A Transliteration System for Urdu/Hindi Integrated in the Urdu ParGram Grammar

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

Universität Konstanz†, CRULP, FAST-NUCES ‡

Transliteration – why and what for?

Urdu: Arabic script | **Hindi: Devanagari script**

یاں بھلا کر ترا بھلا ہوگا
اور درویش کی صدا کیا ہے

हां भला कर तिरा भला होगा
और दर्वेश की सदा क्या है

The same text – two different scripts...

We would like to handle both!
(Although we focus on Urdu for the time being.)

Solution:

- Abstract away from each script to a common transliteration
- Use one lexicon and grammar for both languages

Particularities of the Urdu Script

Urdu: Script uses extended Arabic character set

- Full letters for consonants, aerabs (diacritics) for vowels
- Written Urdu: Aerab diacritics are not common

→ Ambiguity: Difficult to interpret the string

- Four different types of full characters in Urdu

(1) Simple consonant characters ف → /f/
(2) Dual behaviour characters ے → /j/ or /æ/
(3) Vowel modifier character ں → /~/
(4) Consonant modifier character ھ → /ʰ/

- Extensive borrowing from Arabic/Persian

→ Foreign spelling retained in written Urdu
→ Arabic/Persian graphemes map onto a single Urdu phoneme (e.g., ص , ث , س all map to */s/*).

The Basic Architecture

Goal: Transliterate from Unicode Urdu to ASCII scheme

- Component-based approach: Pipeline implemented in C++ using four separate modules (see center)
- Components can be used as standalone applications
- Transliterator: Integrated in a computational grammar based on Lexical-Functional Grammar framework using Xerox Linguistic Environment (XLE) grammar development platform (Butt and King 2007).

STEP 1: NORMALIZATION

Unicode Arabic: Characters can be written in two ways

- *Composed* form: Single entity in Unicode block

Alef madda: آ *ā*

- *Decomposed* form: Combined out of 2 or more characters

Alef: ا *a*

+ lengthening diacritic *madda*: ◌̆

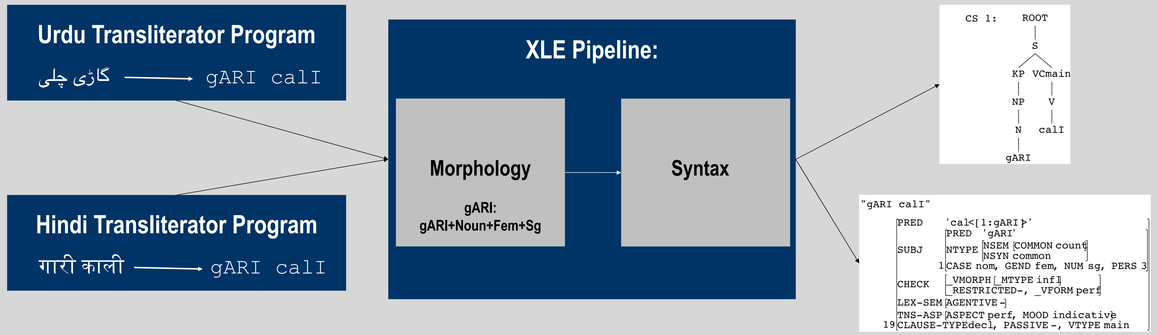
→ To avoid a duplication of rules, the input text is normalized to composed character form.

Transliterator Pipeline Architecture for Urdu

INPUT (Unicode Urdu Text)
↓
STEP 1: NORMALIZATION (Normalize Input Text to Composed Form)
↓
STEP 2: DIACRITIZATION (Add Aerabs to Normalized Form)
↓
STEP 3: UNICODE TO URDU ZABTA TAKHTI CONVERSION (Convert Unicode Encoding to UZT)
↓
STEP 4: TRANSLITERATION (Transliterate UZT Code into Letter-Based ASCII Scheme)
↓
OUTPUT (Letter-Based ASCII Scheme Transliteration)

Integration in the XLE Program

XLE grammar development platform: Load Morphological Analyzer and LFG grammar, parse text, produce syntactic structures



- Morphology: Encoded in ASCII-based transliteration of Urdu/Hindi

→ Both Urdu and Hindi will be able to be processed via *a single lexicon file, grammar and morphological component*

→ Facilitates lexicon development and reduces the grammar development effort

Evaluation of the Transliterator

- Sample test data: 1.000 unique high frequency words
- Data taken from 18 million word corpus (Hussain 2008)

Accuracy of the system:

Accuracy: $A = C_w / T_w$

A : Accuracy of the system
C_w : Words correctly transliterated
T_w : Total number of words taken as input

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**

STEP 2: DIACRITIZATION

Vowel diacritics are normally not written in Urdu

- Urdu Lexicon Data (Center for Research in Urdu Language Processing; 80.000 diacritized words)

→ Lexicon lookup: Place diacritics in input text by looking up words in the lexicon

→ Ambiguity created by absence of aerab diacritics is resolved

STEP 3: UNICODE TO URDU ZABTA TAKHTI CONVERSION

Urdu Zabta Takhti (UZT): Standard encoding for Urdu language processing

- UZT: Maps Unicode Urdu characters onto *unique* number sequences (Afzal and Hussain 2001)
- UZT: Developed because there was no standard industry codepage available

→ Included in pipeline for reasons of compatibility

a) Urdu Unicode text:
čābī چابی
b) UZT-converted text:
čābī `898083120`

STEP 4: TRANSLITERATION

Transliteration using Finite-State Machinery: Fast & efficient

- Transliteration rules convert number-based UZT notation to ASCII-based transliteration scheme
- Rules compiled into a finite-state machine using the Xerox Finite-State Tools (XFST; Beesley and Karttunen 2003)

a) UZT-converted text:
čābī `898083120`
b) Transliterated, letter-based ASCII notation:
čābī `cAbI`

- Loan words from Arabic/Persian include graphemes from these languages

→ Some Urdu graphemes map onto the same phoneme:

ص , ث , س → /s/

Solution:

- Map genuine Urdu character to general letter, foreign characters to variants – keeps lexicon easy to read in most cases!

س → s
ث → s2 ; ص → s3

References

Afzal, Muhammad and Hussain, Sarmad. 2001. Urdu Computing Standards: Development of Urdu Zabta Takhti (UZT) 1.01. In Proceedings of the 2001 IEEE International Multi-topic Conference, pages 216– 222.

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

Butt, Miriam and Tracy Holloway King. 2007. 'Urdu in a Parallel Grammar Development Environment'. In T. Takenobu and C.-R. Huang (eds.) Language Resources and Evaluation: Special Issue on Asian Language Processing: State of the Art Resources and Processing 41, pages 191-207.

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