JKimmo: A Multilingual Computational Morphology Framework for PC-KIMMO

Md. Zahurul Islam and Mumit Khan
Center for Research on Bangla Language Processing, 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) framework allows easy localization in other languages as well, using a property file for the interface definitions and a transliteration scheme for the analysis.

Keywords: Computational Linguistics, JKimmo, PC-KIMMO, Java Native Interface.

I. 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 usable 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 interface to PC-KIMMO backend.

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

II. 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. The Xerox system accepts modern standard Arabic words and returns morphological analyses and glosses. 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.

III. METHODOLOGY

A. PC-KIMMO OVERVIEW

PC-KIMMO version 1 is a morphological analyzer based on Kimmo Koskenniemi's model of two-level morphology [5]. While PC-KIMMO is adequate to decompose a word into morphemes, it is not able directly to compute the part of speech of a derivationally complex word or return a word's inflectional features – precisely the information required for syntactic parsing. Koskenniemi's model of two-level morphology was based on the traditional distinction that linguists make between *morphotactics,* which enumerates the inventory of morphemes and specifies in what order they can occur, and *morphophonemics,* which accounts for alternate forms or "spellings" of morphemes according to the phonological contexts in which they occur. For example, the Bengali word খেয়েছি is analyzed morphotactically as the stem খা followed by two the suffixes -য়ে and ছি. However, the addition of the suffixes -য়েছি apparently causes the replace া by ে of খা*;* thus খা and খে are allomorphs or alternate forms of the same morpheme. Koskenniemi's model is "two-level" in the sense that a word is represented as a direct, letter-for-letter correspondence between its lexical or underlying form and its surface form. For example, the word খেয়েছি is given this two-level representation (where + is a morpheme boundary symbol and *0* is a null character):

Lexical form: খ া + য় ে ছ ি

Surface form: খ ে 0 য় ে ছ ি

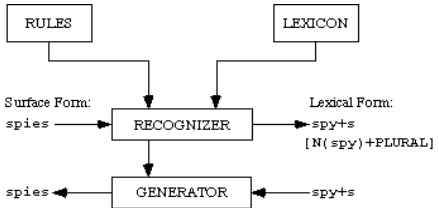

Fig. 1 Main components of Karttunen's KIMMO parser version 1

PC-KIMMO version 1 has a one major deficiency, which may result in incorrect output in addition to the correct one when used in the recognition mode. Version 2 of PC-KIMMO was developed specifically to correct this deficiency. It does so by adding a third analytical component, a word grammar [7]. The word grammar is a unification-based chart parser (based on the PATR-II formalism described by Schieber in [14]) that provides parse trees and feature structures. The chart parser was originally designed for syntactic parsing. Just as a sentence parser produces a parse tree with words as its leaf nodes, a word parser produces a parse tree with morphemes as its leaf nodes. PC-KIMMO has two functional components: *generation* and *recognition*.

B. GENERATION

The PC-KIMMO's generator function recursively computes surface forms from a lexical form using a set of two-level rules expressed as finite state automata. The generator function does not make use of the lexicon. This means that it will accept input forms that are not found in the lexicon or that even violate the lexicon's constraints on morpheme order, and will still apply the phonological rules to them. To produce a surface form from a lexical form, the generator processes the input form one character at a time, left to right. For each lexical character, it tries every surface character that has been declared as corresponding to it in a feasible pair sanctioned by the description. The generator function has these inputs:

Lexical Form

Initially the input form, this string contains whatever is left to process. As the function is recursively called, this string gets shorter as the result string gets longer.

Result

Initially empty, this string contains the results of the generator up to the point of the current function call.

Rules

This is the set of active finite state automata defined for this language.

Configuration

This is an array representing the current state of all rules (automata). Initially, all states are set to 1.

C. RECOGNITION

The PC-KIMMO's recognizer function recursively computes lexical forms from a surface form using a lexicon and a set of two-level rules expressed as finite state automata. The recognizer function operates in a way similar to the generator, only in a surface to lexical direction. The recognizer processes the surface input form one character at a time, left to right. For each surface character, it tries every lexical character that has been declared as corresponding to it in a feasible pair sanctioned by the description.

The recognizer also consults the lexicon. The lexical items recorded in the lexicon are structured as a letter tree. When the recognizer tries a lexical character, it moves down the branch of the letter tree that has that character as its head node. If there is no branch starting with that letter, the lexicon blocks further progress and forces the recognizer to backtrack and try a different lexical character.

The recognizer function has these inputs:

Surface form

Initially the input form, this string contains whatever is left to process. As the function is recursively called, this string shorter as the result string gets longer.

Result

Initially empty, this string contains the results of the recognizer up to the point of the current function call.

Gloss

Initially empty, this string contains glosses for the lexical items contains in the result string.

Rules

This set of active finite state automata defined for this language.

Configuration

This is an array representing the current state of all rules (automata). Initially, all states are set to 1.

IV. 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.

A. JKIMMO COMPONENTS

A.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.

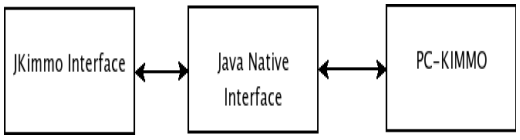

Fig 2: Communication protocol of JKimmo and PC-KIMMO

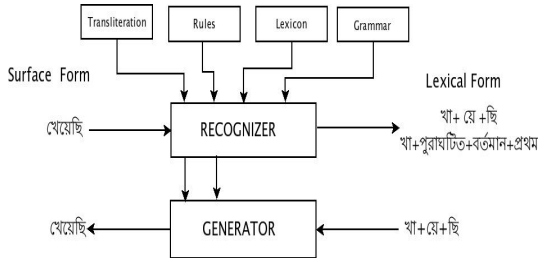

Fig 3: Main components of JKimmo

A.2 Rule File

Two level orthographic rules are required for JKimmo. The rule file is same as PC-KIMMO rule file. 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. 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.

A.3 Lexicon File

JKimmo lists lexical items (indivisible words and morphemes) in their underlying forms, and encodes morphotactic constraints. Its main job is to decompose a word into a sequence of 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. 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. To write lexicons that will be used in JKimmo for language that uses other than Latin script then we has to follow the transliteration scheme.

A.4 Grammar File

Grammar is optional for both JKimmo and PC-KIMMO. When the morphemes are given by the

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	ঁ	~

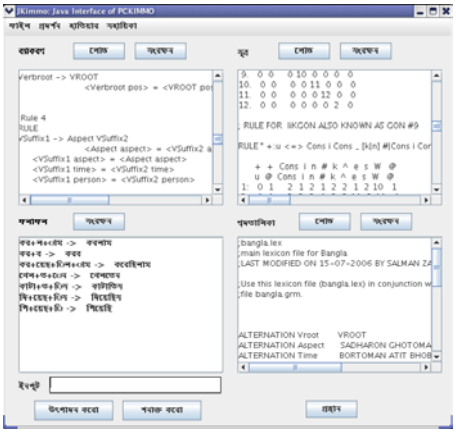

Fig 4.a: Generation Example

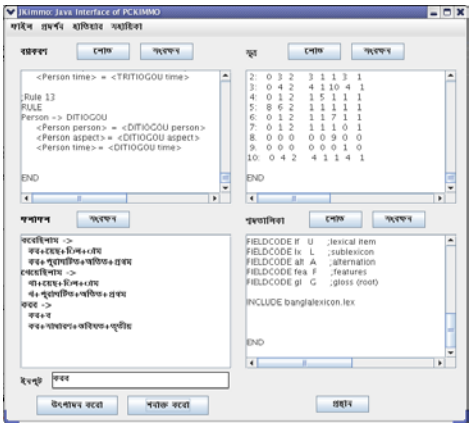

Fig 4.b: Recognition Example

Lexicon section they are combined using a word parser. Here correlations between different morphemes are considered using feature unification. The grammar file contains three sections. The first section of the grammar file contains feature abbreviations. Feature abbreviations can be used either in lexical entries or in grammar rules and are expanded by "Let" statements. For example, the feature abbreviation *pl* is expanded into the feature structure [number: PL]. The second section of the grammar file contains category templates. These are feature specifications that are attached to lexical categories such as Noun and Adjective. This greatly reduces the amount of information that must be stored in the lexicon. For example, the statement Let N be <number> = SG means that all nouns are assigned singular number. The third section of the grammar file contains the word grammar rules. Associated with each rule are feature constraints. A feature constraint consists of two feature structures that must unify with each other. Feature constraints have two functions: they constrain the operation of a rule and they pass features from one node to another up the parse tree.

A.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].

B. ALGORITHM

The algorithms used by PC-KIMMO to generate and recognize are based on descriptions in Karttunen 1983. JKimmo uses PC-KIMMO library and PC-KIMMO has two data structures: KimmoData and KimmoResult. The KimmoData data structure collects the information used for data processing within the PC-Kimmo functions. Its

general purpose is to reduce the number of parameters needed by the various functions. 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 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 dose not use Latin script. For languages that uses Latin script just omit transliteration related portion.

B.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 (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.

B.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:

2.1 If input string is correct

2.1.1 Translate the Unicode string to Latin characters string.

2.1.2 JKimmo interface calls recognize native method with translated string as argument.

2.1.3 Native method calls *applyKimmoRecognizer* function of PC-KIMMO library. PC-KIMMO library save the results into result data structure.

2.1.4 JKimmo interface now call *getResult and getGloss* native method to get the results.

2.1.5 Native method extracts the results (Latin character string) from KimmoResult data structure and send to JKimmo interface.

2.1.6 JKimmo interface translate the Latin characters string to Unicode string and show the results.

2.2 If input string is wrong

2.2.1 JKimmo will show a warning message and do nothing.

V. 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.

VI. ACKNOWLEDGEMENT

This work has been supported in part by the PAN Localization Project (www.panl10n.net), grant from the International Development Research Center, Ottawa, Canada, administrated through Center for Research in Urdu Language Processing, National University of Computer and Emerging Sciences, Pakistan. We would also like to thank Arnab Zaheen, Kamrul Hayder, Naira Khan and other members of our research group.

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, pp. 363-380 (1996).

[2] Samit Bhattacharya, Monojit Choudhury, Sudeshna Sarkar and Anupam Basu, "Inflectional Morphology Synthesis for Bengali Noun, Pronoun and Verb Systems", Proc. of the National Conference on Computer Processing of Bangla (NCCPB 05), pp. 34 - 43, Dhaka, Bangladesh, March, 2005.

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

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

[5] Koskenniemi, Kimmo. "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] Antworth, Evan L. "PC-KIMMO: a two-level processor for morphological analysis", Occasional Publications in Academic Computing No. 16. Dallas, TX: Summer Institute of Linguistics (1990).

[7] Antworth, Evan L. "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/

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

[14] Stuart 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/