Developing a Finite-State Morphological Analyzer for Urdu and Hindi

Tina Bögel, Miriam Butt, Annette Hautli, Sebastian Sulger

Universität Konstanz

14th September, 2007

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)
- together with Hindi the second/third most spoken language in the world (316 Million speakers; Graddol 2004)
- written with an Arabic-based script.

The ParGram Project

We have been working on an LFG (Lexical-Functional Grammar; e.g., Dalrymple 2000) Grammar for Urdu as part of the ParGram (Parallel Grammar) project (Butt and King 2007).

- Large-scale grammars currently exist for: English, French, German, Japanese and Norwegian.

The ParGram Project

We have been working on an LFG (Lexical-Functional Grammar; e.g., Dalrymple 2000) Grammar for Urdu as part of the ParGram (Parallel Grammar) project (Butt and King 2007).

- Large-scale grammars currently exist for: English, French, German, Japanese and Norwegian.
- Smaller-scale grammars include: Welsh, Turkish, Malagasy, Chinese (and Urdu).

The ParGram Project

We have been working on an LFG (Lexical-Functional Grammar; e.g., Dalrymple 2000) Grammar for Urdu as part of the ParGram (Parallel Grammar) project (Butt and King 2007).

- Large-scale grammars currently exist for: English, French, German, Japanese and Norwegian.
- Smaller-scale grammars include: Welsh, Turkish, Malagasy, Chinese (and Urdu).
- Like all of the other ParGram grammars, the Urdu Grammar relies heavily on a *finite-state morphology* that interfaces with the syntactic rules.

Xerox Finite-State Tools

- Most of the ParGram grammars use the Xerox Finite-State tools described in Beesley and Karttunen (2003).
- Our development work so far has shown that the finite-state tools and solutions in Beesley and Karttunen (2003) prove to be more than adequate to meet the challenges posed by Urdu.
- We report here on some of the more interesting challenges:
  - Script transliteration
  - Tokenization (the Urdu future)
  - Reduplication

Urdu Resources

- Very few computational resources exist for Urdu (and other Indian languages).
- Fonts, Corpora, Taggers, Morphological Analyzers, etc. all are just being developed (e.g., see `http://www.crulp.org/` for some resources).
- As part of the Urdu ParGram project, we therefore have to develop our own finite-state morphological analyzer.
- We connect up the morphological analyzer to the syntax via the morphology-syntax interface (Kaplan et al. 2004) defined for LFG.

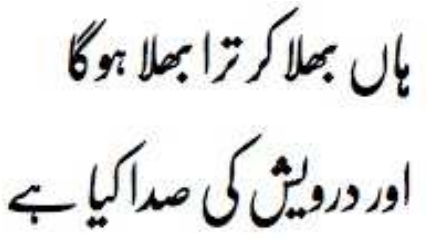
Urdu and Hindi Scripts

- Recall that Urdu and Hindi are structurally almost identical.
- Any morphological analyzer developed for Urdu can therefore in principle also be used for Hindi (and vice versa).
- **Problem:** The scripts for Urdu and Hindi differ absolutely.
  - Urdu: version of the *Arabic* script (Unicode fonts have only recently been developed, Rahman and Hussain 2003).
  - Hindi: *Devanagari*, a phonetic-based script passed down over the millenia from Sanskrit.
  - Urdu is written right-to-left, Hindi left-to-right.

Urdu and Hindi Scripts

The following illustrates the same couplet (162,9) from the poet Mirza Ghalib (1797–1869)

- **Urdu** vs. **Hindi**



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

- **Common Transliteration in Roman 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.'

Transliteration

- We use Glassman's (1977) transliteration system for our Urdu grammar and morphological analyzer.
  - Capitalized vowels indicate length
  - H marks aspiration
  - N indicates nasalization
  - S stands for ʃ
  - other capitalized consonants indicate retroflexes
- **Goal:** Use the common transliteration scheme to parse/generate both Urdu and Hindi.

Transliteration

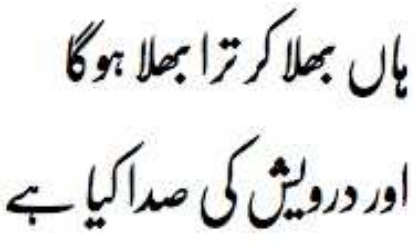
- **Current:** Abbas Malik (2006) has used the XFST tools to implement HUMTS (Hindi-Urdu Machine Transliteration System).
  - Cascade of finite-state transducers.
  - Takes Urdu or Hindi input, transliterates into a common ASCII base and generates back out either Urdu or Hindi (regardless of what the input was).
- **To Do:** Integrate HUMTS into our system.
- **Note:** Other projects are adopting the same general strategy of transliterating the different South Asian language scripts into a common underlying ASCII representation, e.g., Humayoun et al. (2007).

Identifying Word Boundaries

- Any transliterator working on Arabic-based scripts also has to deal with the very serious problem of identifying word boundaries.
- This problem is notorious and will not be discussed here (for some discussion of problems with Urdu, see Abbas Malik (2006)).
- Beyond this, when dealing with both Urdu and Hindi simultaneously, difficulties arise because the scripts do not always agree on what a word ist.
- **One Illustrative Example:** The Urdu/Hindi future.

Urdu/Hindi Future

- An example is found in our Ghalib couplet: the rendition of *hOgA* 'he/it will be'.
- **Urdu** vs. **Hindi**



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

- **Common Transliteration in Roman 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.'

Urdu/Hindi Future

- Why this difference in opinion between the scripts?
- Let's take a closer look at the Urdu/Hindi future paradigm.

Urdu Future Paradigm

	Singular M/F	**Plural** M/F	**Respect** (Ap) M/F	**Familiar** M/F
1st	mAr-UN-g-A/I	mAr-EN-g-E/I		
2nd	mAr-E-g-A/I		mAr-EN-g-E/I	mAr-O-g-E/I
3rd	mAr-E-g-A/I	mAr-EN-g-E/I		

mAr- 'hit'

A Complex Morphological Structure:

1. The stem (*mAr*)
2. Number/person (*UN/E/EN/O*) (identical to subjunctive)
3. 'g' as future morphology
4. Number/gender morphology (*A/I/E*)

Urdu/Hindi Future

- **Question:** Why does one need to mark number twice?
- **Answer:** Historical accident.
  - The future morphology is the result of reanalysis from an old periphrastic construction involving the word 'go' (Sanskrit *gā*).
  - A participle of 'go' combined with a main verb to give a future meaning — similar to the English 'I am **going** to eat.'
  - The old 'go' participle was inflected for number and gender (like all participles and adjectives).
  - The number/person morphology is actually how you mark subjunctive tense (inherited from Middle Indo-Aryan).
  - These present/subjunctive forms also gave rise to *contingent future* readings and these forms were used as a basis for the new, improved future.
- So one ended up marking number twice as part of a process of historical reanalysis.

Urdu/Hindi Future

Comparing the current Future and Subjunctive Paradigms

Urdu Future Paradigm

	Singular M/F	**Plural** M/F	**Respect** (Ap) M/F	**Familiar** M/F
1st	mAr-UN-g-A/I	mAr-EN-g-E/I		
2nd	mAr-E-g-A/I		mAr-EN-g-E/I	mAr-O-g-E/I
3rd	mAr-E-g-A/I	mAr-EN-g-E/I		

mAr- 'hit'

Urdu Subjunctive Paradigm

	Singular	**Plural**	**Respect** (Ap)	**Familiar**
1st	mAr-UN	mAr-EN		
2nd	mAr-E		mAr-EN	mAr-O
3rd	mAr-E	mAr-EN		

mAr- 'hit'

Urdu/Hindi Future

- In Hindi, this historical development has been "forgotten" and the *ga* morpheme is treated as part of the word.
- In Urdu, the *ga* is still treated as a separate word.
- In both languages all the pieces of morphology involved nevertheless perform exactly the same function as far as syntax and semantics is concerned.
- From the perspective of syntax, our morphological analyzer therefore should treat them in parallel.

The Future at the Morphology/Syntax Interface

Our Current Solution:

- Turn the Urdu input of two separate words into one word and analyze that.

Sample FSM Analysis

mArEgI ⇔
mAr+Verb+Subjunct+2P+Sg+Fut+Fem
mAr+Verb+Subjunct+3P+Sg+Fut+Fem

- Identifying *gI/gE* as part of the previous word should pose no problem as they are not words in their own right.

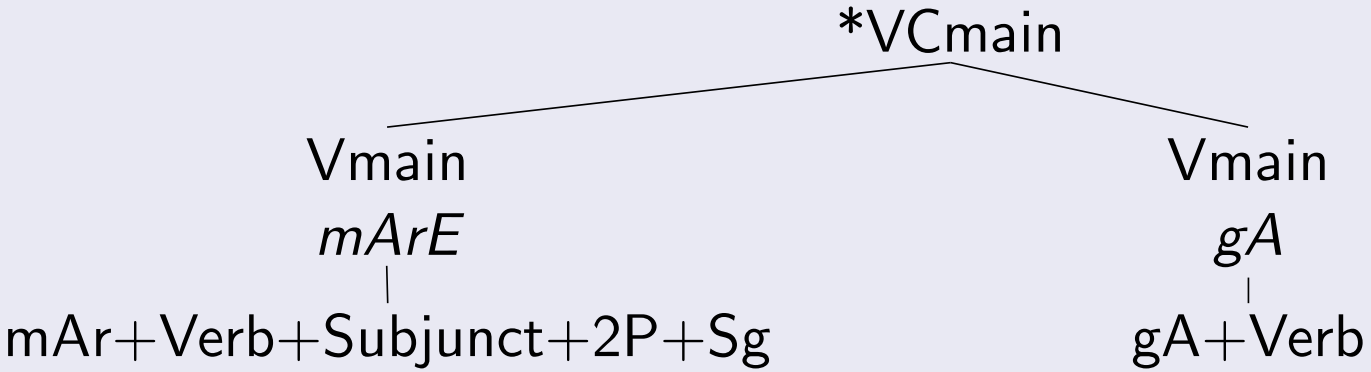
The Future at the Morphology/Syntax Interface

A Mixed Solution:

- But *gA* is a word in its own right (it can also mean 'sing').
- So it might lead to unnecessary errors to simply always identify *gA* as part of the previous word.
- Solution: output both possibilities from the morphological analyzer:
  1. *mArEgA*
  2. *mArE gA*
- And let syntax take care of ruling out illicit configurations.

The Future at the Morphology/Syntax Interface

Syntactically Unlicensed Configuation



- That is, for *gA* the FSM will produce ambiguous output to be filtered by the syntactic parser. **[Demo]**
- *gE/gI*, on the other hand, are dealt with completely within the morphological analyzer and tokenizer — they are not words known to the syntax.

The Morphology-Syntax Interface

There are, of course, several other issues that arise with respect to the morphology-syntax interface.

Some Fundamental Issues:

- Is all the information coming from the morphology relevant for the syntax? — Example: **Urdu/Hindi Future**
- Is the information coming from the morphology sufficient for syntactic purposes? — Example: **Urdu/Hindi Infinitives**

Back to the Urdu/Hindi Future

Recall that we analyze Urdu/Hindi future forms as follows:

Sample Analysis of Urdu/Hindi Future

mArUNgI ⇔
mAr+Verb+Subjunct+1P+Sg+Fut+Fem

- This is because the future *g*- morphology has been historically added to an originally subjunctive form.
- So in terms of morphology-internal analysis, we are doing the correct thing.
- However, from the perspective of syntax, we are delivering useless information since every future form is also subjunctive.

Making a Strategic Decision

- There might be other clients for the Urdu FSM, so it is probably not wise to simply throw away the +Subjunctive tag.
- On the other hand, experience in ParGram (Butt et al. 1999) has shown that it is better to eliminate tags of this kind, since dealing with them complicates the morphology-syntax interface.

Solution using Rewrite Rules and Composition

- For our purposes an easy solution is offered by XFST: **Rewrite Rules**
- The output of the morphological analyzer is composed via the `.o.` **composition operator** with a regular expression containing a rewrite rule (Beesley and Karttunen 2003).

Rewrite Rule

```
read regex ["+Subjunct" -> 0 || ___ ?* "+Fut"]
```

- This rule rewrites the +Subjunct tag as $\emptyset$ exactly when a +Fut tag is encountered as well.

Result of Composition with Rewrite Rule

mArUNgI $\Leftrightarrow$
mAr+Verb+1P+Sg+Fut+Fem

Urdu/Hindi Infinitives

- In Urdu/Hindi all infinitives can also be used as nouns.
- **Question:** Should the morphological analyzer produce just one analysis?

Current Analysis of Urdu Infinitives

dEkHnA ⇔
dEkH+Verb+Inf+Masc+Sg

- Or two analyses?

Alternative Possible Analysis of Urdu Infinitives

dEkH+Verb+Inf+Masc+Sg
dEkH+Noun+Deverb+Masc+Sg

Strategic Decision

- Producing two analyses for each infinitive in Urdu will introduce ambiguity into the morphological analyzer and add an extra path for each verb.
- This could be avoided by giving the +Inf tag two different interpretations at the morphology-syntax interface.

Two Interpretations of the +Inf tag

```
+Inf      INF_SFX    xle (^ VFORM) = inf;
          DEVERB_SFX xle (^ NTYPE) = deverbal.
```

- And then writing a sublexical rule that allows for a deverbal noun.

Possible Analysis

- Output from Morphological Analyzer:

Current Analysis of Urdu Infinitives

dEkHnA ⇔
dEkH+Verb+Inf+Masc+Sg

- Sublexical Rule to interpret the output as a deverbal noun

Sublexical Rule for Deverbal Nouns

```
N -->   NOUN-S_BASE          "the noun stem"
        V-T_BASE             "we actually have a verb here"
        DEVERB_SFX_BASE      "processing the +Inf tag"
        GEND_BASE            "processing the +Gend tag"
        NUM_BASE             "processing the +Num tag"
        (N-REDUP_BASE).      "possibly reduplicated"
```

Urdu Infinitives

- Recognizing that infinitives may act as deverbal nouns can be done
  - Either in the Morphological Analyzer
  - Or as part of the morphology-syntax interface
- We are currently experimenting with these two possiblities, but favor the latter.
- This is because morphologically the infinitives simply are infinitives — it is the syntax/semantics that presses them into service as nouns.

Reduplication — The Phenomenon

- Another issue at the morphology-syntax interface is reduplication.
- Two reduplication patterns:
  - Onset of word is permutated (echo reduplication), e.g. dEkHnA vekHnA 'seeing and such things'
  - Word is simply repeated (full word reduplication), e.g. tanhA tanhA 'very lonely (lonely lonely)'
- Why is Urdu doing this? Answer: Reduplication is used to add some meaning to the word:
  - Strengthen/emphasize the meaning of adjectives
  - Express something like *and those kinds of things*, thus relativise, the meaning of nouns and verbs

The Script/Morphology Interface

- How is this reflected in the text?
- In texts, reduplicated word forms are realized as two separate words.
- Transliteration yields two words separated by white space.
- Current Approach: Treat reduplicated forms as multiword expressions (cf. New York).
- However — this is tricky because reduplication is a productive process: not all reduplicated forms can/should be listed.

Analysis

Current Analysis of Reduplication

caInA vaInA ⇔
cal+Verb+Inf+Masc+Sg+Redup
'walking and such things'
kHAnA vAnA ⇔
kHAnA+Noun+Masc+Sg+Redup
'food and those kinds of things'
tHanDA tHanDA ⇔
tHanDA+Adj+Masc+Sg+Redup
'ice cold (cold cold)'

- We mark reduplicated forms with the feature +Redup.

Our solution

- Our implementation of reduplication is by and large adopted from Beesley and Karttunen (2003); our system proves that their approach works out quite nicely as far as morphology is concerned.
- However, our system needs deal with the white space, which is a challenge for the morphology-syntax interface.

Our solution

The Implementation, step by step:

- Multicharacter brackets `^[` and `^]` are used to mark the domain of reduplication; in addition, the right bracket is marked with character `^2`, representing the regular expression for reduplication.
- This means that the lexicon size is doubled.
- It also means that we might get unwanted bracketing, e.g. brackets on only one side of the string, cf. `^[{kitab} or {kitab}^2^]`

Our solution

- A Bracket Filter is applied to allow only for those strings which have the correct bracketing (brackets on each side of the string or no brackets at all).

bracketfilter.regex

```
[ ~[ ?* "^[" ~$["^]"] ] & ~[ ~$["^["] "^]" ?* ] ];
```

- This is taken from Beesley & Karttunen (2003).
- Result: only strings of the form `{kitab} / ^[{kitab}^2^]`

Our solution

- The multicharacter symbol %^Hyphen is introduced in the lexicon source file; two words are internally represented as being connected by a hyphen, namely this symbol
  - Example: ^[{kitab%^Hyphen}^2^]
- On the upper side of the transducer, the feature +Redup is added.
- Compile-Replace is applied to resulting network, treating the marked up lower side as a regular expression which is to be interpreted, thus translating ^[{...%^Hyphen}^2^] into well-formed strings of this type: {...%^Hyphen}{...%^Hyphen}

Our solution

- The simple rule hyph.regex is then applied to replace the hyphen multicharacter symbol %^Hyphen by a white space; it is also used to delete the unwanted second instance of %^Hyphen created by the reduplication.

hyph.regex

```
[%^Hyphen -> 0 || %^Hyphen ?* _] .o. [%^Hyphen -> " "];
```

- This rule, with an input of the form {...%^Hyphen}{...%^Hyphen}, returns {...} {...}
- The grammar can decide how to use the +Redup feature from the upper side (or whether to use the very subtle semantic information implied by reduplication at all).

The Morphology/Syntax Interface

- The tokenizer is thus faced with the task to recognize a reduplicated item as such, and not as two completely independent words.
- Within XFST, Reduplication works fine; the problem is how to integrate this into the syntax.
- The ultimate solution will include fitting reduplication into the XFST tokenizer; however, further research has to be done how to achieve this

Echo Forms

- In echo reduplication, the first onset of the reduplicated form undergoes phonological change:
  - If the word begins with a consonant, this consonant is replaced by v
  - If the word begins with a vowel, a v is inserted as onset.
  - So far, all the words in our lexicon have just simple consonants as onsets - this seems to be a strong tendency, if not a hard phonotactic constraint of Urdu.
- Principally every content word can be "echoed".

Echo Forms

- The phonological rule to account for echo reduplication is as follows:

reduprules.regex

```
Cons -> v || ?* %^Hyphen _ ?* "@P.ECHO.v@"
.o. [[..] -> v || ?* %^Hyphen _ Vowels ?* "@P.ECHO.v@"]
```

- The flag diacritic "@P.ECHO.v@" is used simply as a condition to trigger the rule here.
- This maps *calnA calnA* to *calnA valnA* and *kitab kitab* to *kitab vitab*.

Problems

- Encountering a reduplicated item in normal text, analyzing it as such will be quite difficult as reduplication is a productive process.
- The Question is how to integrate it in the tokenizer (as opposed to lexicalized multiword expressions such as New York).

Remaining Work on Reduplication

- Include reduplicated forms in the tokenizer
- Reduplication **[Demo]**

Summary

- Some problems building a FSM analyzer for Urdu and integrating it into the syntax have been presented.
- An analogous treatment of both Urdu and Hindi can be achieved by our approach, as a common ASCII transliteration is used.
- Urdu/Hindi Future and Reduplication have been explored and integrated in our system
- The problems that arise from these structures for the morphology/syntax interface (defined between finite-state morphological analyzers and LFG grammars as part of the ParGram project) have been formulated.

References I