JKimmo: A Multilingual Computational Morphology Framework for PC-KIMMO

Md. Zahurul Islam and Mumit Khan

Center for Research on Bangla Language Processing, Department of Computer Science and Engineering, BRAC University, Dhaka Bangladesh
zahurul@bracu.ac.bd, mumit@bracu.ac.bd

Abstract

Morphological analysis is of fundamental interest in computational linguistics and language processing. While there are established morphological analyzers for mostly Western and a few other languages using localized interfaces, the same cannot be said for Indic and other less-studied languages for which language processing is just beginning. There are three primary obstacles to computational morphological analysis of these less-studied languages: the generative rules that define the language morphology, the morphological processor, and the computational interface that a linguist can use to experiment with the generative rules. In this paper, we present JKimmo, a multilingual morphological open-source framework that uses the PC-KIMMO two-level morphological processor and provides a localized interface for Bangla morphological analysis. We then apply Jkimmo to Bangla computational morphology, demonstrating both its recognition and generation capabilities. Jkimmo's internationalization (i18n) frame-work allows easy localization in other languages as well, using a property file for the interface definitions and a transliteration scheme for the analysis.

1. Introduction

Morphological analysis is a key component of Natural Language Processing (NLP) and Computational Linguistics, and is a fundamental requirement of most advanced language processing applications from grammar checkers to automatic machine translators. With the current wave of work in Bangla Computational Linguistics, the need for a robust morphological analyzer has become critical. Our goal is to create a robust and reusable framework for doing morphological analysis of Bangla. There are three primary components in such a robust morphological analyzer for a language: the generative morphological rules, the underlying morphological processor, and the computational interface through which the user experiments with the language morphology. There is ongoing work in developing the computational morphology for Bangla, using both simple rewriting rules and feature unification grammars [1-4]. There are also well-established implementations for two-level morphological analyzers,

with PC-KIMMO being one of the more widely available ones that implements Kimmo Koskenniemi's two-level morphology [5-8]. What is missing however is the framework in which Bangla morphology can be implemented using Bangla language interface. The available processors were created before the widespread use of Unicode [9], predominantly using the Latin script. This creates an obstacle in creating us-able local language interfaces, making it difficult to experiment with the morphology of languages that use complex scripts, such as the Indic scripts including Bangla. Instead of creating yet another two-level morphological processor, we chose instead to Jkimmo by harnessing the existing PC-KIMMO implementation [8], using the generative rules defined by existing efforts, and created a software interface that allows Bangla language interface to PC-KIMMO. Our implementation uses Java Native Interface [10] as the bridge between PC-KIMMO and the Unicode-enabled user interface, allowing the user to experiment in any script supported by the Unicode standard. Since the analysis framework uses standard internationalization (i18n) schemes, it is trivially localized to any language by using property files for interface definitions, and transliteration schemes for the Latin-Unicode-Latin conversion needed to inter-face to PC-KIMMO backend.

In section 2, we review some related work including work on Bangla morphological analyzers, followed by our methodology and implementation details in sections 3 and 4, and then conclude with some discussion of Jkimmo.

2. Related Work

Pykimmo [11] is a python implementation of PC-KIMMO developed by Carl de Marcken, Beracah Yankama, and Rob Speer at Massachusetts Institute of Technology. It was designed for "laboratory" experimentation with two-level morphological rules. However, since Pykimmo uses Latin scripts for both input and output, it requires the use of transliteration and English language user interface to interact with the system, thereby limiting its use. Another limitation of Pykimmo is that it's based PC-KIMMO version 1, which implements the two-level rules and the lexicon, but does not implement the grammar needed to describe non-concatenative and otherwise complex morphology. An effort for creating an interface for Bangla

morphological analysis has been developed at the Indian Institute of Technology - Kharagpur [12], which provides a web interface to the underlying morphological engine using the iTRANS transliteration scheme. Another such effort is the Xerox Arabic Morphological Analyzer and Generator [13], created with the Xerox Finite-State Technology. It has a Java Applet interface and uses ISO-8859-6 and Unicode character encodings. It is notable that none of these systems, unlike Jkimmo, is easily extendible to other languages using Unicode-encoded input and output.

3. Methodology

3.1. PC-KIMMO overview

PC-KIMMO is a morphological analyzer based on Kimmo Koskenniemi's Two-Level Morphology model [5]. The first implementation of the two-level model was PC-KIMMO version 1, developed by the Summer Institute of Linguistics in 1990. PC-KIMMO implemented the rules and lexicon components of the two-level model using two files: (i) the rules file (.RUL) which specifies all the orthographic rules, and (ii) the lexical file (.LEX) which specifies all the lexicons, classification of lexicons and morphotactic constraints of these classes. [5] The structure of PC-KIMMO is shown in Figure 1.
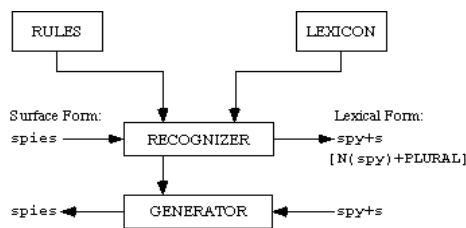

Figure 1: Structure of PC-KIMMO parser version 1

One limitation of the version 1 was its inability to perform syntactic parsing using a word grammar. This limitation was removed in PC-KIMMO version 2 with the introduction of a word grammar, adding it as the third component. [7] The version 2 included a feature-structure unification based chart parser capable of producing parse trees based on Shieber's PATR-II formalist. [14] The word grammar is specified using a grammar (.GRM) file, so this version requires a total of 3 files for morphological processing. PC-KIMMO has two functional components: *generation* and *recognition*.

3.2. Generation and Recognition

The generator uses the two-level rules to recursively compute the surface form from the lexical form. The rules are encoded using a finite automata that can be automatically generated from the rules.
The recognizer performs the inverse task to compute the lexical form from the surface form. Unlike the generator, the recognizer needs a lexicon in addition to the two-level rules.

4. Implementation

JKimmo is a graphical user interface (GUI) implemented in the JAVA programming language, using PC-KIMMO version 2 as the back end. PC-KIMMO has tree main component: two level orthographic rule, lexicon and grammar. These are also the main components of Jkimmo; in addition, JKimmo has another component – the transliteration scheme. The rule file must be loaded for morphological generation and both the rule and lexicon files must be loaded for morphological recognition. For generation, JKimmo does not need the grammar file; for recognition, the grammar file is optional. Since it uses PC-KIMMO as the backend, JKimmo automatically uses feature unification grammar.

4.1. JKimmo components

4.1.1. Transliteration file. The original PC-KIMMO software is written in C programming language and uses only Latin alphanumeric characters for input and output purposes. For inputs using scripts other than Latin, the user has to come up with his/her own transliteration scheme that uses Latin characters corresponding to characters of the non-Latin script. Viewing and understanding the input and output strings in such a way can be cumbersome and non-intuitive for the user.

JKimmo solves this problem in a modular, abstract fashion. It requires that the whole transliteration scheme be written down in a separate file. The user can then load that transliteration file. Once the transliteration file is loaded, the user can input strings and view output strings in his preferred language in an intuitive way. Transliteration scheme for Bengali language is given in Table 1.

4.1.2. Rule file. Two level orthographic rules are required for JKimmo. The rule file is same as PC-KIMMO rule file, reproduced here from the reference manual: "the general structure of the rules file is a list of declarations composed of a keyword followed by data. The set of valid keywords in a rules file includes COMMENT, ALPHABET, NULL, ANY, BOUNDARY, SUBSET, RULE, and END. The COMMENT, SUBSET and RULE declarations are optional and also can be used more than once in a rules file. The END declaration is also optional, but can only be used once". [7] PC-KIMMO only recognizes Latin characters in rule file. To implement rule for language that uses other than Latin script we must follow the transliteration scheme. There is a free rule compiler for PC-KIMMO called kgen is available. It takes rule specification and it generate rule for PC-KIMMO. There are more free tools available that can be used for rule generation.
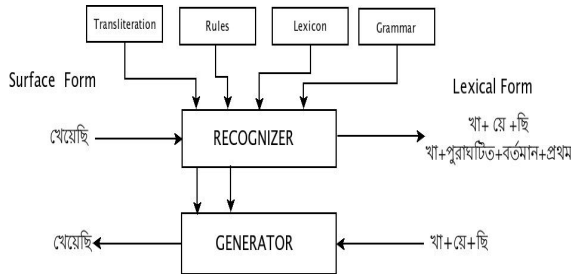
Figure 2: Communication protocol of JKimmo and PC-KIMMO

Figure 3: Main components of JKimmo

4.1.3. Lexicon file. The lexicon contains the indivisible words and morphemes in their lexical forms, i.e., the lexical items, as well as the morphotactic constraints. Its primary task is to decompose a word into its constituent morphemes using a simple positional analysis. The positional analysis need only go far enough to ensure that all correct parses are produced but not too many incorrect parses. Co-occurrence restrictions between morpheme positions are best handled in the word grammar, not the lexicon, because that will raise complexities of morphotactic analysis. The format for the lexicon is reproduced from the reference manual: "A lexicon consists of one main lexicon file plus one or more files of lexical entries. The general structure of the main lexicon file is a list of keyword declarations. The set of valid keywords is ALTERNATION, FEATURES, FIELDCODE, INCLUDE, and END." [7] To write lexicons that will be used in JKimmo for language that use other than Latin script then we have to follow the transliteration scheme.

4.1.4. Grammar file. The word grammar is encoded in the grammar file, which is optional for PC-KIMMO and, consequently, JKimmo. The grammar file has three sections: (i) feature abbreviations, (ii) category templates, and (iii) grammar rules. As in any feature-structure language, the grammar rules specify the feature constraints.

Table 1: Bengali transliteration scheme

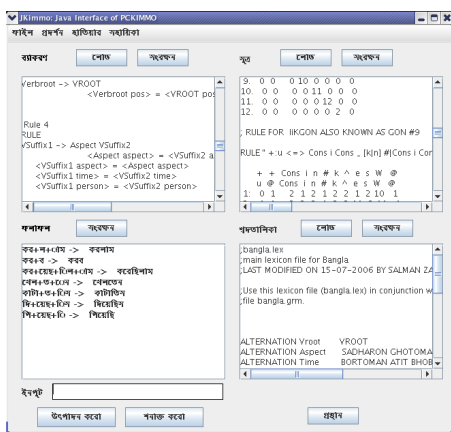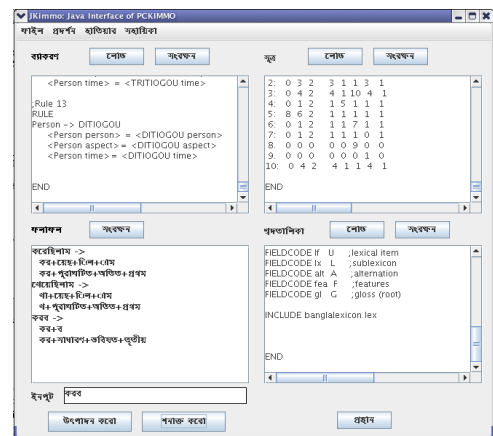| Bangla	Latin	Bangla	Latin	Bangla	Latin	Bangla	Latin	Bangla	Latin
্	^	া	a	গ	G	ণ	N	র	R
অ	A	ি	I	ঘ	G	ত	t	ল	L
আ	F	ী	I	ঙ	?	থ	T	শ	S
ই	H	ু	u	চ	C	দ	d	ষ	$
ঈ	L	ূ	U	ছ	C	ধ	D	স	S
উ	M	ৃ	R	জ	J	ন	n	হ	H
ঊ	Q	ে	e	ঝ	J	প	p	ড়	'
ঋ	V	ৈ	E	ঞ	Q	ফ	P	ঢ়	"
এ	W	ো	o	ট	V	ব	b	য়	Y
ঐ	X	ৌ	O	ঠ	W	ভ	B	ং	%
ও	Z	ক	k	ড	X	ম	m	ঃ	&
ঔ	F	খ	K	ঢ	Z	য	y	ঁ	~

Figure 4a: Generation example

Figure 4b: Recognition example

4.1.5. Localized interface. JKimmo provides the choice of language for its interface. Currently JKimmo only support Bangla and English language for its interface. New language can be added by adding a new java ResourceBundles property file for that language [15].

4.2. Algorithm

The algorithms used by the underlying morphological processor are described in [7]. JKimmo communicates with the PC-KIMMO API using the two data structures: KimmoData and KimmoResult. The KimmoData data structure collects the information used for data processing within the PC-Kimmo functions, and designed to hold as much of the processing parameters as possible to reduce the number of parameters needed for each function. The KimmoResult data structure contains a single result from one of the PC-Kimmo processing functions (applyKimmoGenerator, applyKimmoRecognizer). It can be used to build a linked list for ambiguous results. These algorithms pertain only to the communication between JKimmo interface and PC-KIMMO library. We have used JNI as a bridge between JKimmo interface and PC-KIMMO library. We have used both PC-KIMMO data structures to access internal components. The JNI also have some native methods for communication. This algorithm is for languages that do not use Latin script. For languages that uses Latin script just omit transliteration related portion.

4.2.1. The generator. This algorithm has some perquisites like transliteration file and rule file must be loaded. The algorithm works as follows:

1   If the input specified in the lexical form is empty but user click on generate button
  1.1 JKimmo will do nothing
2   For each input pair containing the first character in the lexical form as the lexical character, do the following steps:
  2.1 If input string is correct:
    2.1.1   Translate the Unicode string to Latin characters string.
    2.1.2   JKimmo interface calls *generate* native method with translated string as argument.
    2.1.3   Native method calls applyKimmoGenerator function of PC-KIMMO library. PC-KIMMO library save the result into result data structure.
    2.1.4   JKimmo interface now call *getResult* native method to get the result.
    2.1.5   Native method extracts the result (Latin character string) from KimmoResult data structure and sends to JKimmo interface.
    2.1.6   JKimmo interface translate the Latin characters string to Unicode string and show the result.
  2.2 If input string is wrong
    2.2.1   JKimmo will show a warning message and do nothing.

Figure 4a shows an example of JKimmo generation.

4.2.2. The recognizer. This algorithm also has some perquisites like transliteration file, rule file, lexicon must be loaded and grammar is optional. The algorithm works as follows:

1   If the input (surface) is empty but user click on recognize button
  1.1 JKimmo will do nothing
2   For each input pair containing the first character in the surface form as the lexical character, do the following steps:
  2.3 If input string is correct
    2.3.1   Translate the Unicode string to Latin characters string.
    2.3.2   JKimmo interface calls recognize native method with translated string as argument.
    2.3.3   Native method calls applyKimmoRecognizer function of PC-KIMMO library. PC-KIMMO library save the results into result data structure.
    2.3.4   JKimmo interface now call *getResult and getGloss* native method to get the results.
    2.3.5   Native method extracts the results (Latin character string) from KimmoResult data structure and send to JKimmo interface.
    2.3.6   JKimmo interface translate the Latin characters string to Unicode string and show the results.
  2.4 If input string is wrong
    2.4.1   JKimmo will show a warning message and do nothing.

Figure 4b shows an example of JKimmo recognition.

5. Conclusion

Our goal is to develop a reusable and robust open-source framework for computational morphological analysis of Bangla. We started with the existing efforts in defining the Bangla generative morphology for the rules, PC-KIMMO version 2 for the two-level morphological processor for the backend, and developed a Unicode-based multilingual interface, JKimmo, that can be used to experiment with Bangla morphology using Bangla language interface. JKimmo has been developed from the ground up as internationalized software, which means that it can be localized in any language using standard localization idioms such as property files and transliteration schemes.

Some of the limitations of the current implementation of JKimmo are however noteworthy. One of most useful features of PC-KIMMO version 2 is creating the parse tree when recognizing a surface form. JKimmo currently only shows the lexical form and its glosses. The other limitation is in error handling, specifically where the errors are generated by the back-end. The next release of JKimmo will correct both of the limitations.

6. Acknowledgement

7. References

[1] P. Sengupta and B.B. Chaudhuri, "Morphological processing of Indian languages for lexical interaction with application to spelling error correction", *Sadhana, Vol. 21, Part. 3*, 1996, pp. 363-380.

[2] S. Bhattacharya, M. Choudhury, S. Sarkar and A. Basu, "Inflectional Morphology Synthesis for Bengali Noun, Pro-noun and Verb Systems", *Proc. of the National Conference on Computer Processing of Bangla*, Dhaka, Bangladesh, March, 2005, pp. 34 - 43.

[3] S. Dasgupta and M. Khan, "Morphological Parsing of Bangla Words Using PC-KIMMO", *Proc. 7th International Conference on Computer and Information Technology, ICCIT 2004*, Dhaka, Bangladesh, Dec., 2004.

[4] S. Dasgupta and M. Khan, "Feature Unification for Morphological Parsing in Bangla", *Proc. 7th International Conference on Computer and Information Technology, ICCIT 2004*, Dhaka, Bangladesh, 2004.

[5] K. Koskenniemi, "Two-level morphology: a general computational model for word-form recognition and production.", *Publication No. 11. Helsinki: University of Helsinki Department of General Linguistics*, 1983.

[6] E.L. Antworth, "PC-KIMMO: a two-level processor for morphological analysis", *Occasional Publications in Academic Computing No. 16*, Dallas, TX: Summer Institute of Linguistics, 1990.

[7] E.L. Antworth. "Morphological Parsing with Unification-based Word Grammar", A paper presented at North Texas Natural Language Processing Workshop, May 23, 1994.

[8] PC-KIMMO available at http://www.sil.org/pckimmo/

[9] Unicode 4.1 specification, available from http://www.unicode.org/

[10] Java Native Interface Documentation available at http://java.sun.com/j2se/1.4.2/docs/guide/jni/

[11] Pykimmo is available at http://web.mit.edu/course/6/6.863/pykimmo/

[12] Bengali Morphological Analyzer demo, available at www.mla.iitkgp.ernet.in/morph_analyzer.html

[13] K.R. Beesley, "Finite-State Morphological Analysis and Generation of Arabic at Xerox Research: Status and Plans in 2001", ACL Work-shop on Arabic Language Processing: Status and prospects (Invited talk), 2001.

[14] S.M. Shieber, "An introduction to unification-based approaches to grammar", CSLI Lecture Notes No. 4. Stanford, CA, 1986.

[15] Java Localization documentation at http://java.sun.com/developer/technicalArticles/Intl/ResourceBundles/