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

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

LREC2010, Malta
Contents of the Talk:

1. Context of Work – the ParGram Project
2. Urdu & Challenges in Transliterating Urdu
3. Transliterator Architecture
4. Integrating the Transliterator in the ParGram Urdu Grammar
Context of Work

- Computational LFG grammar in development in Konstanz
Context of Work

- Computational LFG grammar in development in Konstanz
- Aim: large-scale LFG grammar for parsing Urdu/Hindi
Context of Work

- Computational LFG grammar in development in Konstanz
- Aim: large-scale LFG grammar for parsing Urdu/Hindi
- Grammar is part of the ParGram project
Context of Work

- Computational LFG grammar in development in Konstanz
- Aim: large-scale LFG grammar for parsing Urdu/Hindi
- Grammar is part of the ParGram project
  - Collaborative, world-wide research project
Context of Work

- Computational LFG grammar in development in Konstanz
- Aim: large-scale LFG grammar for parsing Urdu/Hindi
- Grammar is part of the ParGram project
  - Collaborative, world-wide research project
  - Devoted to developing *parallel* LFG grammars for a variety of languages
Context of Work – the ParGram Project

Computational LFG grammar in development in Konstanz

Aim: large-scale LFG grammar for parsing Urdu/Hindi

Grammar is part of the ParGram project

- Collaborative, world-wide research project
- Devoted to developing parallel LFG grammars for a variety of languages
- Features and analyses are kept parallel for easy transfer between languages
Context of Work

- Computational LFG grammar in development in Konstanz
- Aim: large-scale LFG grammar for parsing Urdu/Hindi
- Grammar is part of the ParGram project
  - Collaborative, world-wide research project
  - Devoted to developing parallel LFG grammars for a variety of languages
  - Features and analyses are kept parallel for easy transfer between languages
- Languages involved:
Context of Work

- Computational LFG grammar in development in Konstanz
- Aim: large-scale LFG grammar for parsing Urdu/Hindi
- Grammar is part of the ParGram project
  - Collaborative, world-wide research project
  - Devoted to developing parallel LFG grammars for a variety of languages
  - Features and analyses are kept parallel for easy transfer between languages
  - Languages involved:
    → large-scale: English, German, French, Japanese, Norwegian
Context of Work

- Computational LFG grammar in development in Konstanz
- Aim: large-scale LFG grammar for parsing Urdu/Hindi
- Grammar is part of the ParGram project
  - Collaborative, world-wide research project
  - Devoted to developing parallel LFG grammars for a variety of languages
  - Features and analyses are kept parallel for easy transfer between languages
  - Languages involved:
    - large-scale: English, German, French, Japanese, Norwegian
    - smaller-scale (yet...): Welsh, Georgian, Hungarian, Turkish, Chinese, **Urdu** (among many others)
The ‘Parallel’ in ParGram

Analysis for transitive sentence in English ParGram grammar (F-Structure, “Functional Structure”):
The ‘Parallel’ in ParGram

Analysis for transitive sentence in English ParGram grammar (F-Structure, “Functional Structure”):

"Nadya saw the book."

```
PRED 'see<1:Nadya> [113:book]'  |
  PRED 'Nadya' |
  CHECK [_LEX-SOURCE morphology _PROPER known-nam] |

SUBJ NTYPE NSEM [PROPER [NAME-TYPfirst_name PROPER-TYPname]] |
  NSYN proper |
  1 CASE nom, GEND-SEM female, HUMAN +, NUM sg, PERS 3 |
  PRED 'book' |
  CHECK [_LEX-SOURCE countnoun-lex] |

OBJ NTYPE NSEM [COMMON count] |
  NSYN common |
  SPEC [DET PRED 'the'] |
  113 CASE obl, NUM sg, PERS 3 |
  CHECK [_SUBCAT-FRAMEV-SUBJ-OBJ] |

TNS-ASP MOOD indicative, PERF --, PROG --, TENSE past |

57 CLAUSE-TYPE decl, PASSIVE --, VTYPE main |
```
The ‘Parallel’ in ParGram (cont.)

Analysis for the same transitive sentence in Urdu ParGram grammar (F-Structure, “Functional Structure”):
The ‘Parallel’ in ParGram (cont.)

Analysis for the same transitive sentence in Urdu ParGram grammar (F-Structure, “Functional Structure”):

"nAdiyah nE kitAb dEkHI"

\[
\begin{array}{l}
\text{PRED} \quad 'dEkH<[1:nAdiyah] \ [19:kitAb>]' \\
\quad \text{PRED} \quad 'nAdiyah' \\
\quad \text{CHECK} \quad ['\_NMORPH\_obl'] \\
\text{SUBJ} \quad \text{NTYPE} \quad ['\_NSEM [\_PROPER [\_PROPER-_TYPE\_ename]]'] \\
\quad \quad \quad \text{NSYN} \quad \text{proper} \\
\quad \quad \quad \text{SEM-PROP} [\text{SPECIFIC +}] \\
\quad 1[\text{CASE} \quad \text{erg, GEND} \quad \text{fem, NUM} \quad \text{sg, PERS} \quad 3] \\
\quad \quad \quad \text{PRED} \quad '\text{kitAb}' \\
\text{OBJ} \quad \text{NTYPE} \quad ['\_NSEM [\_COMMON \text{count}]'] \\
\quad \quad \quad \text{NSYN} \quad \text{common} \\
\quad 19[\text{CASE} \quad \text{nom, GEND} \quad \text{fem, NUM} \quad \text{sg, PERS} \quad 3] \\
\quad \quad \quad \text{CHECK} \quad ['\_VMORPH [\_MTYPE \text{inf}]'] \\
\quad \quad \quad \text{-_RESTRICTED-, _VFORM perf} \\
\quad \quad \quad \text{LEX-SEM} [\text{AGENTIVE +}] \\
\quad \quad \quad \text{TNS-ASP} [\text{ASPECT perf, MOOD} \quad \text{indicative}] \\
\quad 40[\text{CLAUSE-TYPE} \quad \text{decl, PASSIVE -}, \text{VTYPE} \quad \text{main}]
\end{array}
\]
The ‘Parallel’ in ParGram (cont.)

Analysis for the same transitive sentence in Urdu ParGram grammar (F-Structure, “Functional Structure”):

"nAdiyah nE kitAb dEkHI"

→ Analyses are kept parallel where possible
The ‘Parallel’ in ParGram (cont.)

Analysis for the same transitive sentence in Urdu ParGram grammar (F-Structure, “Functional Structure”):

"nAdiyah nE kitAb dEkHI"

→ Analyses are kept parallel where possible
→ Features are kept parallel where possible
Urdu is
Urdu is

- a South Asian language spoken primarily in Pakistan and India
Urdu is

- a South Asian language spoken primarily in Pakistan and India
- descended from (a version of) Sanskrit (sister language of Latin)
Urdu

Urdu is

- a South Asian language spoken primarily in Pakistan and India
- descended from (a version of) Sanskrit (sister language of Latin)
- structurally identical to Hindi (spoken mainly in India)
Urdu is

- a South Asian language spoken primarily in Pakistan and India
- descended from (a version of) Sanskrit (sister language of Latin)
- structurally identical to Hindi (spoken mainly in India)
- together with Hindi the fourth most spoken language in the world
  (∼ 250 million native speakers)
Two Scripts, One Language

- While Urdu uses an Arabic-based script, Hindi uses Devanagari
Two Scripts, One Language

- While Urdu uses an Arabic-based script, Hindi uses Devanagari.
- The same couplet by the poet Mirza Ghalib in both of the scripts:
Two Scripts, One Language

- While Urdu uses an Arabic-based script, Hindi uses Devanagari.
- The same couplet by the poet Mirza Ghalib in both of the scripts:

<table>
<thead>
<tr>
<th>Urdu</th>
<th>vs.</th>
<th>Hindi</th>
</tr>
</thead>
<tbody>
<tr>
<td>پائیں بھالکر ترا بھالا توگا</td>
<td></td>
<td>हां भला कर तिरा भला होगा</td>
</tr>
</tbody>
</table>
Two Scripts, One Language

- While Urdu uses an Arabic-based script, Hindi uses Devanagari.
- The same couplet by the poet Mirza Ghalib in both of the scripts:

  **Urdu**
  
  پیان بہلا کر تیرا بہلا ہوگا
  اور روشنی کی سدا کیے

  **Hindi**
  
  हां भला कर तिरा भला होगा
  और दरबेश की सदा कूया है

- **Common transliteration in Latin alphabet:**
  
  hAN bHalA kar tirA bHalA hOgA
  yes good.M.Sg do then good be.Fut.M.Sg
  Or darvES kI sadA kyA he
  and dervish Gen.F.Sg call.F.Sg what be.Pres.3.Sg
  ‘Yes, do good then good will happen, what else is the call of the dervish.’
Abstracting Away from the Scripts

- Faced with 2 possibilities:
Abstracting Away from the Scripts

- Faced with 2 possibilities:
  1. Hard-coding the grammar and lexicon using *both scripts*
Abstracting Away from the Scripts

- Faced with 2 possibilities:
  1. Hard-coding the grammar and lexicon using both scripts
  2. Try to abstract away from scripts to a common transliteration
Abstracting Away from the Scripts

- Faced with 2 possibilities:
  1. Hard-coding the grammar and lexicon using both scripts
  2. Try to abstract away from scripts to a common transliteration

- Since one grammar and lexicon can deal with both languages, efficiency and size considerations commanded us to explore the common transliteration option...
Abstracting Away from the Scripts

- Faced with 2 possibilities:
  1. Hard-coding the grammar and lexicon using both scripts
  2. Try to abstract away from scripts to a common transliteration

- Since one grammar and lexicon can deal with both languages, efficiency and size considerations commanded us to explore the common transliteration option...

Current approach:
Abstracting Away from the Scripts

- Faced with 2 possibilities:
  1. Hard-coding the grammar and lexicon using both scripts
  2. Try to abstract away from scripts to a common transliteration

- Since one grammar and lexicon can deal with both languages, efficiency and size considerations commanded us to explore the common transliteration option...

Current approach:
- Abstract away from both scripts
Abstracting Away from the Scripts

- Faced with 2 possibilities:
  1. Hard-coding the grammar and lexicon using both scripts
  2. Try to abstract away from scripts to a common transliteration

- Since one grammar and lexicon can deal with both languages, efficiency and size considerations commanded us to explore the common transliteration option...

Current approach:
- Abstract away from both scripts
- Use a common ASCII-based transliteration (A-Z, a-z, 0-9)
Abstracting Away from the Scripts

- Faced with 2 possibilities:
  1. Hard-coding the grammar and lexicon using both scripts
  2. Try to abstract away from scripts to a common transliteration

- Since one grammar and lexicon can deal with both languages, efficiency and size considerations commanded us to explore the common transliteration option...

Current approach:
- Abstract away from both scripts
- Use a common ASCII-based transliteration (A-Z, a-z, 0-9)
- Encode a single grammar and lexicon in ASCII-based transliteration
Abstracting Away from the Scripts

Urdu Script

Hindi Script

Common ASCII-based Transliteration

Computational LFG Grammar
Abstracting Away from the Scripts

→ Size of the lexicon is kept minimal
Abstracting Away from the Scripts

→ Size of the lexicon is kept minimal
→ Grammar development effort is kept minimal
The Urdu Script: Some Peculiarities

- Uses extended Arabic character set
The Urdu Script: Some Peculiarities

- Uses extended Arabic character set
- Full letters for consonants/long vowels, *Aerabs* (diacritics) for short vowels
The Urdu Script: Some Peculiarities

- Uses extended Arabic character set
- Full letters for consonants/long vowels, *Aerabs* (diacritics) for short vowels
- Written Urdu: *Aerab* diacritics are not common
The Urdu Script: Some Peculiarities

- Uses extended Arabic character set
- Full letters for consonants/long vowels, *Aerabs* (diacritics) for short vowels
- Written Urdu: *Aerab* diacritics are not common
  - Means that short vowels are normally not written in the script
The Urdu Script: Some Peculiarities

- Uses extended Arabic character set
- Full letters for consonants/long vowels, *Aerabs* (diacritics) for short vowels
- Written Urdu: *Aerab* diacritics are not common
  - Means that short vowels are normally not written in the script
  - Results in some ambiguity – difficult to interpret the string
The Urdu Script: Some Peculiarities

- Uses extended Arabic character set
- Full letters for consonants/long vowels, *Aerabs* (diacritics) for short vowels
- Written Urdu: *Aerab* diacritics are not common
  - Means that short vowels are normally not written in the script
  - Results in some ambiguity – difficult to interpret the string
- Extensive borrowing from Arabic and Persian
The Urdu Script: Some Peculiarities

- Uses extended Arabic character set
- Full letters for consonants/long vowels, *Aerabs* (diacritics) for short vowels
- Written Urdu: *Aerab* diacritics are not common
  - Means that short vowels are normally not written in the script
  - Results in some ambiguity – difficult to interpret the string
- Extensive borrowing from Arabic and Persian
  - Foreign spelling retained in written Urdu
The Urdu Script: Some Peculiarities

- Uses extended Arabic character set
- Full letters for consonants/long vowels, *Aerabs* (diacritics) for short vowels
- Written Urdu: *Aerab* diacritics are not common
  - Means that short vowels are normally not written in the script
  - Results in some ambiguity – difficult to interpret the string
- Extensive borrowing from Arabic and Persian
  - Foreign spelling retained in written Urdu
  - Arabic and Persian graphemes map onto a single Urdu phoneme
The Urdu Script: Some Peculiarities

- Urdu has 4 different character classes:
The Urdu Script: Some Peculiarities

- Urdu has 4 different character classes:

1. Simple Consonant Characters, e.g. \( \text{فا} \rightarrow /f/ \)
The Urdu Script: Some Peculiarities

- Urdu has 4 different character classes:

(1) Simple Consonant Characters, e.g. ﬂ → /f/
(2) Dual (Consonant and Vocalic) Characters, e.g. แ → /j/ or /ae/
Urdu & Challenges in Transliterating Urdu

The Urdu Script: Some Peculiarities

- Urdu has 4 different character classes:
  
  1. Simple Consonant Characters, e.g. Ù → /f/
  2. Dual (Consonant and Vocalic) Characters, e.g. Ý → /j/ or /æ/
  3. A Vowel Modifier Character: Ù → /~/
The Urdu Script: Some Peculiarities

- Urdu has 4 different character classes:

1. Simple Consonant Characters, e.g.  → /f/
2. Dual (Consonant and Vocalic) Characters, e.g.  → /j/ or /ae/
3. A Vowel Modifier Character:  → /~/
4. A Consonant Modifier Character:  → /ʰ/
The Urdu Script: Some Peculiarities

- Urdu has 4 different character classes:

  1. Simple Consonant Characters, e.g. \( \text{ف} \rightarrow /f/ \)
  2. Dual (Consonant and Vocalic) Characters, e.g. \( \text{ہ} \rightarrow /j/ \) or \( /ae/ \)
  3. A Vowel Modifier Character: \( \text{ن} \rightarrow /\sim/ \)
  4. A Consonant Modifier Character: \( \text{ھ} \rightarrow /h/ \)

- For classes (1), (3) and (4), the mapping from graphemes to phonemes is one-to-one: a simple rule-based model can be developed.
The Urdu Script: Some Peculiarities

- Urdu has 4 different character classes:

1. Simple Consonant Characters, e.g. ﬀ → /f/
2. Dual (Consonant and Vocalic) Characters, e.g. 力 → /j/ or /ae/
3. A Vowel Modifier Character: ﻁ → /~/
4. A Consonant Modifier Character: ﺘ → /ʰ/

- For classes (1), (3) and (4), the mapping from graphemes to phonemes is one-to-one: a simple rule-based model can be developed.

- For class (2), context-sensitive rules were designed to account for the dual behavior.
The Urdu Script: Some Peculiarities

An excerpt from our scheme table:

<table>
<thead>
<tr>
<th>Unicode Urdu character</th>
<th>Latin letter in transliteration scheme</th>
<th>Phoneme</th>
</tr>
</thead>
<tbody>
<tr>
<td>ب</td>
<td>b</td>
<td>/b/</td>
</tr>
<tr>
<td>ب</td>
<td>p</td>
<td>/p/</td>
</tr>
<tr>
<td>ت</td>
<td>t</td>
<td>/t/</td>
</tr>
<tr>
<td>ئ</td>
<td>T</td>
<td>/t/</td>
</tr>
<tr>
<td>ج</td>
<td>j</td>
<td>/j/</td>
</tr>
<tr>
<td>چ</td>
<td>c</td>
<td>/ʧ/</td>
</tr>
</tbody>
</table>
The Transliterator: A Modular Approach

- Transliterator program: component-based approach
The Transliterator: A Modular Approach

- Transliterator program: component-based approach
- Pipeline implemented using 4 different modules
The Transliterator: A Modular Approach

- Transliterator program: component-based approach
- Pipeline implemented using 4 different modules
- Components may be used as standalone applications
The Transliterator: A Modular Approach

- Transliterator program: component-based approach
- Pipeline implemented using 4 different modules
- Components may be used as standalone applications
- Program implemented in C++
The Transliterator: A Modular Approach

- Transliterator program: component-based approach
- Pipeline implemented using 4 different modules
- Components may be used as standalone applications
- Program implemented in C++
  - Program development done at CRULP, Lahore, Pakistan
The Transliterator: A Modular Approach

- Transliterator program: component-based approach
- Pipeline implemented using 4 different modules
- Components may be used as standalone applications
- Program implemented in C++
  - Program development done at CRULP, Lahore, Pakistan
  - ASCII-based transliteration scheme devised in Konstanz
The Transliterator: A Modular Approach

- Transliterator program: component-based approach
- Pipeline implemented using 4 different modules
- Components may be used as standalone applications
- Program implemented in C++
  - Program development done at CRULP, Lahore, Pakistan
  - ASCII-based transliteration scheme devised in Konstanz
  - Integration in computational LFG grammar done in Konstanz
The Transliterator Pipeline

Input: Unicode Urdu Text

STEP 1: Normalization

STEP 2: Diacritization

STEP 3: Unicode to Urdu Zabta Takhti Conversion

STEP 4: Transliteration

Output: ASCII-based Scheme Transliteration
STEP 1: Normalization

- Unicode Arabic script: characters can be written in 2 ways
STEP 1: Normalization

- Unicode Arabic script: characters can be written in 2 ways
  - Composed form: as a single entity in Unicode block:
STEP 1: Normalization

- Unicode Arabic script: characters can be written in 2 ways
  - *Composed form*: as a single entity in Unicode block:
    
    *Alef madda*: ̣ā ā
STEP 1: Normalization

- Unicode Arabic script: characters can be written in 2 ways
  - **Composed form**: as a single entity in Unicode block:
    
    *Alef madda*: $\tilde{\text{a}}$
  
  - **decomposed form**: combined out of 2 or more characters:
STEP 1: Normalization

- Unicode Arabic script: characters can be written in 2 ways
  - Composed form: as a single entity in Unicode block:
    - *Alef madda*: َā
  - Decomposed form: combined out of 2 or more characters:
    - *Alef*: ٌā
STEP 1: Normalization

- Unicode Arabic script: characters can be written in 2 ways
  - *Composed form*: as a single entity in Unicode block:
    - *Alef madda*: \( \overline{\text{א}} \overline{\text{א}} \)
  - *decomposed form*: combined out of 2 or more characters:
    - *Alef*: \( \overline{\text{א}} \overline{\text{א}} \)
    - + lengthening diacritic *madda*: \( \overline{\text{א}} \overline{\text{א}} \)
Transliterator Architecture

STEP 1: Normalization

- Unicode Arabic script: characters can be written in 2 ways
  - *Composed form*: as a single entity in Unicode block:
    \[\text{Alef madda: } \bar{\text{a}}\]
  - *decomposed form*: combined out of 2 or more characters:
    \[\text{Alef: } \text{a} + \text{lengthening diacritic madda: } \bar{\text{a}}\]

- To avoid a duplication of rules, the input text is normalized to composed character form
STEP 1: Normalization

- Unicode Arabic script: characters can be written in 2 ways
  - Composed form: as a single entity in Unicode block:
    \[\text{Alef madda: } \overset{\sim}{\text{ā}}\]
  - Decomposed form: combined out of 2 or more characters:
    \[\text{Alef: } \overset{\sim}{\text{a}}\]
    \[\text{+ lengthening diacritic madda: } \overset{\sim}{\text{a}}\]
- To avoid a duplication of rules, the input text is normalized to composed character form

→ The system works on composed characters only!
STEP 2: Diacritization

- Problem: short vowel diacritics (*Aerabs*) usually not written in Urdu
STEP 2: Diacritization

- Problem: short vowel diacritics (*Aerabs*) usually not written in Urdu
  - *Aerabs* combine with simple consonants to indicate short vowels
STEP 2: Diacritization

- Problem: short vowel diacritics (*Aerabs*) usually not written in Urdu
  - *Aerabs* combine with simple consonants to indicate short vowels
  - *Aerabs* combine with dual behavior characters to indicate long vowels
STEP 2: Diacritization

- Problem: short vowel diacritics (Aerabs) usually not written in Urdu
  - Aerabs combine with simple consonants to indicate short vowels
  - Aerabs combine with dual behavior characters to indicate long vowels
- Our solution: lexicon lookup
STEP 2: Diacritization

- Problem: short vowel diacritics (*Aerabs*) usually not written in Urdu
  - *Aerabs* combine with simple consonants to indicate short vowels
  - *Aerabs* combine with dual behavior characters to indicate long vowels

- Our solution: lexicon lookup
  - Urdu lexicon data (80,000 diacritized words - gathered by CRULP in Lahore)
STEP 2: Diacritization

- Problem: short vowel diacritics (*Aerabs*) usually not written in Urdu
  - *Aerabs* combine with simple consonants to indicate short vowels
  - *Aerabs* combine with dual behavior characters to indicate long vowels
- Our solution: lexicon lookup
  - Urdu lexicon data (80,000 diacritized words - gathered by CRULP in Lahore)
  - Lexicon lookup: place diacritics in input words by looking up words in the lexicon
STEP 2: Diacritization

- Problem: short vowel diacritics (*Aerabs*) usually not written in Urdu
  - *Aerabs* combine with simple consonants to indicate short vowels
  - *Aerabs* combine with dual behavior characters to indicate long vowels

- Our solution: lexicon lookup
  - Urdu lexicon data (80,000 diacritized words - gathered by CRULP in Lahore)
  - Lexicon lookup: place diacritics in input words 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): national standard encoding for Urdu language processing
STEP 3: Unicode to Urdu Zabta Takhti Conversion

- Urdu Zabta Takhti (UZT): national standard encoding for Urdu language processing
  - Maps Unicode Urdu characters onto unique number sequences
STEP 3: Unicode to Urdu Zabta Takhti Conversion

- Urdu Zabta Takhti (UZT): national standard encoding for Urdu language processing
  - Maps Unicode Urdu characters onto unique number sequences
  - Developed as there was no standard industry codepage available
STEP 3: Unicode to Urdu Zabta Takhti Conversion

- **Urdu Zabta Takhti (UZT):** national standard encoding for Urdu language processing
  - Maps Unicode Urdu characters onto unique number sequences
  - Developed as there was no standard industry codepage available
  - Included in the pipeline for reasons of compatibility
STEP 3: Unicode to Urdu Zabta Takhti Conversion

- Urdu Zabta Takhti (UZT): national standard encoding for Urdu language processing
  - Maps Unicode Urdu characters onto unique number sequences
  - Developed as there was no standard industry codepage available
  - Included in the pipeline for reasons of compatibility

Example:

Urdu Unicode text

\[ \text{jābī} \]

UZT–converted text

898083120
STEP 4: Transliteration

- Convert number-based UZT format into our ASCII-based transliteration scheme
STEP 4: Transliteration

- Convert number-based UZT format into our ASCII-based transliteration scheme
- Transliteration rules are compiled into a Finite-State Machine – fast & efficient
STEP 4: Transliteration

- Convert number-based UZT format into our ASCII-based transliteration scheme
- Transliteration rules are compiled into a Finite-State Machine – fast & efficient

Example:

UZT–converted text
898083120 ˇc¯ab ¯ı
transliterated Latin letter-based notation
cAbI ˇc¯ab ¯ı 'key'
Loan words from Arabic and Persian include graphemes from these languages.
STEP 4 (cont.): Transliteration of Loan Graphemes

- Loan words from Arabic and Persian include graphemes from these languages
- These graphemes occur in loan words in Urdu
STEP 4 (cont.): Transliteration of Loan Graphemes

- Loan words from Arabic and Persian include graphemes from these languages
- These graphemes occur in loan words in Urdu

→ Result: multiple graphemes in Urdu can map to the same phoneme
STEP 4 (cont.): Transliteration of Loan Graphemes

- Loan words from Arabic and Persian include graphemes from these languages
- These graphemes occur in loan words in Urdu
  → Result: multiple graphemes in Urdu can map to the same phoneme
- Solution: Map genuine Urdu letter to general letter $s$; map foreign variants to $s_2, s_3$ etc.
STEP 4 (cont.): Transliteration of Loan Graphemes

- Loan words from Arabic and Persian include graphemes from these languages.
- These graphemes occur in loan words in Urdu.
  → Result: multiple graphemes in Urdu can map to the same phoneme.
- Solution: Map genuine Urdu letter to general letter s; map foreign variants to s2, s3 etc.

\[ ص ، ث ، س \rightarrow /s/ \]
STEP 4 (cont.): Transliteration of Loan Graphemes

- Loan words from Arabic and Persian include graphemes from these languages.
- These graphemes occur in loan words in Urdu.
- Result: multiple graphemes in Urdu can map to the same phoneme.

- Solution: Map genuine Urdu letter to general letter s; map foreign variants to s2, s3 etc.

\[
\begin{align*}
\ddot{s}, \dddot{s}, \mathring{s} & \rightarrow /s/ \\
\end{align*}
\]

- Most common, genuine Urdu character: \(\ddot{s} \rightarrow s\)
Loan words from Arabic and Persian include graphemes from these languages.
These graphemes occur in loan words in Urdu.

→ Result: multiple graphemes in Urdu can map to the same phoneme.

Solution: Map genuine Urdu letter to general letter $s$; map foreign variants to $s_2, s_3$ etc.

- Most common, genuine Urdu character: س → s
- Borrowed characters: ص، ث → s_2, s_3
STEP 4 (cont.): Transliteration of Loan Graphemes

- Loan words from Arabic and Persian include graphemes from these languages.
- These graphemes occur in loan words in Urdu.
  → Result: multiple graphemes in Urdu can map to the same phoneme.
- Solution: Map genuine Urdu letter to general letter $s$; map foreign variants to $s_2, s_3$ etc.
  
  ص، ث، س → /s/
  
  Most common, genuine Urdu character: س → s
  
  Borrowed characters: ص، ث → s_2, s_3
  
  → Lexicon is kept simple to read in most of the cases.
Evaluation of the Transliterator

- 1000 high frequency words collected from 18 million word Urdu corpus
Evaluation of the Transliterator

- 1000 high frequency words collected from 18 million word Urdu corpus
- Accuracy is near flawless if input is diacritized
Evaluation of the Transliterator

- 1000 high frequency words collected from 18 million word Urdu corpus
- Accuracy is near flawless if input is diacritized
- Accuracy is almost as good (0.07 difference) if input contains foreign words and no diacritics
Evaluation of the Transliterator

- 1000 high frequency words collected from 18 million word Urdu corpus
- Accuracy is near flawless if input is diacritized
- Accuracy is almost as good (0.07 difference) if input contains foreign words and no diacritics
- Performance of the transliterator:

<table>
<thead>
<tr>
<th>Test Corpus Size</th>
<th>$A = \frac{C_w}{T_w}$ (diacritized input)</th>
<th>$A = \frac{C_w}{T_w}$ (input without diacritics, with foreign words)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.995</td>
<td>0.925</td>
</tr>
</tbody>
</table>

Table: Accuracy Results for Transliterator
The Architecture of the Grammar

The transliterator is integrated into a parsing architecture using a Finite-State Morphological Transducer (FSMT) and the XLE Grammar Development Platform (XLE).
The Architecture of the Grammar

The transliterator is integrated into a parsing architecture using a Finite-State Morphological Transducer ($\text{FSMT}$) and the XLE Grammar Development Platform ($\text{XLE}$).

Transliterator (Urdu & Hindi Unicode to ASCII-Based Transliteration) \[ \Downarrow \]
Tokenizer \[ \Downarrow \]
Morphology ($\text{FSMT}$) \[ \Downarrow \]
Syntax (C- and F-Structure) ($\text{XLE}$)
Integrating the Transliterator

→ Transliterator applies first
Integrating the Transliterator in the ParGram Urdu Grammar

Integrating the Transliterator

→ Transliterator applies first

Example (gARI call ‘The car worked//started.’)

transliterator input:

gārī ʧālī
gARI ʧalI

transliterator output:

gārī ʧālī
gARI ʧalI
Integrating the Transliterator (cont.)

→ Transliterator output feeds in XLE tokenizer
Integrating the Transliterator in the ParGram Urdu Grammar

Integrating the Transliterator (cont.)

→ Transliterator output feeds in XLE tokenizer

Example (gARI call ‘The car worked/started.’)

tokenizer input:
gARI calI

tokenizer output:
gARI TB calI TB
Integrating the Transliterator (cont.)

→ Transliterator output feeds in XLE tokenizer

Example (gARI call ‘The car worked/started.’)

tokenizer input:

gARI call

tokenizer output:

gARI TB call TB

gārī ḡalī
gārī ḡalī

→ Tokenizer output feeds in FST morphological transducer
Integrating the Transliterator (cont.)

→ Transliterator output feeds in XLE tokenizer

Example \((gARI \textit{ call} \text{ 'The car worked/started.'})\)

tokenizer input:
\(gARI \text{ call}\)

tokenizer output:
\(gARI \text{ TB \ call TB}\)

→ Tokenizer output feeds in FST morphological transducer

Example \((gARI \textit{ call} \text{ 'The car worked/started.'})\)

morphology output:
\(gARI+Noun+Fem+Sg\)
\(calI+Verb+Perf+Fem+Sg\)
Integrating the Transliterator (cont.)

→ Morphology output feeds in XLE syntactic rules
Integrating the Transliterator in the ParGram Urdu Grammar

Integrating the Transliterator (cont.)

→ Morphology output feeds in XLE syntactic rules

Example (gARI call ‘The car worked/started.’)

Morphology Output/Syntax input:

- gARI+Noun+Fem+Sg
- calI+Verb+Perf+Fem+Sg

Syntax output (C-Structure and F-Structure):

CS 1: ROOT

<table>
<thead>
<tr>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>KP</td>
</tr>
<tr>
<td>NP</td>
</tr>
<tr>
<td>N</td>
</tr>
</tbody>
</table>

"gARI calI"

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


Thank you!

Are there questions?