Handling Unknown Words in Arabic FST Morphology

Khaled Shaalan and Mohammed Attia
Faculty of Engineering & IT,
The British University in Dubai
khaled.shaalan@buid.ac.ae
mohammed.attia@buid.ac.ae

Abstract

A morphological analyser only recognizes words that it already knows in the lexical database. It needs, however, a way of sensing significant changes in the language in the form of newly borrowed or coined words with high frequency. We develop a finite-state morphological guesser in a pipelined methodology for extracting unknown words, lemmatizing them, and giving them a priority weight for inclusion in a lexicon. The processing is performed on a large contemporary corpus of 1,089,111,204 words and passed through a machine-learning-based annotation tool. Our method is tested on a manually-annotated gold standard of 1,310 forms and yields good results despite the complexity of the task. Our work shows the usability of a highly non-deterministic finite state guesser in a practical and complex application.

1 Introduction

Due to the complex and semi-algorithmic nature of the Arabic morphology, it has always been a challenge for computational processing and analysis (Kiraz, 2001; Beesley 2003; Shaalan et al., 2012). A lexicon is an indispensable part of a morphological analyser (Dichy and Farghaly, 2003; Attia, 2006; Buckwalter, 2004; Beesley, 2001), and the coverage of the lexical database is a key factor in the coverage of the morphological analyser. This is why an automatic method for updating a lexical database is crucially important.

We present the first attempt, to the best of our knowledge, to address lemmatization of Arabic unknown words. The specific problem with lemmatizing unknown words is that they cannot be matched against a morphological lexicon. We develop a rule-based finite-state morphological guesser and use a machine learning disambiguator, MADA (Roth et al., 2008), in a pipelined approach to lemmatization.

This paper is structured as follows. The remainder of the introduction reviews previous work on Arabic unknown word extraction and lemmatization, and explains the data used in our experiments. Section 2 presents the methodology followed in extracting and analysing unknown words. Section 3 provides details on the morphological guesser we have developed to help deal with the problem. Section 4 shows and discusses the testing and evaluation results, and finally Section 5 gives the conclusion.

1.1 Previous Work

Lemmatization of Arabic words has been addressed in (Roth et al., 2008; Dichy, 2001). Lemmatization of unknown words has been addressed for Slovene in (Erjavec and Džerosk, 2004), for Hebrew in (Adler at al., 2008) and for English, Finnish, Swedish and Swahili in (Lindén, 2008). Lemmatization means the normalization of text data by reducing surface forms to their canonical underlying representations, which, in Arabic, means verbs in their perfective, indicative, 3rd person, masculine, singular forms, such as شَكَرَ

$akara “to thank”; and nominals in their nominative, singular, masculine forms, such as طالِب TAlib “student”; and nominative plural for *pluralia tantum* nouns (or nouns that appear only in the plural form and are not derived from a singular word), such as ناس nAs “people”. To the best of our knowledge, the study presented here is the first to address lemmatization of Arabic unknown words. The specific problem with lemmatizing unknown words is that they cannot be matched against a lexicon. In our method, we use a machine learning disambiguator, develop a rule-based finite-state morphological guesser, and combine them in a pipelined process of lemmatization. We test our method against a manually created gold standard of 1,310 types (unique forms) and show a significant improvement over the baseline. Furthermore, we develop an algorithm for weighting and prioritizing new words for inclusion in a lexicon depending on three factors: number of form variations of the lemmas, cumulative frequency of the forms, and POS tags.

1.2 Data Used

In our work we rely on a large-scale corpus of 1,089,111,204 words, consisting of 925,461,707 words from the Arabic Gigaword Fourth Edition (Parker et al., 2009), and 163,649,497 words from news articles collected from the Al-Jazeera web site.[1] In this corpus, unknown words appear at a rate of between 2% of word tokens (when we ignore possible spelling variants) and 9% of word tokens (when possible spelling variants are included).

2 Methodology

To deal with unknown words, or out-of-vocabulary words (OOVs), we use a pipelined approach, which predicts part-of-speech tags and morpho-syntactic features before lemmatization. First, a machine learning, context-sensitive tool is used. This tool, MADA (Roth et al., 2008), performs POS tagging and morpho-syntactic analysis and disambiguation of words in context. MADA internally uses the Standard Arabic Morphological Analyser (SAMA) (Maamouri et al., 2010), an updated version of Buckalter Arabic Morphological Analyser (BAMA) (Buckwalter, 2004). Second, we develop a finite-state morphological guesser that gives all possible interpretations of a given word. The morphological guesser first takes an Arabic form as a whole and then strips off all possible affixes and clitics one by one until all potential analyses are exhausted. As the morphological guesser is highly non-deterministic, all the interpretations are matched against the morphological analysis of MADA that receives the highest probabilistic scores. The guesser’s analysis that bears the closest resemblance (in terms of morphological features) with the MADA analysis is selected.

These are the steps followed in extracting and lemmatizing Arabic unknown words:

- A corpus of 1,089,111,204 is analysed with MADA. The number of types for which MADA could not find an analysis in SAMA is 2,116,180.
- These unknown types are spell checked by the Microsoft Arabic spell checker using MS Office 2010. Among the unknown types, the number of types accepted as correctly spelt is 208,188.
- We then select types with frequency of 10 or more. This leave us with 40,277 types.
- We randomly select 1,310 types and manually annotate them with the gold lemma, the gold POS and lexicographic preference for inclusion in a dictionary.
- We use the full POS tags and morpho-syntactic features produced by MADA.
- We use the finite-state morphological guesser to produce all possible morphological interpretations and corresponding lemmatizations.
- We compare the POS tags and morpho-syntactic features in MADA output with the output of the morphological guesser and choose the one with the highest matching score.

3 Morphological Guesser

We develop a morphological guesser for Arabic that analyses unknown words with all possible clitics, morpho-syntactic affixes and all relevant alteration operations that include insertion, assimilation, and deletion. Beesley and Karttunen

[1] http://aljazeera.net/portal. Collected in January 2010.

(2003) show how to create a basic guesser. The core idea of a guesser is to assume that a stem is composed of any arbitrary sequence of Arabic non-numeric characters, and this stem can be prefixed and/or suffixed with a predefined set of prefixes, suffixes or clitics. The guesser marks clitic boundaries and tries to return the stem to its underlying representation, the lemma. Due to the nondeterministic nature of the guesser, there will be a large number of possible lemmas for each form.

The XFST finite-state compiler (Beesley and Karttunen, 2003) uses the "substitute defined" command for creating the guesser. The XFST commands in our guesser are stated as follows.

```
define PossNounStem
[[Alphabet]^{2,24}] "+Guess":0;
define PossVerbStem
[[Alphabet]^{2,6}] "+Guess":0;
```

This rule states that a possible noun stem is defined as any sequence of Arabic non-numeric characters of length between 2 and 24 characters. A possible verb stem is between 2 and 6 characters. The length is the only constraint applied to an Arabic word stem. This word stem is surrounded by prefixes, suffixes, proclitics and enclitics. Clitics are considered as independent tokens and are separated by the '@' sign, while prefixes and suffixes are considered as morpho-syntactic features and are interpreted with tags preceded by the '+' sign. Below we present the analysis of the unknown noun والمُسَوِّقونَ wa-Al-musaw~iquwna "and-the-marketers".

MADA output:
```
form:wAlmswqwn      num:p   gen:m   per:na
       case:n   asp:na   mod:na vox:na  pos:noun
       prc0:Al_det      prc1:0  prc2:wa_conj
       prc3:0  enc0:0  stt:d
```

Finite-state guesser output:
```
والمسوقون    +adjوالمسوق+Guess+masc+pl+nom@
والمسوقون    +adjوالمسوقون+Guess+sg@
والمسوقون    +nounوالمسوق+Guess+masc+pl+nom@
والمسوقون    +nounوالمسوقون+Guess+sg@
والمسوقون    و+conj@ال+defArt@+adjمسوق
             +Guess+masc+pl+nom@
والمسوقون    و+conj@ال+defArt@+adjمسوقون
             +Guess+sg@
والمسوقون    و+conj@ال+defArt@+nounمسوق
             +Guess+masc+pl+nom@
والمسوقون    و+conj@ال+defArt@+nounمسوقون
             +Guess+sg@
والمسوقون    و+conj@+adjالمسوق+Guess+masc
             +pl+nom@
والمسوقون    و+conj@+adjالمسوقون+Guess+sg@
والمسوقون    و+conj@+nounالمسوق+Guess+masc
             +pl+nom@
والمسوقون    و+conj@+nounالمسوقون+Guess+sg@
```

For a list of 40,277 word types, the morphological guesser gives an average of 12.6 possible interpretations per word. This is highly non-deterministic when compared to AraComLex morphological analyser (Attia et al. 2011) which has an average of 2.1 solutions per word. We also note that 97% of the gold lemmas are found among the finite-state guesser's choices.

4 Testing and Evaluation

To evaluate our methodology we create a manually annotated gold standard test suite of randomly selected surface form types. For these surface forms, the gold lemma and part of speech are manually given. Besides, the human annotator gives a preference on whether or not to include the entry in a dictionary. This feature helps to evaluate our lemma weighting equation. The annotator tends to include nouns, verbs and adjectives, and only proper nouns that have a high frequency. The size of the test suite is 1,310.

4.1 Evaluating Lemmatization

In the evaluation experiment we measure accuracy calculated as the number of correct tags divided by the count of all tags. The baseline is given by the assumption that new words appear in their base form, i.e., we do not need to lemmatize them. The baseline accuracy is 45% as shown in Table 1. The POS tagging baseline proposes the most frequent tag (proper name) for all unknown words. In our test data this stands at 45%. We notice that MADA POS tagging accuracy is unexpectedly low (60%). We use Voted POS Tagging, that is when a lemma gets a different POS tag with a higher frequency, the new tag replaces the old low frequency tag.

This method has improved the tagging results significantly (69%).

		Accuracy
	POS tagging	
1	POS Tagging baseline	45%
2	MADA POS tagging	60%
3	Voted POS Tagging	69%

Table 1. Evaluation of POS tagging

As for the lemmatization process itself, we notice that our experiment in the pipelined lemmatization approach gains a higher (54%) score than the baseline (45%) as shown in Table 2. This score significantly rises to 63% when the difference in the definite article 'Al' is ignored. The testing results indicate significant improvements over the baseline.

	Lemmatization	
1	Lemmas found among corpus forms	64%
2	Lemmas found among FST guesser forms	97%
3	Lemma first-order baseline	45%
4	Pipelined lemmatization (first-order decision) with strict definite article matching	54%
5	Pipelined lemmatization (first-order decision) ignoring definite article matching	63%

Table 2. Evaluation of lemmatization

4.2 Evaluating Lemma Weighting

In our data we have 40,277 unknown token types. After lemmatization they are reduced to 18,399 types (that is 54% reduction of the surface forms) which are presumably ready for manual validation before being included in a lexicon. This number is still too big for manual inspection. In order to facilitate human revision, we devise a weighting algorithm for ranking so that the top *n* number of words will include the most lexicographically relevant words. We call surface forms that share the same lemma 'sister forms', and we call the lemma that they share the 'mother lemma'. This weighting algorithm is based on three criteria: frequency of the sister forms, number of sister forms, and a POS factor which penalizes proper nouns (due to their disproportionate high frequency). The parameters of the weighting algorithm has been tuned through several rounds of experimentation.

```
Word Weight = ((number of sister
forms having the same mother
lemma * 800) + cumulative sum of
frequencies of sister forms
having the same mother lemma) /
2 + POS factor
```

Good words	In top 100	In bottom 100
relying on Frequency alone (baseline)	63	50
relying on number of sister forms * 800	87	28
relying on POS factor	58	30
using the combined criteria	78	15

Table 3. Evaluation of lemma weighting and ranking

Table 3 shows the evaluation of the weighting criteria. We notice that the combined criteria gives the best balance between increasing the number of good words in the top 100 words and reducing the number of good words in the bottom 100 words.

5 Conclusion

We develop a methodology for automatically extracting unknown words in Arabic and lemmatizing them in order to relate multiple surface forms to their base underlying representation using a finite-state guesser and a machine learning tool for disambiguation. We develop a weighting mechanism for simulating a human decision on whether or not to include the new words in a general-domain lexical database. We show the feasibility of a highly non-deterministic finite state guesser in an essential and practical application.

Out of a word list of 40,255 unknown words, we create a lexicon of 18,399 lemmatized, POS-tagged and weighted entries. We make our unknown word lexicon available as a free open-source resource[2].

Acknowledgments

This research is funded by the UAE National Research Foundation (NRF) (Grant No. 0514/2011).

[2] http://arabic-unknowns.sourceforge.net/

References

Adler, M., Goldberg, Y., Gabay, D. and Elhadad, M. 2008. Unsupervised Lexicon-Based Resolution of Unknown Words for Full Morphololological Analysis. In: Proceedings of Association for Computational Linguistics (ACL), Columbus, Ohio.

Attia, M. 2006. An Ambiguity-Controlled Morphological Analyzer for Modern Standard Arabic Modelling Finite State Networks. In: Challenges of Arabic for NLP/MT Conference, The British Computer Society, London, UK.

Attia, Mohammed, Pavel Pecina, Lamia Tounsi, Antonio Toral, Josef van Genabith. 2011. An Open-Source Finite State Morphological Transducer for Modern Standard Arabic. International Workshop on Finite State Methods and Natural Language Processing (FSMNLP). Blois, France.

Beesley, K. R. 2001. Finite-State Morphological Analysis and Generation of Arabic at Xerox Research: Status and Plans in 2001. In: The ACL 2001 Workshop on Arabic Language Processing: Status and Prospects, Toulouse, France.

Beesley, K. R., and Karttunen, L.. 2003. Finite State Morphology: CSLI studies in computational linguistics. Stanford, Calif.: Csli.

Buckwalter, T. 2004. Buckwalter Arabic Morphological Analyzer (BAMA) Version 2.0. Linguistic Data Consortium (LDC) catalogue number LDC2004L02, ISBN1-58563-324-0

Dichy, J. 2001. On lemmatization in Arabic, A formal definition of the Arabic entries of multilingual lexical databases. ACL/EACL 2001 Workshop on Arabic Language Processing: Status and Prospects. Toulouse, France.

Dichy, J., and Farghaly, A. 2003. Roots & Patterns vs. Stems plus Grammar-Lexis Specifications: on what basis should a multilingual lexical database centred on Arabic be built? In: The MT-Summit IX workshop on Machine Translation for Semitic Languages, New Orleans.

Erjavec, T., and Džerosk, S. 2004. Machine Learning of Morphosyntactic Structure: Lemmatizing Unknown Slovene Words. *Applied Artificial Intelligence*, 18:17–41.

Kiraz, G. A. 2001. *Computational Nonlinear Morphology: With Emphasis on Semitic Languages*. Cambridge University Press.

Lindén, K. 2008. A Probabilistic Model for Guessing Base Forms of New Words by Analogy. In CICling-2008, 9th International Conference on Intelligent Text Processing and Computational Linguistics, Haifa, Israel, pp. 106-116.

Maamouri, M., Graff, D., Bouziri, B., Krouna, S., and Kulick, S. 2010. LDC Standard Arabic Morphological Analyzer (SAMA) v. 3.1. LDC Catalog No. LDC2010L01. ISBN: 1-58563-555-3.

Parker, R., Graff, D., Chen, K., Kong, J., and Maeda, K. 2009. Arabic Gigaword Fourth Edition. LDC Catalog No. LDC2009T30. ISBN: 1-58563-532-4.

Roth, R., Rambow, O., Habash, N., Diab, M., and Rudin, C. 2008. Arabic Morphological Tagging, Diacritization, and Lemmatization Using Lexeme Models and Feature Ranking. In: Proceedings of Association for Computational Linguistics (ACL), Columbus, Ohio.

Shaalan, K., Magdy, M., Fahmy, A., Morphological Analysis of Il-formed Arabic Verbs for Second Language Learners, In Eds. McCarthy P., Boonthum, C., *Applied Natural Language Processing: Identification, Investigation and Resolution*, PP. 383-397, IGI Global, PA, USA, 2012.

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);

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: کتاب kitAb 'book' and کت, 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.

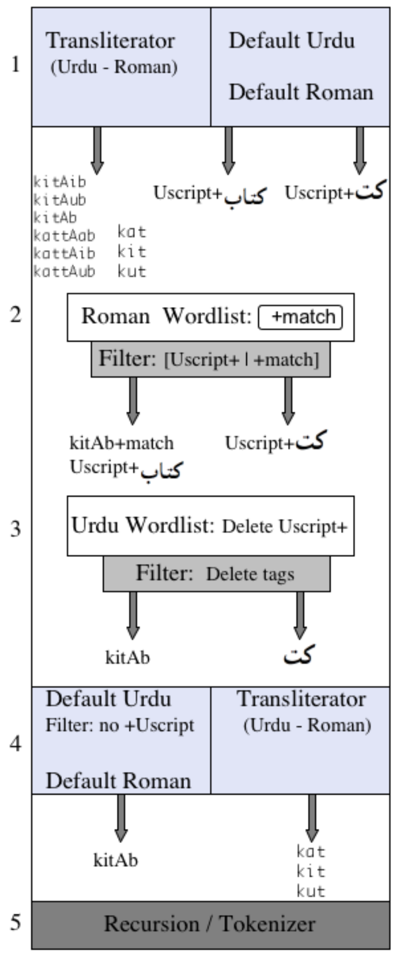

Figure 1: Transliteration of کت and کتاب

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	$U \rightarrow R$	$U \rightarrow R$	$R \rightarrow U$
Input	diacritics	no diacritics	
Diacritics	opt. / compuls.	optional	
Accuracy	86% / 97%	86%	$\sim$ 100%

Table 2: Accuracy rates of the TURF transliterator

7 Conclusion

This paper has introduced a finite state transducer for Urdu $\leftrightarrow$ 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.