Urdu – Roman Transliteration via Finite State Transducers

Tina Bögel
University of Konstanz
Konstanz, Germany
Tina.Boegel@uni-konstanz.de

Abstract

This paper introduces a two-way Urdu–Roman transliterator based solely on a non-probabilistic finite state transducer that solves the encountered scriptural issues via a particular architectural design in combination with a set of restrictions. In order to deal with the enormous amount of overgenerations caused by inherent properties of the Urdu script, the transliterator depends on a set of phonological and orthographic restrictions and a word list; additionally, a default component is implemented to allow for unknown entities to be transliterated, thus ensuring a large degree of flexibility in addition to robustness.

1 Introduction

This paper introduces a way of **t**ransliterating **U**rdu and **R**oman via a non-probabilistic **f**inite state transducer (**TURF**), thus allowing for easier machine processing.[1] The TURF transliterator was originally designed for a grammar of Hindi/Urdu (Bögel et al., 2009), based on the grammar development platform XLE (Crouch et al., 2011). This grammar is written in Roman script to serve as a bridge/pivot language between the different scripts used by Urdu and Hindi. It is in principle able to parse input from both Hindi and Urdu and can generate output for both of these language varieties. In order to achieve this goal, transliterators converting the scripts of Urdu and Hindi, respectively, into the common Roman representation are of great importance.

[1]I would like to thank Tafseer Ahmed and Miriam Butt for their help with the content of this paper. This research was part of the Urdu ParGram project funded by the Deutsche Forschungsgemeinschaft.

The TURF system presented in this paper is concerned with the Urdu–Roman transliteration. It deals with the Urdu-specific orthographic issues by integrating certain restrictional components into the finite state transducer to cut down on overgeneration, while at the same time employing an architectural design that allows for a large degree of flexibility. The transliterator is based solely on a non-probabilistic finite state transducer implemented with the Xerox finite state technology (XFST) (Beesley and Karttunen, 2003), a robust and easy-to-use finite state tool.

This paper is organized as follows: In section 2, one of the (many) orthographic issues of Urdu is introduced. Section 3 contains a short review of earlier approaches. Section 4 gives a brief introduction into the transducer and the set of restrictions used to cut down on overgeneration. Following this is an account of the architectural design of the transliteration process (section 5). The last two sections provide a first evaluation of the TURF system and a final conclusion.

2 Urdu script issues

Urdu is an Indo-Aryan language spoken mainly in Pakistan and India. It is written in a version of the Persian alphabet and includes a substantial amount of Persian and Arabic vocabulary. The direction of the script is from right to left and the shapes of most characters are context sensitive; i.e., depending on the position within the word, a character assumes a certain form.

Urdu has a set of diacritical marks which appear above or below a character defining a particular vowel, its absence or compound forms. In total, there are 15 of these diacritics (Malik, 2006, 13);

Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing, pages 25–29,
Donostia–San Sebastián, July 23–25, 2012.

the four most frequent ones are shown in Table 1 in combination with the letter ب 'b'.

ب + diacritic	Name	Roman transliteration
بَ	Zabar	ba
بِ	Zer	bi
بُ	Pesh	bu
بّ	Tashdid	bb

Table 1: The four most frequently used diacritics

When transliterating from the Urdu script to another script, these diacritics present a huge problem because in standard Urdu texts, the diacritics are rarely used. Thus, for example, we generally are only confronted with the letter ب 'b' and have to guess at the pronunciation that was intended. Take, e.g., the following example, where the word کتا kuttA 'dog' is to be transliterated. Without diacritics, the word consists of three letters: *k*, *t* and *A*. If in the case of transliteration, the system takes a guess at possible short vowels and geminated consonants, the output contains multiple possibilities ((1)).

(1)
```
fst[1]: up کتا
kuttA
kutA
kittA
kitA
kattA
katA
```

In addition to the correct transliteration *kuttA*, the transliterator proposes five other possibilities for the missing diacritics. These examples show that this property of the Urdu script makes it extremely difficult for any transliterator to correctly transliterate undiacriticized input without the help of word lists.

3 Earlier approaches

Earlier approaches to Urdu transliteration almost always have been concerned with the process of transliterating Urdu to Hindi *or* Hindi to Urdu (see, e.g., Lehal and Saini (2010) (Hindi → Urdu), Malik et al. (2009) (Urdu → Hindi), Malik et al. (2010) (Urdu → Roman) or Ahmed (2009) (Roman → Urdu). An exception is Malik (2006), who explored the general idea of using finite state transducers and an intermediate/pivot language to deal with the issues of the scripts of Urdu and Hindi.

All of these approaches are highly dependent on word lists due to the properties of the Urdu script and the problems arising with the use of diacritics. Most systems dealing with undiacriticized input are faced with low accuracy rates: The original system of Malik (2006), e.g., drops from approximately 80% to 50% accuracy (cf. Malik et al. (2009, 178)) – others have higher accuracy rates at the cost of being unidirectional.

While Malik et al. (2009) have claimed that the non-probabilistic finite state model is not able to handle the orthographic issues of Urdu in a satisfying way, this paper shows that there are possibilities for allowing a high accuracy of interpretation, even if the input text does not include diacritics.

4 The TURF Transliterator

The TURF transliterator has been implemented as a non-probabilistic finite state transducer compiled with the **lexc** language (Lexicon Compiler), which is explicitly designed to build finite state networks and analyzers (Beesley and Karttunen, 2003, 203). The resulting network is completely compatible with one that was written with, e.g., regular expressions, but has the advantage in that it is easily readable. The transliteration scheme used here was developed by Malik et al. (2010), following Glassman (1986).

As has been shown in section 1, Urdu transliteration with simple character-to-character mapping is not sufficient. A default integration of short vowels and geminated consonants will, on the other hand, cause significant overgeneration. In order to reduce this overgeneration and to keep the transliterator as efficient as possible, the current approach integrates several layers of restrictions.

4.1 The word list

When dealing with Urdu transliteration it is not possible to *not* work with a word list in order to exclude a large proportion of the overgenerated output. In contrast to other approaches, which depend on Hindi or Urdu wordlists, TURF works with a Roman wordlist. This wordlist is derived from an XFST finite state morphology (Bögel et al., 2007) independently created as part of the Urdu ParGram development effort for the Roman intermediate language (Bögel et al., 2009).

4.2 Regular expression filters

The regular expression filters are based on knowledge about the phonotactics of the language and are a powerful tool for reducing the number of possibilities proposed by the transliterator. As a concrete example, consider the filter in (2).

(2) [∼[y A [a |i |u]]]

In Urdu a combination of *[y A short vowel]* is not allowed (∼). A filter like in (2) can thus be used to disallow any generations that match this sequence.

4.3 Flag diacritics

The XFST software also provides the user with a method to store 'memory' within a finite state network (cf. Beesley and Karttunen (2003, 339)). These so-called *flag diacritics* enable the user to enforce desired constraints within a network, keeping the transducers relatively small and simple by removing illegal paths and thus reducing the number of possible analyses.

5 The overall TURF architecture

However, the finite state transducer should also be able to deal with unknown items; thus, the constraints on transliteration should not be too restrictive, but should allow for a default transliteration as well. Word lists in general have the drawback that a matching of a finite state transducer output against a word list will delete any entities not on the word list. This means that a methodology needs to be found to deal with unknown but legitimate words without involving any further (non-finite state) software. Figure 1 shows the general architecture to achieve this goal. For illustrative purposes two words are transliterated: كتاب kitAb 'book' and كت, which transliterates to an unknown word *kt*, potentially having the surface forms *kut, kat* or *kit*.

5.1 Step 1: Transliteration Part 1

The finite state transducer itself consists of a network containing the Roman–Urdu character mapping with the possible paths regulated via flag diacritics. Apart from these regular mappings, the network also contains a default Urdu and a default Roman component where the respective characters are simply matched against themselves (e.g. k:k, r:r). On top of this network, the regular expression filters provide further restrictions for the output form.

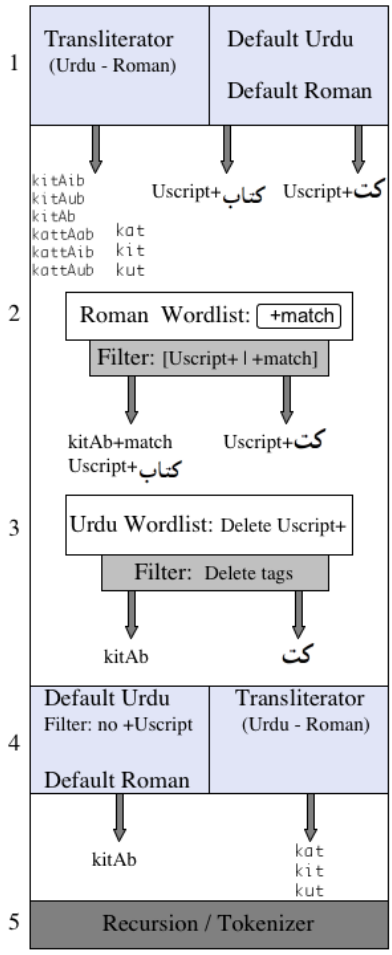

Figure 1: Transliteration of كت and كتاب

The Urdu script default 1-1 mappings are marked with a special identification tag (`[+Uscript]`) for later processing purposes. Thus, an Urdu script word will not only be transliterated into the corresponding Roman script, but will also be 'transliterated' into itself plus an identificational tag.

The output of the basic transliterator shows part of the vast overgeneration caused by the underspecified nature of the script, even though the restricting filters and flags are compiled into this component.

5.2 Step 2: Word list matching and tag deletion

In step 2, the output is matched against a Roman word list. In case there is a match, the respective word is tagged `[+match]`. After this process, a

filter is applied, erasing all output forms that contain neither a `[+match]` nor a `[Uscript+]` tag. This way we are left with two choices for the word کتاب – one transliterated 'matched' form and one default Urdu form – while the word کت is left with only the default Urdu form.

5.3 Step 3: Distinguishing unknown and overgenerated entities

The Urdu word list applied in step 3 is a transliteration of the original Roman word list (4.1), which was transliterated via the TURF system. Thus, the Urdu word list is a mirror image of the Roman word list. During this step, the Urdu script words are matched against the Urdu word list, this time *deleting* all the words that *find* a match. As was to be expected from matching against a mirror word list of the original Roman word list, all of the words that found a match in the Roman word list will also find a match in the Urdu word list, while all unknown entities fail to match. As a result, any Urdu script version of an already correctly transliterated word is deleted, while the Urdu script unknown entity is kept for further processing – the system has now effectively separated known from unknown entities.

In a further step, the tags of the remaining entities are deleted, which leaves us with the correct transliteration of the known word *kitAb* and the unknown Urdu script word کت.

5.4 Step 4: Transliteration Part 2

The remaining words are once again sent into the finite state transducer of step 1. The Roman transliteration *kitAb* passes unhindered through the Default Roman part. The Urdu word on the other hand is transliterated to all possible forms (in this case three) within the range of the restrictions applied by flags and filters.

5.5 Step 5: Final adjustments

Up to now, the transliterator is only applicable to single words. With a simple (recursive) regular expression it can be designed to apply to larger strings containing more than one word.

The ouput can then be easily composed with a standard tokenizer (e.g. Kaplan (2005)) to enable smooth machine processing.

6 Evaluation

A first evaluation of the TURF transliterator with unseen texts resulted in an accuracy of 86%, if the input was *not* diacriticized. The accuracy rate for undiacriticized text always depends on the size of the word list. The word list used in this application is currently being extended from formerly 20.000 to 40.000 words; thus, a significant improvement of the accuracy rate can be expected within the next few months.

If the optional inclusion of short vowels is removed from the network, the accuracy rate for diacriticized input is close to 97%.

When transliterating from Roman to Urdu, the accuracy rate is close to a 100%, iff the Roman script is written according to the transliteration scheme proposed by Malik et al. (2010).

Transliteration	U → R	U → R	R → U
Input	diacritics	no diacritics	
Diacritics	opt. / compuls.	optional	
Accuracy	86% / 97%	86%	∼ 100%

Table 2: Accuracy rates of the TURF transliterator

7 Conclusion

This paper has introduced a finite state transducer for Urdu ↔ Roman transliteration. Furthermore, this paper has shown that it is possible for applications based *only* on non-probabilistic finite state technology to return output with a high state-of-the-art accuracy rate; as a consequence, the application profits from the inherently fast and small nature of finite state transducers.

While the transliteration from Roman to Urdu is basically a simple character to character mapping, the transliteration from Urdu to Roman causes a substantial amount of overgeneration due to the underspecified nature of the Urdu script. This was solved by applying different layers of restrictions.

The specific architectural design enables TURF to distinguish between unknown-to-the-word-list and overgenerated items; thus, when matched against a word list, unknown items are not deleted along with the overgenerated items, but are transliterated along with the known items. As a consequence, a transliteration is always given, resulting in an efficient, highly accurate and robust system.

References

Tafseer Ahmed. 2009. Roman to Urdu transliteration using wordlist. In *Proceedings of the Conference on Language and Technology 2009 (CLT09)*, CRULP, Lahore.

Kenneth R. 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 T. Hanneforth und K. M. Würzner, editor, *Proceedings of the Sixth International Workshop on Finite-State Methods and Natural Language Processing*, pages 86–96, Potsdam. Potsdam University Press.

Tina Bögel, Miriam Butt, Annette Hautli, and Sebastian Sulger. 2009. Urdu and the modular architecture of ParGram. In *Proceedings of the Conference on Language and Technology 2009 (CLT09)*, CRULP, Lahore.

Dick Crouch, Mary Dalrymple, Ron Kaplan, Tracy King, John Maxwell, and Paula Newman. 2011. *XLE Documentation*. Palo Alto Research Center, Palo Alto, CA. URL: http://www2.parc.com/isl/groups/nltt/xle/doc/xle_toc.html.

Eugene H. Glassman. 1986. *Spoken Urdu*. Nirali Kitaben Publishing House, Lahore, 6 edition.

Ronald M. Kaplan. 2005. A method for tokenizing text. In *Festschrift in Honor of Kimmo Koskenniemi's 60th anniversary*. CSLI Publications, Stanford, CA.

Gurpreet S. Lehal and Tejinder S. Saini. 2010. A Hindi to Urdu transliteration system. In *Proceedings of ICON-2010: 8th International Conference on Natural Language Processing*, Kharagpur.

Abbas Malik, Laurent Besacier, Christian Boitet, and Pushpak Bhattacharyya. 2009. A hybrid model for Urdu Hindi transliteration. In *Proceedings of the 2009 Named Entities Workshop, ACL-IJCNLP*, pages 177–185, Suntec, Singapore.

Muhammad Kamran Malik, Tafseer Ahmed, Sebastian Sulger, Tina Bögel, Atif Gulzar, Ghulam Raza, Sarmad Hussain, and Miriam Butt. 2010. Transliterating Urdu for a Broad-Coverage Urdu/Hindi LFG Grammar. In *Proceedings of the Seventh Conference on International Language Resources and Evaluation (LREC 2010)*. European Language Resources Association (ELRA).

Abbas Malik. 2006. Hindi Urdu machine transliteration system. Master's thesis, University of Paris.