Developing a Finite-State Morphological Analyzer for Urdu and Hindi

Tina Bögel, Miriam Butt, Annette Hautli, and Sebastian Sulger

Universität Konstanz

Abstract. We introduce and discuss a number of issues that arise in the process of building a finite-state morphological analyzer for Urdu, in particular issues with potential ambiguity and non-concatenative morphology. Our approach allows for an underlyingly similar treatment of both Urdu and Hindi via a cascade of finite-state transducers that transliterates the very different scripts into a common ASCII transcription system. As this transliteration system is based on the XFST tools that the Urdu/Hindi common morphological analyzer is also implemented in, no compatibility problems arise.

1 Introduction

As part of the ParGram (Parallel Grammar) project [1], [2], we are developing a grammar for the South Asian language Urdu.[1] Very few resources exist for this language, in particular, no broad-coverage finite-state morphological analyzer exists to date. Part of the Urdu Grammar project is therefore to build a finite-state morphological analyzer for Urdu and to connect it up with the syntax via the morphology-syntax interface [3] defined for Lexical-Functional Grammar (LFG; [4]).

Current features of the Urdu ParGram project in the context of parallel grammar development have already been discussed elsewhere [5]. In this paper, we focus on some issues that have arisen with respect to the morphological analyzer in particular. All the (larger) ParGram grammars to date include a finite-state morphological analyzer that interfaces with the syntax. These morphological analyzers are generally built with the Xerox finite-state technology tools and follow the methodology established by [6]. The finite-state tools and the solutions already proposed by [6] prove to be more than adequate to meet the challenges posed by Urdu. However, some interesting issues do arise with respect to 1) the script and tokenization (section 2); 2) reduplication (section 3) ; 3) potentially ambiguous information at the morphology-syntax interface (section 4).

2 Two Different Scripts, One Representation

Urdu is structurally almost identical to Hindi. The major difference is that the vocabulary of Urdu bears more Persian/Arabic influences, while the vocabulary

[1] Thanks go to Tafseer Ahmed for helping us understand some issues with respect to the script and the morphology.

of Hindi is more Sanskrit based. Both are ultimately descended from a version of Sanskrit (i.e., are Indo-European). Urdu as a separate version of the language came into being when the Moghuls invaded the Indian subcontinent. The language of their court was Persian, which came into contact with a local language generally referred to as Hindustani (or Hindi). The very Persianized version of this language came to be known as Urdu.[2]

This brief historical sketch is of relevance because lexical items borrowed in from Persian tend to behave differently (i.e., have different inflectional possibilities). However, questions of lexical and morphological origin tend to be minor issues. A more major issue is that Urdu and Hindi are written in very different scripts. Urdu is written with a version of the Arabic script.[3] Hindi, in contrast, is written in *Devanagari*, a phonetic-based script passed down over the millenia from Sanskrit.

2.1 A Common Transliteration System

(1) and (2) show a couplet (162,9) from the poet Mirza Ghalib (1797–1869): (1) is written in Urdu, (2) is the same couplet, but written in Devanagari (Hindi). Note that Urdu is written right-to-left, whereas Hindi is written left-to-right.

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

(1)

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

(2)

Although the two writing systems differ markedly, the languages they encode are structurally almost identical. Given this fact, our general strategy in building a morphological analyzer is to produce a resource that can be used for text written in both Urdu and Hindi. This involves building a transliteration system that goes from whichever script is being processed to a common ASCII base and then being able to generate back out from the common ASCII base to either one of the scripts. That is, both the texts in (1) and (2) are rendered as in (3).

[2] Modern Hindi naturally also bears traces of language contact with Persian, but not as markedly as Urdu.

[3] Unicode fonts for this script have only recently been developed (e.g., see `http://www.crulp.org`; [7]).

(3) 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.'

Our transliteration is based on proposals by [8]. Capitalized vowels indicate length, H marks aspiration, N nasalization, S stands for ∫ and other capitalized consonants indicate retroflexes.

A transliterator in accordance with our overall strategy has been implemented by [9]. Malik's HUMTS (Hindi-Urdu Machine Transliteration System) is written as a cascade of finite-state transducers that transliterate from the Urdu and Hindi scripts to SAMPA [10], a common underlying phonetic ASCII alphabet, and back out from SAMPA to the two differing scripts. SAMPA has been developed to enable coverage of all the world's languages; however, for the purposes of Urdu, it is unwieldy and very difficult to read. In integrating Malik's work into the Urdu grammar, we will therefore use Glassman's transliteration system. Beyond the simple conversion of letters that is necessary to do this, we anticipate no further (major) problems as HUMTS was written with the same XFST tools used in our Urdu grammar project.[4]

2.2 Future Morphology: Illustrating Tokenization Problems

Writing a transliterator that takes one script as an input and is able to output another script is not an easy task. Many of the problems that arise are discussed in Malik's work. In terms of the Urdu Grammar, most relevant to us are problems of tokenization. In particular, problems associated with the future morphology in Urdu/Hindi was one of the first to arise.

We already had an example of future usage in (1) and (2). An inspection of each example will quickly reveal one of the very general problems in dealing with the Urdu script: while in Hindi, each word is clearly demarcated and easy to identify, in Arabic-based scripts in general, word boundaries are very difficult to identify. One must basically know the language (i.e., be able to access the lexical items) in order to be able to read the script.[5]

Beyond this very general problem, the scripts also encode differences of opinion as to what exactly a word is. This is illustrated in (1) and (2) with respect to the future form of 'be' *hOgA*. In (1) it is expressed by the last two letter groups on line one (reading from right to left). In (2), the form is expressed by just one letter group: the last one (reading from left to right) on line 1. This difference in encoding reflects an on-going historical change.

The future in Urdu/Hindi is formed as shown in the paradigm (4) for the stem *mAr* 'hit/kill'. The stem is followed by information about person and number

[4] Related work has been done by [11], who provide a transliterator into ASCII as well, but do morphological analysis using the Functional Morphology Toolkit [12].

[5] The same is not true for Devanagari, which, being phonetically based, allows a sounding out of the words.

(*UN/E/EN/O*), to which the future marker *g* is attached. This, finally, is followed about information about number and gender.

(4) **Urdu Future Paradigm**

	Singular **M/F**	**Plural** **M/F**	**Respect** (Ap) **M/F**	**Familiar** (tum) **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'

The future paradigm is thus a relatively complex assemblage of morphological pieces. The person/number morphology is identical to that used in the subjunctive paradigm, shown in (5). To these essentially subjunctive forms, a *-g-* is attached to mark the future. The consensus in the available literature is that the future *-g-* is derived from a Sanskrit participle of the verb *gā* 'go' [13], [14]. This analysis immediately explains the gender and number agreement morphology (*A/I/E*) exhibited by the future: Participles functioned like adjectives and so generally had number and gender agreement morphology. This morphology has simply been retained in all the verb forms in Urdu/Hindi that derive from old participles (i.e., the perfect, imperfect and progressive forms), including the future.

(5) **Urdu Subjunctive Paradigm**

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

mAr- 'hit'

The old participle of the verb *gā* 'go' used to form its own word. Indeed, as recently as a century ago, clitics like the emphatic *hI* 'even/only' could intrude between the *-g-* and the stem+subjunctive morphology. This is illustrated in (6).

(6) kɑh-ũ=hi=ga
say-1.Sg=Emph=Fut.M.Sg
'I will say (it), of course.' (Hindi, from Kellogg 1893:§399)

These examples suggest that while the old participle was no longer functioning as an independent word a century ago, it retained some prosodic independence and was probably functioning as a clitic (indicated by the glossing with '='). This is entirely consonant with well known processes of historical change whereby words are reanalyzed as clitics and then reanalyzed further as inflectional morphology as they move from expressing content words to functional elements (e.g., [15], [16]).

The examples in (6) are only marginally possible in modern Urdu, whereas speakers of Hindi tend to reject them outright. This difference in native speaker

judgements may or may not be correlated with the differences encoded in the writing system. Recall that in written Hindi, the future is expressed in one word together with the subjunctive stem. In Urdu however, the stem+subjunctive and the future+number+gender are generally written as two separate words.

In both languages all the pieces of morphology involved nevertheless perform exactly the same function, so our morphological analyzer should treat them in parallel. In the morphological analyzer, the future *-g-* is treated as an inflectional morpheme and a form like *mArEgI* would be analyzed as in (7).

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

The tokenizer thus has to turn the Urdu input of *mArE gI* into *mArEgI*. This in and of itself does not present a problem, since the deletion of white space is not a problem. In principle, since forms like *marE* are also words in their own right, a serious ambiguity problem could arise. However, as *gI/gA/gE* are not words in their own right,[6] we do not anticipate serious problems with our basic approach.

In sum, the future morphology discussed here provides a good example of the potentially problematic factors that must be dealt with. Another, perhaps more interesting problem posed by Urdu is that of reduplication.

3 Reduplication

Urdu/Hindi, like most of the South Asian languages, tends to use reduplication quite frequently [17]. All content words can generally be reduplicated and the effect of the reduplication is to either strengthen/emphasize the original word or to express something like "and those kinds of things".

(8) a. kHAnA vAnA
food.M.Sg. Redup
'food and those kinds of things'
b. tHanDA tHanDA
cold.M.Sg. Redup
'ice cold (cold cold)'
c. kHAtA vAtA
eat.Impf.M.Sg Redup
'he is eating and such'

There are two different kinds of reduplication strategies. In the one illustrated by (8a), the onset of the content word is replaced with another consonant. This consonant could be either *v*, *ṯ* (T) or ∫ (S). In another strategy ((8b)), the word is simply repeated. We will refer to this latter strategy as *full word reduplication*, the former strategy is generally described as *echo formation* or *echo reduplication*.

[6] *gA* is a word, namely the bare form of the verb 'sing'. However, this would never (or rarely) occur in conjunction with a subjunctive verb.

3.1 General Strategy

Generally, reduplications are written as seperate words in both Urdu and Hindi. The fundamental problem facing the tokenizer is thus the fact that a reduplicated item must be recognized. The transliteration system will yield two words, as shown in (9), for example, which are separated by white space.

(9) calnA valnA
walk.Inf.M.Sg Redup
'walking and such things'

Our morphological analyzer basically follows the solution for full stem reduplication presented by [6] for Malay. The basic lexicon built independently of reduplication for nouns, verbs, adjectives and other content words interacts with reduplicating regular expressions.

The morphological analysis of reduplications as in (9) is shown in (10). That is, within the morphological analyzer, the reduplicated form is simply registered via the tag +Redup and is passed on as such to the Urdu grammar, which can decide how to use this information (or whether to use the very subtle semantic information implied by reduplication at all).

(10) cal+Verb+Inf+Masc+Sg+Redup

In the Malay example presented by Beesley and Karttunen (B&K), the original word and the reduplicated part are merged into a single word. Our implementation differs from theirs in that we need to deal with the white space. Currently, we do this by introducing the multiword `%^Hyphen` into the *lexc* source file (which encodes the basic lexicon plus the morphological continuation classes). When dealing with reduplication, we thus internally represent the two words involved as being connected with a hyphen.

Reduplication itself is managed, as in B&K, via the introduction of the multicharacter brackets `"^["` and `"^]"` in order to mark the domain of reduplication. The right bracket is additionally marked with the characters ^2. The lower side of the finite-state network thus ends up being marked up via the brackets `"^["` and `"^2^]"`. As discussed in B&K, the *compile-replace* algorithm can be applied to the resulting network — compile-replace essentially treats the marked up lower side as a regular expression which is to be interpreted. The overall effect is that something like *calnA* ends up being doubled to *calnA-calnA* due to the ^2 specification (and the addition of the hyphen).

We illustrate our approach more concretely with respect to just the adjective 'strange' in terms of full word reduplication. The code illustrates a simple *lexc* file which allows for two possibilities for all adjectives. In one, a bracketing is begun which is intended for the reduplicated version. This is notated by the regular expression ^2, which results in the doubling of the material delimited by the brackets. The bracket filter from B&K removes any unmatched brackets that

may have resulted from paths which contain only one bracket.[7] The bracket filter and the lexc file are composed, and the compile-replace algorithm is applied to the resulting network. Compile-replace translates the reduplication [...]^2 into well-formed strings of this type: [...]%^Hyphen[...]%^Hyphen. In a last step a regular expression (illustrated below as hyph.regex) then replaces the hyphens (%^Hyphen) used for internal management of the reduplicated forms with a white space.

```
* !AdjRedup.txt, lexc file just for ajIb 'strange'
* Multichar_Symbols
* +Adj +Unmarked +Redup +Intensifier
*
* Lexicon Root
* 0:^[[{ Unmarked ;
*        Unmarked;
*
* Lexicon Unmarked
* ajIb Ending ;    !the adjective 'strange'
*
* Lexicon Ending
* +Adj+Unmarked+Redup+Intensifier:}%^Hyphen]^2^]  # ;
* +Adj+Unmarked:0 # ;
*********************************************************
* ! bracketfilter.regex --- bracket filter from B&K
* [ ~ [ ?* "^[" ~$["^]"] ] & ~[ ~$["^["] "^]" ?* ] ];
*********************************************************
* !hyph.regex, removes '%^Hyphen' and inserts a white space
* [ %^Hyphen -> 0 || %^Hyphen ?* _ ]
* .o.
* [ %^Hyphen -> " " ] ;
```

3.2 Echo Reduplication

Recall that echo reduplication further requires the use of a different consonant/onset in the reduplicated form ((11)). In order to deal with this further complication, we introduce replace rules to effect the phonological change and further make use of flag diacritics (`@P.ECHO.v@` in the rules below, cf. B&K) in order to flag that the echo type of reduplication has taken place.

(11) AlU vAlU
potato.M Redup
'potatoes and such'

[7] This can be done differently, by controlling the continuation paths of the lexc file more tightly, however, in the long run, this results in a conceptually more complex structure of the lexc file and it is thus preferable (and more efficient) to simply apply the bracket filter on unwanted paths.

The phonological replace rules shown below exemplify just two cases. In reduplicating contexts (i.e., contexts which have been marked up by a Hyphen), either the first consonant[8] is replaced by a *v*, or if there is no onset as in (11), a *v* is inserted. We have formulated similar rules for reduplications with ṯ (T) or ʃ (S).

`Cons` stands for the set of consonants (this is predefined). The phonological replacement rule below thus operates on Consonants or Vowels (listed here individually, though this could also be done differently). Consonants are replaced by a *v* (or T or S in the rules not shown here). If there is no consonant, then a *v* (or T or S) is inserted before the vowel.

```
Cons -> v || ?* %^Hyphen _ ?* "@P.ECHO.v@"
.o.
a -> v a , e -> v e , i -> v i, o -> v o,
u -> v u || ?* %^Hyphen _ ?* "@P.ECHO.v@";
```

We thus implement the two differing reduplication strategies by using a range of FST methodologies. Full word reduplication is treated via a markup that feeds into the compile-replace algorithm. Echo reduplication additionally requires the use of phonological replace rules and flag diacritics.

Overall, allowing for reduplication results in a threefold increase of the basic lexicon. However, this increase is dealt with in a conceptually elegant manner and can be achieved by writing just a few extra lines of code (regular expressions) that are composed with the source lexc file. In our approach, we have based ourselves on the B&K solution — the successful application of their basic idea to Urdu provides a confirmation of the basic principles of finite-state based non-concatenative morphology formulated by B&K.

4 Issues in Potential Ambiguity

In this final section of the paper, we address some issues that arise with respect to the morphology-syntax interface. Recall from the discussion of the Urdu/Hindi future in section 2.2 that the future is formed in combination with subjunctive forms. Our present analysis of future forms is thus as in (12).

(12) mArUNgI ⇔
mAr+Verb+Subjunct+1P+Sg+Fut+Fem

From the perspective of the syntax (and semantics), marking these forms as subjunctive as well as future is unnecessary as every future form also carries some subjunctive meaning with it (this has been dubbed *contingent future* in the literature). Experience gathered with respect to the German ParGram grammar [1] has shown that it is ultimately better to eliminate tags of this kind from the

[8] 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.

morphology, since dealing with them complicates the morphology-syntax interface. Given that there are simple subjunctive uses as in (13), the interpretation of the +Subjunct tag within the morphological component will need to differ depending on whether it is found in conjunction with future morphology or not.

(13) mArUN ⇔
mAr+Verb+Subjunct+1P+Sg

We have therefore decided to eliminate the +Subjunct tag from the morphological analysis of future forms altogether even though the morphology involved is in actual fact the subjunctive morphology.

A somewhat different version of this same problem is found with respect to Urdu/Hindi infinitives as in *dEkHnA* 'to look/looking', which can also be used as verbal nouns. To date, the morphology provides analyses as in (14).

(14) dEkHnA ⇔
dEkH+Verb+Inf+Masc+Sg

It will be imperative to know that infinitives can also function as nouns in the grammar. It might therefore be necessary to anticipate this in the morphology and provide both the analyses in (15) for the syntax.

(15) dEkHnA ⇔
dEkH+Verb+Inf+Masc+Sg
dEkH+Noun+Deverb+Masc+Sg

However, this would result in quite a bit of ambiguity within the morphological analyzer. Our current solution, shown in terms of LFG functional annotations in (16) is therefore to add the information that this form could optionally (denoted by the round brackets) be used as a noun whose type is deverbal as part of the definition of the morphology-syntax interface.

(16) +Inf ((↑NTYPE) = deverbal).

The abstract morphological tag +Inf is thus annotated with the functional information that it could also be used as a noun, in which case it is deverbal. This solution pushes the ambiguity from the morphology into the syntax, but since the syntax can eliminate the ambiguity by means of unifying in other information, it may be better to deal with the ambiguity in the syntax, rather than in the morphology, where no contextual information is available. We are currently experimenting with both possible solutions to determine the better one.

5 Conclusion

In this paper, we have introduced and addressed a number of issues that arise in the process of building a finite-state morphological analyzer for Urdu. Our

approach allows for an underlyingly similar treatment of both Urdu and Hindi via a cascade of finite-state transducers that transliterates the very different scripts into a common ASCII transcription system. As this transliteration system is based on the XFST tools that the Urdu/Hindi common morphological analyzer is also implemented in, no compatibility problems arise.

We further explored reduplication in Urdu, again basing ourselves on solutions proposed with respect to XFST and show how differing reduplication patterns in Urdu/Hindi can be dealt with elegantly with the finite-state methods proposed by B&K.

Finally, we addressed some potential ambiguity problems and discussed different ways of solving them. The discussion here mainly revolves around where and how information should be encoded with respect to the morphology-syntax interface that has been defined between finite-state morphological analyzers and LFG grammars as part of the ParGram project.

References

1. Butt, M., King, T.H., Niño, M.E., Segond, F.: A Grammar Writer's Cookbook. CSLI Publications (1999)
2. Butt, M., Dyvik, H., King, T.H., Masuichi, H., Rohrer, C.: The Parallel Grammar project. In: Proceedings of COLING, Workshop on Grammar Engineering and Evaluation, Taipei (2002) 1–7
3. Kaplan, R.M., Maxwell III, J.T., King, T.H., Crouch, R.: Integrating finite-state technology with deep LFG grammars. In: Proceedings ESSLLI, Workshop on Combining Shallow and Deep Processing for NLP. (2004)
4. Dalrymple, M.: Lexical Functional Grammar. Academic Press (2001)
5. Butt, M., King, T.H.: Urdu in a parallel grammar development environment. Language Resources and Evaluation (2007) Special Issue on Asian Language Processing: State of the Art Resources and Processing. To Appear.
6. Beesley, K., Karttunen, L.: Finite State Morphology. CSLI Publications, Stanford, CA (2003)
7. Rahman, S., Hussain, S.: Development of character based Urdu Nastaleeq font. Asian Media and Communication Bulletin **33**(2) (2003)
8. Glassman, E.H.: Spoken Urdu. Nirali Kitaben, Lahore (1977)
9. Abbas Malik, M.: Hindi Urdu machine transliteration system. MSc Thesis, Paris 7 (2006)
10. Wells, J.: SAMPA computer readable phonetic alphabet. In Gibbon, D., Moore, R., Winski, R., eds.: Handbook of Standards and Resources for Spoken Language Systems. Mouton de Gruyter, Berlin and New York (1997)
11. Humayoun, M., Hammarström, H., Ranta, A.: Urdu morphology, orthography and lexicon extraction. In Farghaly, A., Megerdoomian, K., eds.: Proceedings of the 2nd Workshop on Computational Approaches to Arabic Script-based Languages. (2007) 59–66 Held at the Stanford LSA 2007 Institute.
12. Forsberg, M., Ranta, A.: Functional morphology. In: Proceedings of Ninth ACM SIGPLAN International Conference of Functional Programming. (2004) 213–223
13. Kellogg, S.H.: Grammar of the Hindi Language. Munshiram Manoharlal Publishers Pvt. Ltd., Delhi (1893) Second Edition, reprinted 1990.

14. McGregor, R.: The Language of Indrajit of Orchā. Cambridge University Press, Cambridge (1968)
15. Harris, A.C., Campbell, L.: Historical Syntax in Cross-Linguistic Perspective. Cambridge University Press, Cambridge (1995)
16. Hopper, P.J., Traugott, E.C.: Grammaticalization. Cambridge University Press, Cambridge (1993)
17. Abbi, A.: Reduplication in South Asian Languages. An Areal, Topological and Historical Study. Allied, New Delhi (1991)