifier character.

The characters in the first category can be rewritten in a straightforward way, mapping them one-to-one. The characters in the second category exhibit dual behavior, meaning that they can refer to consonants in some contexts and to vowels in other contexts. Rules have to be designed to account for this behavior. The third category consists of the vowel modifier character *Noon Ghunna*, which nasalizes a preceding vowel. The fourth category contains the character *Do-Chashmey Hay*, which can combine with stops and affricates to form aspirated forms of consonants.

Many words of Urdu are loan words from Arabic and Persian, which were borrowed retaining the original spelling. As a consequence, many Arabic/Persian graphemes map onto a single Urdu phoneme — but the different Arabic/Persian consonant characters are still used in written Urdu. For example, the Urdu characters ذ , ز , ض and ظ all map to the same sound */ z /* (section 4.4.).

The Urdu script contains diacritics on consonant characters to represent vowels. Vowel diacritics are combined with consonants of category (1) to indicate short vowels; they are combined with dual behavior characters of category (2) to indicate long vowels.

All of these phenomena were dealt with by implementing a pipeline of modules which are either rule-based or lexicon-based, taking as input Unicode Urdu text and producing Unicode Roman text based on a Roman transliteration scheme as output.

3. Transliteration Scheme

Our broad coverage grammar will parse both Urdu and Hindi, hence a transliteration scheme is designed to represent characters of Urdu and Hindi.

The consonants are represented by the similar sound consonant characters in roman letters. We define correspondences between Unicode Urdu consonants and simple Roman characters as in Table 2, which is shown at the end of this document. The scheme is case-sensitive, hence *t* and *T* represent two different consonants. In Urdu (and Hindi), we have pairs of dental and retroflex consonants. The first type of consonants are represented by small letters e.g. *t* and *d*. The corresponding retroflexes are represented by the capital letters i.e. *T* and *D*.
However, a capital letter does not always correspond to a retroflex. The letters *S* is used for voiceless palatal fricative as used in *shop*. The letter *N* is used after long vowel to represent nasalization. Similarly, *H* used after a consonant represents the aspirated form of that consonant.
There are many loan words from Arabic and Persian that include graphemes from these languages, retained in the Urdu spelling. As a consequence, there are several different Urdu characters mapping to the same phoneme (e.g., ذ , ز , ض and ظ all map to the same sound / *z* /). The transliteration module maps the UZT sequence of the genuine Urdu character to a general letter *t*; it maps the loan characters to *t2, t3, t4* etc. For example, as ز is the most common letter among the above, it is mapped to *z*. ذ , ض and ظ are mapped to *z2, z3, z4* respectively. As a result, the lexicon is kept simple to read in most of the cases.
Urdu has 3 short and 8 long vowels. The long vowels also have nasalized versions that are represented by adding *N* after the vowel. The short vowels are written as diactric marks in Urdu script. Table 3 shows the short vowels used after the consonant ب *bay*. As the diacrtic marks can not be rendered without a consonant, we have to use a consonant to show the shape and sequence of the diacritics and vowels in Urdu script. The three short vowels are *a* for *Zabar*, *i* for *Zer* and *u* for *Pesh*. The long vowels are either a dual consonant/vowel character, or a sequence of diactric mark followed by these characters.
The Unicode characters are mapped to their UZT counterparts in step 3 of the pipeline, and to their Roman letter equivalents of our scheme in step 4 of the pipeline.

4. Transliteration Pipeline Architecture

To transliterate from Unicode Urdu to our Roman letter scheme, a component-based approach was taken and a pipeline including several modules was implemented in C++. Figure 1 shows the overall architecture of the transliterator. Each component in the pipeline is a standalone application that can be used for other NLP tasks.

4.1. Normalization

In the Unicode standard notation of Arabic, some characters can be written in two forms: the *composed* form as in (1a) and the *decomposed* form (1b). In their composed form, characters occur as a single entity in the Unicode block (e.g. U+0622 for Long *Alef*). In their decomposed form, characters are written by combining two or more Unicode characters (e.g. Long *Alef* can be combined out of U+0627 and U+0653). To avoid a duplication of rules, the input text was normalized to the composed character form; character sequences like the one in (1b) are therefore normalized to the composed form in (1a).

(1) a. composed form:
Alef madda: آ *ā*
b. decomposed form:
Alef: ا *a*
+ lengthening diacritic *madda*: ٓ

4.2. Diacritization

The diacritization component deals with the problem of the vowel diacritics. Urdu is normally written without any aerabs (vowel diacritics), which makes it difficult to interpret. This component uses the Urdu lexicon data developed at the Center for Research in Urdu Language Processing (CRULP), containing 80.000 diacritized Urdu words (Ijaz and Hussain, 2007). The diacritization component places aerabs in the input text by looking up the words in the Urdu lexicon. If multiple options are available, the component selects the first option encountered.
Choosing the first option results in loosing information, since the right word might not always be the first one encountered. A possible improvement would be to give all the possible word forms as output, keep them throughout the pipeline and let the morphology and syntax modules decide which is the correct one.

4.3. Unicode to Urdu Zabta Takhti (UZT) Conversion

The Urdu Zabta Takhti (UZT) encoding is a standard developed for Urdu language processing that maps every single Unicode Urdu character onto a sequence of numbers (Hussain and Afzal, 2001). For software development in Urdu, there was no industry standard available like ASCII for English. UZT now provides such a standard and was included for reasons of compatibility with other applications. An example is shown in (2) for *čābī* 'key'.

(2) a. Urdu Unicode text
čābī چابِی
b. UZT–converted text
čābī `898083120`

4.4. Transliteration

This component applies transliteration rules which convert the number-based UZT notation to the Roman letter-based scheme. The rules are compiled into a finite-state machine using the XFST toolset (Beesley and Karttunen, 2003).

(3) a. UZT–converted text
čābī `898083120`
b. transliterated Roman letter-based notation
čābī `cAbI`

The transformation of UZT to our transliteration scheme is not a simple one to one replacement. The dual (consonant and vowel) characters ی , ے , و and آ can be transliterated in different ways based on the context.
When ا *Alef* is used at the beginning of a word, it is used as a dummy consonant for carrying the vowels. Hence ا at

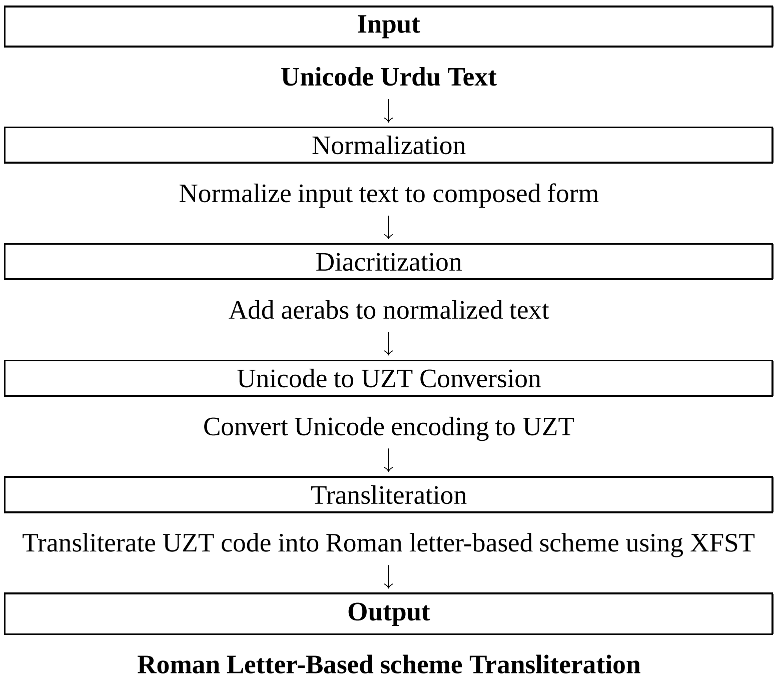

Input
Unicode Urdu Text
Normalization
Normalize input text to composed form
Diacritization
Add aerabs to normalized text
Unicode to UZT Conversion
Convert Unicode encoding to UZT
Transliteration
Transliterate UZT code into Roman letter-based scheme using XFST
Output
Roman Letter-Based scheme Transliteration

Figure 1: Cascaded Transliterator Architecture

the beginning of the word is transliterated to null. At other places, it is transliterated as a long vowel *A*. The word آب *ab* is composed out of the vowel ا *Alef*, the diacritic *Zabar* and the consonant ب *bay*. As ا appears at word initial position, it is not transliterated; we get *ab* as its transliteration. On the other hand, the word بابا has ا *Alef* at non-initial position, hence it is transliterated into the vowel, and the output is *bAbA*.

The handling of the character ی *Chooti-ye* is more complex. It can represent the consonant *y* or the vowels *I*, *E* and *e*. If it is preceeded by the diacritics *Zer* or *Pesh*, then it is considered as part of the vowel combination, and the previous vowel and *Chooti-ye* are transliterated as one single vowel. *Zabar*, and *Chooti-ye* are transliterated as *e* and *Zer* and *Chooti-ye* are transliterated as *I*. If there is no vowel before it, then it is transliterated as *E*.

The examples are مِیل 'mile', میل 'meeting' and مَیل 'dirt/filth'. As مِیل has a *Zer* before *Chooti-ye*, it is transliterated as *mIl*, مَیل has a zabar before *Chooti-ye*, hence it is transliterated as *mel*. There is no diacritic preceeding *Chooti-ye* in میل, hence it is transliterated as *mEl*.

On the other hand, if *Chooti-ye* is followed by a vowel or vowel combination, then it is transliterated as consonant *y*. The word بُنیَاد has vowel *Zabar* and *Alef* following *Chooti-ye*. Hence, it is transliterated as *bunyAd*.

5. Evaluation of the Transliterator

A sample data set of 1000 unique high frequency words was compiled. The data was taken from an 18 million word Urdu corpus (Hussain, 2008) collected from two news websites.[1] The frequency is calculated simply by counting the occurrence of a particular word:

(4) Frequency: $F(W_i); 0 < i <= N$

W_i is a unique word, $F(W_i)$ its number of occurrences, i the word index, and N the size of the corpus. The accuracy of the system given the test corpus was then calculated as in (5).

(5) Accuracy: $A = C_w/T_w$

A stands for the accuracy of the system, C_w for the words correctly transliterated and T_w for the total number of words taken as input. The results are given in Table 1. The system successfully and accurately transliterates 99.5% of the data, if the data is fully diacritized. However, the accuracy is reduced to 92.5% for data containing non-diacritized and foreign words. Accuracy was checked manually.

6. Integration of the Transliterator in XLE

The XLE platform is used by grammar writers to develop and load an LFG grammar and produce syntactic structures — C- and F-Structures (Dalrymple, 2001). Before annotating syntactic structure, the program can break input text into sentences, tokenize sentences into words and look up words in lexicons. All of these pre-processing steps are usually handled via finite-state transducers (Kaplan et al., 2004).

The grammar developed in the Urdu ParGram project uses the same basic architecture. After tokenization, XLE looks up tokens in a computational morphology developed using XFST (Bögel et al., 2007; Beesley and Karttunen, 2003).

[1]Jang Urdu (http://www.jang net/Urdu/), BBC Urdu (http://www.bbc.co.uk/urdu/)

Test Corpus Size	$A = C_w/T_w$ (diacritized input)	$A = C_w/T_w$ (input without diacritics, with foreign words)
1000	0.995	0.925

Table 1: Accuracy Results

The morphology is encoded using the Roman transliteration of Urdu. Thus, both Urdu and Hindi will be able to be processed via a single lexicon file, grammar and morphological component. This not only facilitates lexicon development, but also reduces the grammar development effort.
The Urdu transliterator is integrated into the front-end of XLE. The transliterator takes an Urdu Unicode file as input and produces a Roman transliteration encoded in Unicode (UTF-8) as described in section 3. The transliterated sentence is fed into the remaining XLE pipeline consisting of the morphology and the syntax. That is, if we feed the Urdu script sentence in (6a) into XLE, we get the right side of (6b) as output from the transliterator.

(6) a. example (*gARI calI* ‘The car worked/started.’):
gāṛī čalī　　گاڑِی چَلِی

b. transliterator output:
gāṛī čalī　　`gARI calI`

Next, the tokenizer inserts token boundaries (TB), so that XLE can identify individual tokens to look up in the XFST morphology.

(7) a. tokenizer input:
gāṛī čalī　　`gARI calI`

b. tokenizer output:
gāṛī čalī　　`gARI TB calI TB`

XLE then passes the individual tokens on to the morphology, which consists of a finite-state transducer producing a sequence of morphosyntactic tags for each of the input tokens as in (8).

(8) morphology output:

gāṛī	`gARI+Noun+Fem+Sg`
čalī	`calI+Verb+Perf+Fem+Sg`

The morphology output is given back to XLE, which feeds each of the tokens including their attached tags into the syntax module, which then produces syntactic structures based on the LFG framework. The process is shown here for the example in (6a). Sublexical rules attach the morphological tags to the correct lexical categories as in Figure 1. Functionally annotated syntactic rules produce C-Structures as given in Figure 2 and F-Structures as given in Figure 3. The C- and F-structures follow the guidelines established by the ParGram Project (Butt et al., 1999; Butt et al., 2002).

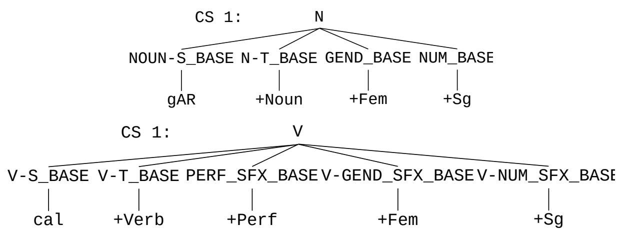


Figure 1: Lexical analysis in XLE with morphological tags

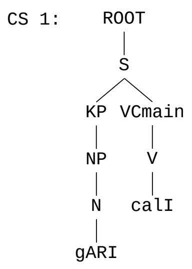
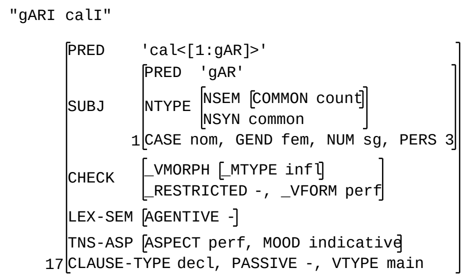


Figure 2: Example C-Structure in XLE



```
"gARI calI"

  PRED      'cal<[1:gAR]>'
            PRED  'gAR'
  SUBJ      NTYPE  NSEM [COMMON count]
                   NSYN common
          1 CASE nom, GEND fem, NUM sg, PERS 3
  CHECK     _VMORPH [_MTYPE infl]
            _RESTRICTED -, _VFORM perf
  LEX-SEM   [AGENTIVE -]
  TNS-ASP   [ASPECT perf, MOOD indicative]
17CLAUSE-TYPE decl, PASSIVE -, VTYPE main
```

Figure 3: Example F-Structure in XLE

7. Conclusion and Future Work

We presented a transliterator that converts Unicode Urdu script to Unicode based on an Roman letter transliteration scheme using a cascaded sequence of modules. We successfully dealt with language specific problems like multiple characters for one sound and diacritization. We abstracted away from the script to a Roman transliteration in order to eventually parse both Urdu and Hindi. The transliterator has been successfully integrated into the Urdu ParGram grammar.
However, there is an issue with generation since the present C++ transliterator is not bidirectional. One solution we are exploring is to reimplement the transliteration cascade in terms of a finite-state transducer (e.g., as sketched in Malik (2006)), which is inherently bidirectional.
As it is, we have built and integrated an initial transliterator with high accuracy (and efficient performance) into the existing Urdu ParGram grammar, thus leaving the door open to parse Hindi as well with just a minimum of additional grammar development effort. In addition, the entire transliterator can not only be used as a stand-alone module, just parts of it could also be used, so that one could convert to UZT instead of the Roman transliteration scheme, for example, depending on the application. The transliterator thus allows for maximum flexibility while providing high accuracy due to the built-in lexicon and its deterministic rule-based character.

8. References

Kenneth Beesley and Lauri Karttunen. 2003. *Finite State Morphology*. CSLI Publications, Stanford, CA.

Tina Bögel, Miriam Butt, Annette Hautli, and Sebastian Sulger. 2007. Developing a finite-state morphological analyzer for Urdu and Hindi. In *Proceedings of the Sixth International Workshop on Finite-State Methods and Natural Language Processing*, Potsdam.

Miriam Butt, Tracy H. King, María-Eugenia Niño, and Frédérique Segond. 1999. *A Grammar Writer's Cookbook*. CSLI Publications.

Miriam Butt, Helge Dyvik, Tracy H. King, Hiroshi Masuichi, and Christian Rohrer. 2002. The Parallel Grammar project. In *Proceedings of COLING-2002, Workshop on Grammar Engineering and Evaluation*, pages 1–7, Taipei.

Dick Crouch, Mary Dalrymple, Ronald M. Kaplan, Tracy Holloway King, John T. Maxwell III, and Paula Newman, 2008. *XLE Documentation*. Palo Alto Research Center.

Mary Dalrymple. 2001. *Lexical Functional Grammar*. Academic Press.

Sarmad Hussain and Muhammad Afzal. 2001. Urdu computing standards: Urdu zabta takhti (uzt) 1.01. In *Proceedings of the 2001 IEEE International Multi-Topic Conference*, pages 223–228.

Sarmad Hussain. 2004. Letter-to-sound conversion for Urdu text-to-speech system. In *Proceedings of COLING-2004, Workshop on Arabic Script Based Languages*, Geneva, Switzerland.

Sarmad Hussain. 2008. Resources for Urdu Language Processing. In *Proceedings of the 6th Workshop on Asian Language Resources*, IIIT Hyderabad.

Madiha Ijaz and Sarmad Hussain. 2007. Corpus based Urdu lexicon development. In *Proceedings of the Conference on Language and Technology 2007 (CLT07)*, University of Peshawar, Pakistan.

Ronald M. Kaplan, John T. Maxwell III, Tracy H. King, and Richard Crouch. 2004. Integrating finite-state technology with deep LFG grammars. In *Proceedings of ESSLLI, Workshop on Combining Shallow and Deep Processing for NLP*.

Abbas Malik. 2006. Hindi Urdu machine transliteration system. MSc Thesis, University of Paris 7.

Unicode Urdu character	Roman letter in transliteration scheme
ب	b
پ	p
ت	t
ٹ	T
ث	s2
ج	j
چ	c
ح	h2
خ	x
د	d
ڈ	D
ذ	z2
ر	r
ڑ	R
ژ	y2
ز	z
س	s
ش	S,S2
ص	s3
ض	z3
ط	t2
ظ	z4
ع	a2
غ	G
ف	f
ق	q
ک	k
گ	g
ل	l
م	m
ن	n
ہ	h
ة	t3
و	v
ھ	H
ں	N
ی	y

Table 2: Transliteration Scheme for consonants

Urdu vowel (with consonant ب)	Urdu names of vowel characters	Roman letter in transliteration scheme
بَ	Zabar	ba
بِ	Zer	bi
بُ	Pesh	bu
بَا	Zabar Alif	bA
بِی	Zer Chooti-ye	bI
بُو	Pesh Wao	bU
بو	Wao	bO
بَو	Zabar Wao	bo
بے	Chooti/Bari-ye	bE
بَے	Zabar Chooti/Bari-ye	be

Table 3: Transliteration Scheme for vowels. The vowels with consonant ب *bay*
.