Sādhanā, Vol. 21, Part 3, June 1996, pp. 363–380. © Printed in India.

Morphological processing of Indian languages for lexical interaction with application to spelling error correction

P SENGUPTA and B B CHAUDHURI

Computer Vision and Pattern Recognition Unit, Indian Statistical Institute, 203 B T Road, Calcutta 700 035, India
email: [sprobal, bbc]@isical.ernet.in

Abstract. An NLP system for Indian languages should have a lexical sub-system that is driven by a morphological analyzer. Such an analyzer should be able to parse a word into its constituent morphemes and obtain lexical projection of the word as a unification of the projections of the constituent morphemes. Lexical projections considered here are *f-structures* of the Lexical Functional Grammar (LFG). A formalism has been proposed, by which the lexicon writer may specify the lexicon in four levels. The specifications are compiled into a stored lexical knowledge base on one hand and a formulation of derivational morphology called Augmented Finite State Automata (AFSA) on the other to achieve a compact lexical representation. The aspects of AFSA, especially its power of morphological parsing of words in a computationally attractive manner, has been discussed. An additional utility of the AFSA, in the form of spelling error corrector, has also been discussed. Bangla, or Bengali is considered as a case study.

Implementation notes based on object-oriented programming principles has been provided.

Keywords. Natural language processing; morphological sub-system; lexical representation; augmented finite state automata; spelling corrector; object-oriented implementation.

1. Introduction

As most Indian languages are richly inflectional, a realistic natural language processing (NLP) system for any such language should have a morphological sub-system for *parsing* surface forms of words into its constituent morphemes. Such a sub-system achieves reduction in *lexical redundancy* and compactness of lexical representation. During the course of NLP research, the present authors have proposed a formalism for *lexical specification* that leads to a compact *lexical representation* for an inflectional Indian language (Sengupta & Chaudhuri 1993). The formalism performs efficient parsing of words leading

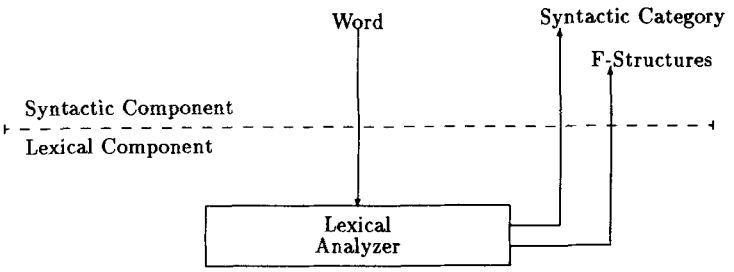


Figure 1. Conventional lexical interaction.

to *lexical projection* in the form required by a Lexical Functional Grammar (LFG) (Kaplan & Bresnan 1982). While conventional lexical analysis may be explained by figure 1, our proposed morphosyntactic analyzer performs lexical analysis as shown in figure 2. A detailed description of the formalism may be found in Sengupta (1994).

The central idea of morphological parsing of words is tackled in our formalism by an *Augmented Finite State Automata* or AFSA that has been proposed by us. We have not only studied the formal aspects of the AFSA but have also attempted an object-oriented implementation of the same. The efficacy of the AFSA as a tool for dictionary storage with spelling error detection/correction has also been investigated.

The importance of morphological processing is an established fact. Originally dealt with at reasonable depth by Kaplan & Bresnan (1982), in contemporary works, the *two-level* approach of Koskenniemi (1983) is most well known. The approach has been extensively reviewed (Gazder 1985) and used in implemented frameworks (Ritchie *et al* 1987). Our proposed recognition system is intermediate between the approaches of Kaplan *et al* and Koskenniemi.

In § 2, we provide a general background of lexical analysis of Indian languages. Lexical specification and representation as suggested in our scheme are taken up in § 3 and § 4, respectively, the AFSA being introduced in the latter section. Spelling correction with the AFSA is discussed in § 5 and notes on Object Oriented implementation of the lexical analyzer, especially the AFSA are included in § 6.

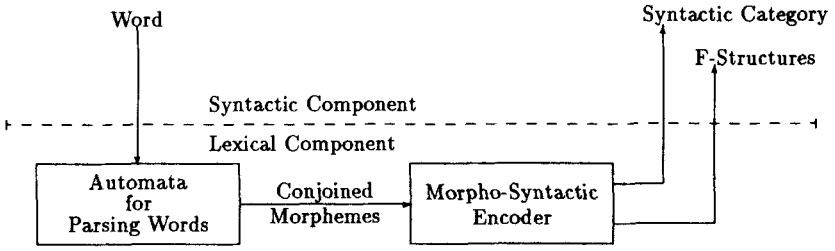


Figure 2. Lexical interaction with morpho-syntactic analysis.

2. General background

The constituent morphemes of words in an Indian language are *Stems* which form the essential component of words providing the meaning and *Affixes*. Depending on position of conjoining, an affix is a *prefix*, an internal *affix* or a *declension*. We shall however keep prefixes out of the purview of the present paper. Words are produced as a result of conjoinings between a stem and an appropriate number of affixes (may be none), the last of which will be called a *declension*. Every constituent morpheme of a word contributes to its overall linguistic property. The morphemes may be partitioned into several classes with morpho-syntactic rules, that are essentially non-recursive (i.e. regular), restricting conjoinings among them. Another class of rules of generative morphology called *spelling rules*, are concerned with morpho-phonemic or morpho-graphemic restructuring of symbols at the boundaries of two conjoining morphemes. The spelling rules make the job of detection of morphemic boundaries more difficult.

The basic idea of our proposed formalism is to unite the lexicon, the lexical description and the surface description into an integrated system. A few major aspects of the formalism are:

- **Stored lexical knowledge bases** for morpheme classes, their inter-relationships and **spelling rules**.
- A **Lexical Specification** phase in which the lexicon writer imparts linguistic knowledge to build up the above knowledge bases.
- A **Representation** phase in which the following two levels of lexical representation are generated:
  - A *Comprehensive Lexicon* for every morpheme containing every relevant lexical knowledge.
  - A formulation of the rules of morpho-syntax and derivational morphology in the form of an Augmented Finite State Automata (AFSA), which has the capability of parsing words and detecting morpheme boundaries, even when spelling deformities are present. The AFSA has pointer leading into the Comprehensive Lexicon for information retrieval.

The formalism is presented as a software tool to be used by a linguistic expert for specification of the lexicon of the target language. These specifications are to be 'compiled' into the proposed representation scheme.

3. Lexical specification

The linguistic expert would provide lexical description in four levels:

1. Specification of different morpheme classes.
2. Specification of rules of morpho-syntax.
3. Specification of a set of spelling rules.
4. Specification of a list of morphemes (constituting the vocabulary).

Unless otherwise mentioned, the specification format is like in LISP lists.

3.1 *Specification of morpheme classes*

The lexicon writer provides the names of different morpheme classes with the feature structures — a set of attribute names with a list of permitted attribute values (default in brackets, $ for strings). Directives STEM or END indicate that it is a stem class or declension class, respectively.

An Example Specification for morpheme classes VSTEM, NSTEM, VDEC, VCAUS, NCASE and DEF (representing verbal stems, nominal stems, verbal declensions, verbal causational affix, nominal case declensions and nominal definiteness affix, respectively):

```
(((VSTEM STEM END) ( (VALENCY  ([0] 1 2 3))
                     (PRED ($))
  ((NSTEM STEM END) ( (CAT ([material] abstract instrument place))
                     (ANIM (+ [-]))
                     (PRED ($))))
  ((VDEC END) ( (TENSE (PAST [HABIT] PRESENT ))
                (GNPH (0 1p [2p-0h] 2p-1h 3p-1h 2/3p-2h ))
              ) )
  ((VCAUS END) ( (CAUS (0 [1] 2))))
  ((NCASE END) ( (CASE ([NOM] DAT LOC OBLQ POSS ))))
  ((DEF END) ( (DEF ([YES] NO))))
)
```

3.2 *Specification of rules of morpho-syntax*

Rules of morpho-syntax (*Word Grammar* rules) govern formation of words from morphemes and lead to restrictions on a morpheme of one class following a morpheme of another in a word that has to be reflected in the AFSA. They are specified in the usual manner of specifying syntactic LFG rules, i.e. a lexical category is constituted of (denoted by $\longrightarrow$) morphemes of some classes. The rules may be annotated with f-structure schema. An *artificial morpheme class* called NULL may be used as a mechanism to project lexical information when certain morphemes are missing.

An example specification for verbs and nouns:

1. *VERB* $\longrightarrow$ *VSTEM* *NULL*
   $(\uparrow$ *TENSE*$) =$ *IMPER*
   $(\uparrow$ *GNPH*$) = 2p - 0h$
2. *VERB* $\longrightarrow$ *VSTEM VDEC*
3. *VERB* $\longrightarrow$ *VSTEM VCAUS VDEC*
4. *NOUN* $\longrightarrow$ *NSTEM* *NULL*
   $(\uparrow$ *CASE*$) =$ *NOM*
5. *NOUN* $\longrightarrow$ *NSTEM NCASE*
6. *NOUN* $\longrightarrow$ *NSTEM DEF NCASE*
7. *NOUN* $\longrightarrow$ *VSTEM* *VCAUS* *NCASE*
   $(\uparrow$ *CAT*$) =$ *gerund*
8. *NOUN* $\longrightarrow$ *VSTEM* *VCAUS* *DEF NCASE*
   $(\uparrow$ *CAT*$) =$ *gerund*

3.3 *Specification of spelling rules*

Morphological conjoining in Bangla, especially in the common dialect, fails to be *strictly concatenative* because there can be deformation of symbols around the boundary of

conjoining morphemes in many cases. For example, when the two morphemes **dhu** (verb stem "wash") and **ben** (a verb declension) conjoin, the resultant 'surface' form is **dhoben**. Note that the deformation of **u** to **o** in the example is very near the conjoining boundary. We therefore assume that words may have two different levels of representation. The representation that we write, read, speak and hear is the *surface* level representation, while at a *lexical* level of representation, morphological conjoining is strictly concatenative. Thus the lexical form of the above example would be **dhuben**.

Paul (1986) has closely studied spelling deformities in Bangla, especially of the verbal paradigm. From those observations, backed up by some of our own, we may list the following salient features of spelling deformity:

- Any deformity may be characterized entirely by the 'atomic' operations addition or deletion of a symbol and replacement of one (or more) symbol(s) by one (or more) symbol(s). If $\emptyset$ is used to denote the 'absence' of a symbol, all the above atomic operations may be expressed by the single operation – replacement.
- The innumerable individual instances of deformities may be reasonably generalized with "global" (i.e., applicable during a conjoining between any two morphemes) or "local" (applicable between pairs of morphemes from certain particular morpheme class pairs) applicabilities.

We now introduce the concept of spelling rules and specifications thereof.

Alphabet: The set of all characters that can constitute a lexical form (resp. surface form) of a word constitutes the alphabet Σ_L (resp. Σ_R). Σ_L and Σ_R are not not necessarily identical.

An l-r pair: We define $a : b$, where $a \in \Sigma_L \cup \emptyset$ and $b \in \Sigma_R \cup \emptyset$, to be an *l-r pair* or simply *pair*. A **union** $a_1|a_2|\ldots|a_k : b_1|b_2\ldots|b_k$ of pairs represents a disjunctive choice $a_i : b_i$, $1 \leq i \leq k$, from the k possible pairs.

An R-expression (RE): This is defined as a finite string of unions of pairs. For example, $RE = (a|b) : (x|x)\,\emptyset : y\,c : \emptyset$ is an R-Expression. If there are n unions in an R-Expression RE, it represents all distinct l-r-pairs of length n obtained by opening out the disjunctive choices of the unions.

String matching: A string t is *tail matched* by a string s if $\mid t \mid \geq \mid s \mid = n$, and the last n symbols of t spell out the string s. A string s *head matches* a string t if s is a prefix of t.

Spelling rule: A *spelling rule* is a template of the form $RE_1 + RE_2$, where RE_1 and RE_2 are R-expressions. The character + represents the abstract morpheme boundary. Intuitively, a rule $RE_1 + RE_2$ means the following. At the boundary between two morphemes, let the left morpheme tail match and right morpheme head match RE_1^L and RE_2^L, respectively. In the surface, the matched portion of the morphemes get translated to the corresponding

symbols from RE_1^R and RE_2^R. In other words, a *feasible pair* of morphemes [1] $m1 + m2$ matches a rule $RE_1 + RE_2$ and gets translated to surface $s_1t_1t_2s_2$ if and only if $m_1 = s_1r_1$, $m_2 = r_2s_2$, $r_1 \in RE_1^L, t_1 \in RE_1^S$, $r_2 \in RE_2^L$, $t_2 \in RE_2^S$ and r_1, t_1 and r_2, t_2 are corresponding pairs.

Often, as a shorthand, a spelling rule template may be written as:
$\ldots S : T \ldots + \ldots$,
where S and T are strings of identical size. This representation is a shorthand for $|\ S\ |$ different rules formed by taking members from S and T in order.

Local and global rules: Rules may be defined to be applicable at the boundary between only some specified pairs of morpheme classes. In such cases, the rules are adorned by the pairs of morpheme class. Such a rule will be called a *local* rule. All unadorned rules, called *global* rules, are applicable at the boundaries of all morphemic pairs. In the event of a rule clash between a local and a global rule, the local rule prevails. The specification format of spelling rules is:

`<Spelling Rule Template>` [at (M_1, M_2)]

where, M_1 and M_2 are morpheme classes. The "at" clause, if present, indicates a local rule. '0' is the null symbol and symbols 'V', 'C' and = represent the set of all vowels, set of all consonants and the entire alphabet set, respectively. Consider the following spelling rules as examples.

1. `V:V + a:o`
2. `eiuo:eiuo + 0:y e:e`
3. `V:V + ieu:000 nsk:nsk at VSTEM, VDEC`
4. `C:C a:e C:C + E:e at VSTEM, VDEC`
5. `a':i + i:e y:0 a':0 at VCAUS, VDEC`
6. `a':a' + 0:c 0:/ ch:ch at VCAUS, VDEC`

The first two of the rules are global. For a more comprehensive list of spelling rules for Bangla, see (Sengupta & Chaudhuri 1993; Sengupta 1994).

3.4 *Morpheme list specification*

The final level of lexical specification involves providing a list of morphemes.
Example:

```
 i) pa':  ( (VSTEM (VALENCY 1) (PRED 'get'))
            (NSTEM (CAT instrument) (PRED 'foot')))

ii) pa't: ( (VSTEM (VALENCY 1) (PRED 'lay')))
```

[1] We say that a pair of morpheme classes (M_1, M_2) is a *feasible* pair, if any morpheme of class M_2 can follow any morpheme of class M_1 in a word.

```
 iii) mar:  ( (VSTEM (PRED  'die') (CAUS 0)))

  iv) ma'r: ( (VSTEM (VALENCY 1) (PRED 'kill')))

   v) a':   ( (VCAUS))

  vi) t'a': ( (DEF))

 vii) er:   ( (NCASE (CASE POSS)))

viii) ke:   ( (NCASE (CASE DAT)))

  ix) e:    ( (NCASE (CASE LOC) (CAT place))
              (NCASE (CASE OBLQ) (CAT material))
              (VDEC) )

   x) E:    ( (VDEC (TENSE CONTINF) (GNPH 0)))

  xi) te:   ( (VDEC (GNPH 2p-1h))
              (VDEC (TENSE CONDINF) (GNPH 0))
              (NCASE (CASE LOC) (CAT place))
              (NCASE (CASE OBLQ) (CAT material)) )

 xii) ten:  ( (VDEC (TENSE HABIT) (GNPH 2/3p-2h)))
```

Observe the following in the above specification:

1. A morpheme may belong to multiple classes as in i), ix), xi).
2. There may be alternations in the lexical specification as in ix) xi).
3. A morpheme may be specified only with its class as in v), vi), ix).
4. There is no harm in re-specifying a default attribute value as in xii).
5. An attribute not in set of default attributes of a word may also be specified, as in iii).

4. Lexical representation

There are two levels of lexical representation:

- A *Comprehensive Lexicon* for every morpheme.
- An Augmented Finite State Automata (AFSA).

4.1 *The comprehensive lexicon*

The comprehensive lexicon is basically an indexed database of morphemes. The specification of an individual morpheme is first completed by copying the default specifications from the class it belongs. From this, tuples of the type:

$< Morpheme\ Class > (< Attribute\ Name/Path >< Attribute\ Value >)(\ldots)$

are created and stored in the database entry for the morpheme. There may be multiple tuples for the same morpheme.

4.2 *Introduction to the AFSA*

The *Augmented Finite State Automata* (AFSA) is our proposed tool for parsing words into constituent morphemes. In the normal mode of use, the AFSA accepts the surface representation of a word as input and ultimately generates index pointers into the comprehensive lexicon for the morphemes constituting the word. The lexical projection of the word can be recovered as a result of *unification* of the lexical projections of the constituent morphemes. We have assumed that the application of a rule at a boundary is *context-free*, neither affecting nor being affected by an earlier or later application of a rule at some other boundary.

The AFSA consists of a forest of *Directed Acyclic Graphs* (DAGs). Each DAG represents a finite state recognizer for a class of morphemes. However, there is a single DAG for all STEM type morphemes. The DAGs consist of two types of edges — *lexical* or *l*-edge and *surface* or *s*-edge. Transition along l-edges only from the root node to a terminal node of a DAG recognizes a lexical morpheme. Transition along s-edges however, recognize *one* surface form of some lexical morpheme. The different DAGs in the system are also interconnected by l- and s-edges. However, the inter-DAG edges are qualitatively different from intra-DAG edges. We call inter-DAG edges *active* and intra-DAG edges *passive*. The active edges encode the morpho-syntactic restrictions applicable for the language specified by the lexicon writer as described in § 3.2.

4.3 *Formal definition of the AFSA*

The AFSA consists of a forest of DAGs, where every DAG consists of:

a. A set of *nodes* representing *states*. Nodes are labelled as *passive*, *l-active* and/or *s-active*.

b. A set of *l-passive edges* between a pair of nodes in the same DAG.

c. A set of *s-passive* edges between a pair of nodes in the same DAG.

d. A set of *l-active edges* linking a (terminal) node of one DAG to the root node of another DAG. The word *active* node will be used interchangeably with terminal node.

e. A set of *s-active* edges linking a terminal node of one DAG to a root node of another DAG. Every s-active edge has a *disjunction of one or more* index pointers into the comprehensive lexicon. Every s-active node is associated with one or more s-active edges. Additionally, if the DAG to which an s-active node belongs represents a morpheme class with the END directive, the node has one *trivial* s-active edge. Unlike normal s-active edges, a trivial s-active edge does not link to any node in a different DAG. However, it has an index pointer into the Comprehensive Lexicon.

f. A set of *associations* connecting an l-active and one or more s-active nodes.

Every DAG has a unique *root l-node* and one or more *root s-nodes*. The root l-node is also a root s-nodes.

4.4 *Parsing in the AFSA*

Input: The surface representation of a word — *s*

Aim: To recover pointers into the Comprehensive Lexicon of the constituent morphemes of s.

Data structures: The AFSA and a *Stack* of quadruples $(dag, node, index, k)$, where *node* is an active node in DAG *dag*, *index* is an index pointer into the Comprehensive Lexicon and latest morphemic boundary is at the $k-1$-th symbol of the input.

Driving routine of the AFSA:

Step-1 $dag \leftarrow STEM$; $node \leftarrow$ root l-node of the STEM DAG; $k \leftarrow 1$. (Here k points to the character in s currently being scanned). Clear the Stack.

Step-2 If the end of string has not been reached, proceed to Step-3. Otherwise, check if *node* is an s-active node. If not, proceed to Step-5. Otherwise, let $p \leftarrow p_t$, where p_t is the trivial index pointer for *node*. *Push* $(dag, node, p, k)$ in *stack* and *exit* with success.

Step-3 If *node* is an s-active node, non-deterministically decide whether to make an active transition. If yes, let the chosen non-trivial s-active edge lead to node n in DAG d and let p be the index pointer of the chosen edge. *Push* (d, n, p, k) onto stack. Make $dag \leftarrow d$ and $node \leftarrow n$ and repeat step 3. If active transition is not taken, proceed to Step-4.

Step-4 If there is an s-passive edge in DAG *dag* from node *node* to node n on the k-th character of s, make $node \leftarrow n$; $k \leftarrow k+1$ and go to Step-2.

Step-5 If there is no s-active or s-passive transition possible in DAG *dag* from node *node* based on the k-th character of s, or if k points beyond the last character of s, *pop* $(dag, node, p, k)$ from *stack* (where p is a dummy variable) and resume in Step-2. If the *pop* operation fails, *exit* with failure, i.e. declare the input word to be ill-formed.

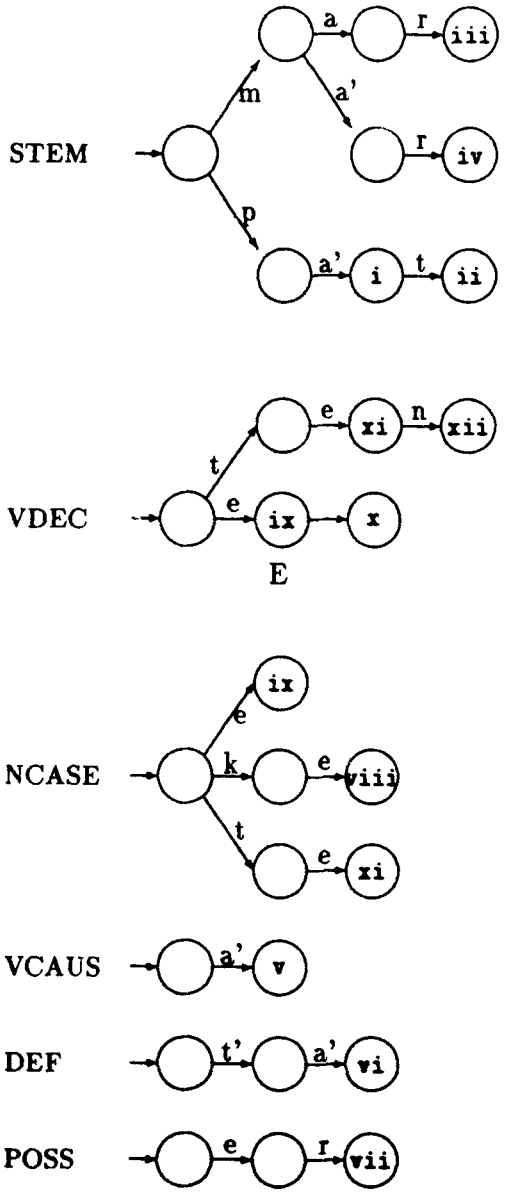
Output pointers to comprehensive lexicon: If the driving routine terminates successfully, the *stack* contains r, $r \geq 1$ quadruples $(d_1, n_1, p_1, k_1), (d_2, n_2, p_2, k_2), \ldots (d_r, n_r, p_r, k_r)$. The lexical projection of the word is the *union* of the r sets of schema obtained from the comprehensive lexicon by following the pointers $p_1, p_2, \ldots, p_r$.

4.5 *Automatic generation of AFSA*

The specifications provided by the lexicon writer are *compiled* into an AFSA. The compilation proceeds with two passes over the list of morphemes, along with an intermediate pass over the list of spelling rules. The compilation process is pre-processed by a pass over the list of morpheme classes. The second pass also consults the set of spelling rules. The compilation process has been discussed in detail in Sengupta (1994). The AFSA obtained after the first pass of compilation has been shown in figure 3 and the final AFSA produced is shown in figure 4.

4.6 *Complexity and related issues*

The worst case time complexity of parsing in AFSA is of exponential order, primarily due to the non-determinisms at various stages. Let there be n DAGs in the system corresponding



Edges are both l-passive and s-passive. Numbered circles are active nodes. Figures on terminal nodes indicate entry number of Morpheme Lexicon indexed by the nodes.

Figure 3. The AFSA after first pass of compilation.

to $n-1$ non-STEM morpheme classes and one common STEM DAG. The worst case *first level* non-determinism occurs at an active node (which is also a passive node) of the STEM DAG where the next symbol has active transitions to every $n-1$ non-STEM DAG as well as a passive transition. This gives rise to an n level non-determinacy. The worst case *second level* non-determinism is of order $n-1$ and may occur at an active node of a non-STEM

DAG, where there may be $n-2$ active transitions to the remaining $n-2$ non-STEM DAG (active transitions from a DAG to itself are not possible) as well as a passive transition. Similarly, k-th level non-determinism is of order $n-k+1$. Of course, while parsing a given word, there may be at most n levels of non-determinism, since otherwise there must be a cyclic active transition. As all non-determinisms are multiplicative, the worst case scenario during parsing could lead to $n!$ non-deterministic choice for every single symbol of the word, hence giving a $O(k^{n!})$ worst case complexity for a word of length k.

We have observed that while there may be many active transitions from an active node of the STEM DAG, active transitions from an active node of a non-STEM DAG is fewer. This is because, a non-STEM morpheme class may be followed by only a few other DAGs in morpho-syntax rules, allowing us to conclude that only first level non-determinism affects computational complexity appreciably. Pragmatic worst case complexity is therefore $O(k^n)$ — still exponential!

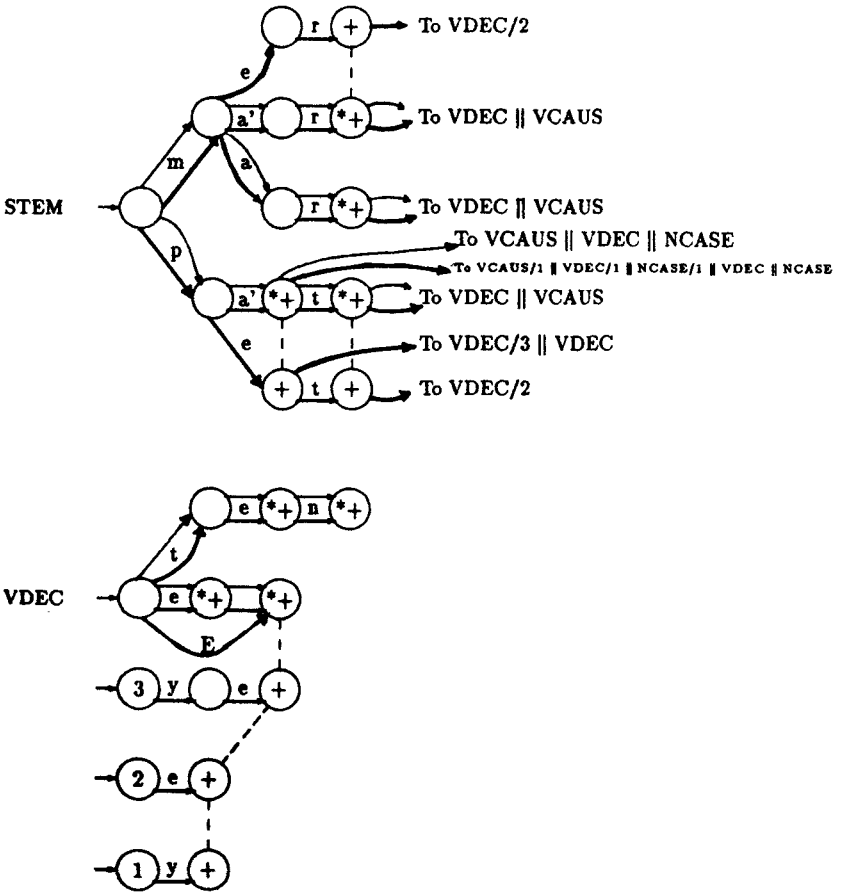


Figure 4. *Continued on next page.*

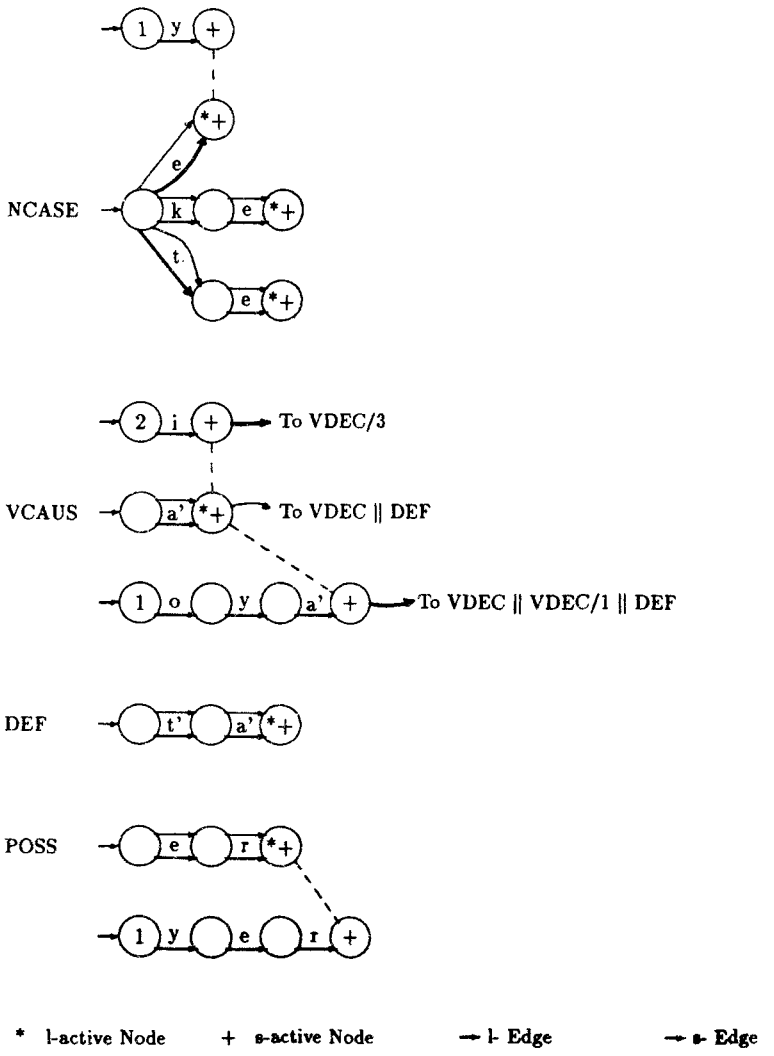


Figure 4. The final compiled AFSA.

Focusing entirely on first level (i.e. localized in the STEM DAG) non-determinism only, it is possible to reduce it further by associating a set of *lookahead* symbols with active edges. An active transition is taken only if the next symbol belongs to the lookahead set. A non-determinism is then encountered when all the following conditions hold:

- The present node is an active node.
- Total transitions (both active as well as passive) possible from the present node, with the next symbol as "lookahead", is more than one.

While worst case complexity still remains exponential, since non-determinism is now governed by two independent events of moderate probability, the practical complexity is quite low. It is difficult to analytically compute the average case complexity since it

is not easy to estimate the distribution of words being fed to the AFSA. We made the following short study for a moderate lexicon consisting of 565 stems of different classes. We identified the nodes in the STEM DAG that cause non-determinism, along with the offending characters.

The results are shown below:

Total number of stems	=	565
Total number of nodes	=	764
Total number of active nodes	=	551
Total number of offending active nodes	=	124

Of the 124 offending nodes (less than 17% of all nodes), in as high as 84, the offending symbol was **i**. Invariably, these nodes recognized verbal stems and the non-determinism was between active transitions to the VCAUS (since by the effect of some spelling rules, the causational affix **i** becomes **iy**, if the following verbal declension is E realized as **e** in the surface) and VDEC (since there are many declensions like **i, ite, ila**, etc., that begin with **i**. We fine-tuned our recognizer to lookahead to the second lookahead character at active nodes, if the next symbol is *i*. This leaves us with 60 isolated offending active nodes in the AFSA. The test runs show near linear (with respect to word length) run time complexity.

5. Use of AFSA in spelling correction

A spelling corrector is a computer program that takes a presumably misspelt word as input and suggests possible correctly spelt alternative words for the same. The suggested words are in a way at close *Hamming vicinity* of the input word. In general, an erroneous word may have been generated from a correct word due to one or more of the following:

- A character symbol at some position of the correct word has been substituted by some other character. This is the most general type of error, generally known as *substitution error.*
- A character symbol at some position of the correct word has been dropped inadvertently. This error is known as a *deletion error.*
- A spurious character has been inserted at some position of the correct word resulting in an *insertion error.*

If the *null* character, whose external manifestation in a string is absence of any symbol at its position, is considered to belong to the logical character set, the deletion and insertion error can be considered to be special cases of the substitution error. In deletion error, the deleted character is substituted by the *null* character while in insertion error, a *null* character is substituted by a non-null character.

Computing the precise Hamming vicinity of a word is very difficult. The accepted spelling correction techniques assume that errors accumulate with decreasing order of probability. Usually, as in our implementation, the probability of a single error (of either of the three types) is assumed to be 0.5. At every subsequent error, the probability hitherto encountered is halved. In this way, there is an informal relationship between the Hamming distance between a correct word and a (possibly) erroneous one to the probability of the latter. The lower the probability, the greater is the Hamming distance.

On principle, our spelling corrector proceeds as follows. Given an input (possibly erroneous) word, it goes about recognizing the same in the AFSA. During the process, it continuously builds up a list of possible words along with their probabilities as well as proper prefixes of such words and their probabilities. In every elementary step, yet another prefix is selected from the latter list proceeded with one more character (may be *null*) at the end. As a result of an elementary operation, more prefixes may be added to the list. However, if the chosen prefix was found to be of a probability below a threshold, i.e., it is already quite far away from any valid word, it may be discarded. If the newly generated prefix represents a full word, it is also appended to the list of possible words.

The AFSA can be easily used as the building block of a spelling corrector. Only the s-edges of the AFSA need be considered by a spelling corrector. In our implementation, described in more detail in Das (1994), the following auxiliary data structures are used:

- The `WordNode` structure is a 4-tuple *(str,pos,prob,node)*, where *str* is a string of characters representing a word-prefix. Recall that every valid word-prefix may be associated with one or more nodes of the AFSA, the multiplicity being due to the non-determinism introduced by the active nodes/edges. In the above tuple, *node* is one such node. *pos* is an integer denoting the position of the input word with which the prefix *str* is associated and *prob* is the hitherto computed probability of *str*.
- The Possible Word List *(PWL)* is a list of possible suggestions in case the input word is erroneous.
- The Search List *SL* is a list of prefixes to be considered, actually stored as a list of *WordNodes*.

In an elementary operation, a *WordNode* w_n is taken out from *SL* and processed upon. During this processing, $w_n.node$ becomes the *current node*. Depending upon the situation, three types of *search* moves may be necessary:

- If $w_n.node$ is a passive node, the move `PassiveSearch` carries out a one character lookahead making transitions along passive edges.
- If on the other hand $w_n.node$ is an active node, the move `ActiveSearch` carries out the lookahead from the root node to the DAGs to which there are active edges from $w_n.node$.
- It may happen that $w_n.pos$ indicates that scanning has proceeded beyond the last character of the input word. In such a case, the look ahead proceeds from $w_n.node$ till a terminal node is reached, i.e., a valid word is detected. This is taken care of by the move `ForwardSearch`.

The overall spelling corrector algorithm begins with an `inputWord` $c_0c_1 \ldots c_{size-1}$, empty `PWL` and `SL` containing the only *WordNode* $w_n = (\text{""}, 0, 1.0, n)$, where, n is the root node of the STEM DAG. Note that the initial probability $w_n.prob$ is 1.0.
The main loop of the spelling corrector is as follows:

```
while (SL is not empty) {
  wn = Some item taken out of SL;
  if (wn.node is terminal &&
      0.5**(size-wn.pos)>threshold) { /* ** is to-the-power */
```

```
/* Word recognized at the node is close enough */
    add wn.str to PWL;
  }
  if (wn.prob>threshold) { /* Proceed only if probability is sufficient */
    if (wn.node is active) ActiveSearch(wn);
    if (wn.pos<size) { /* Still possible to associate */
      PassiveSearch(wn);
    }
    else {
      ForwardSearch(wn);
    }
  }
  throw away wn;
}
```

It is appropriate to describe at least the `PassiveSearch` module here.

```
void PassiveSearch(WordNode wn)
{
  if (found (n1=passive transition on inputWord[wn.pos] from wn.node)) {
/* Normal proceed. Do not alter probability */
    add WordNode(wn.str||inputWord[wn.pos],wn.pos+1,wn.prob,n1)
    to SL;
  }
  if (wn.pos<size-1 &&
      found (n1=passive transition on inputWord[wn.pos+1] from wn.node)) {
/* Possibility of detection of an Insertion error.
   The insert character is inputWord[pos].
   This is notionally deleted. */
    add WordNode(wn.str||inputWord[wn.pos+1],wn.pos+2,wn.prob/2,n1)
    to SL;
  }
  k = number of passive edges from wn.node
  for (i=0;i<k-1;i++) { /* for all edges do */
    n1 = node reached following i-th passive edge from wn.node;
    c = character on the i-th passive edge;
    create wn1 = WordNode(wn.str||c,wn.pos,wn.prob,n1);
/* wn1 shall take care of Deletion and/or Substitution error.
   Consolidation done below. */
    if (n1 is active node) ActiveSearch(wn1);
/* Taking active closure of n1 before handling itself */
    if (found (n2=passive transition on inputWord[wn.pos] from n1)) {
/* Taking care of Deletion of c at wn.pos-th position */
      add WordNode(wn.str||c||inputWord[wn.pos],wn.pos+1,wn.prob/2,n2)
      to SL;
    }
    if (wn.pos<size-1 &&
        found (n2=passive transition on inputWord[wn.pos+1] from n1)) {
/* Taking care of Substitution of c by inputWord[wn.pos+1]
   at wn.pos-th position */
      add WordNode(wn.str||c||inputWord[wn.pos],wn.pos+2,wn.prob/2,n2)
      to SL;
    }
  } /* end for */
} /* end function Passive Search */
```

Consider as an example, the input **pa're** and the tiny AFSA as shown in the figure. (Although **pa're** is a valid Bangla word, the AFSA of the figure does not recognize it.) With a threshold of 0.125 (i.e. total error not exceeding two), our spelling corrector would give suggestions **mere, ma're, mare, ma'ra', mara', pa'te, pete, pa'ten, pa'y, pa'ke**.

6. Implementation notes

The major object (also known as `classes` in C++ parlance) used are:

Passive Node: This `class` represents a passive node of the AFSA. It contains lexical and surface back passive edges to the previous node. There is a *container* of pairs (c, lp), where c is a symbol from the lexical alphabet and lp is a pointer to another node. Thus the items of this container are l-passive edges. There is another similar container for s-passive edges. Member functions includes forward/ reverse lexical/ surface transition procedures.

Active Node: Is inherited from PassiveNode. There are two additional containers for l- and s-active edges. Each contained item of these containers is an Active Edge. There are member functions for making l- and s-active transitions. Another important member data of ActiveNode is $pLexList$, which is a pointer to a *linked list* of pointers to the Lexicon `class`. All members of the list are assumed to be disjunctively pointing to different lexical entries.

Active Edge: Is a pair (*pLook*, *pNode*) where, $pLook$ is a pointer to a *set* of "lookahead" symbols and $pNode$ is the root of the DAG to where the active transition is taken.

Lexicon: This `class` does not have any important member data. It serves the purpose of interacting with the lexicon through database management routines to draw lexical projections of morphemes.

Other `classes` used are `FStructure, Pair`, etc., along with member functions to perform Locate and Merge operations of LFG. These `classes` are discussed in more detail in Sengupta (1994).

Object returned by the lexical sub-system: As shown in figure 2, there are two major contents in the object returned by the lexical sub-system — Word category of the input word being parsed and the f-structure. In practice, the structure of the object returned by the lexical component is not exactly so. The reason is that in our proposed system, there is an intervening supra-lexical layer (Sengupta & Chaudhuri 1996; Sengupta 1994) between the syntactic component and the lexical component. The components of the object actually returned is as follows:

- The input word itself.
- Word Category. Indicating the tentative word category of the input word.
- A list of constituent morphemes.
- A list of (codes for) morpheme classes $M_1, M_2, \cdots$ for the constituent morphemes of the word. Thus, the details of morpho-syntactic composition of the input word is also returned.
- A list of (pointers to) f-structures $F_1, F_2, \cdots$ for the constituent morphemes.

- A (pointer to) the unified f-structure of the word.
- A (pointer to) the semantic clause of the word.

Interaction with the lexical sub-system is carried out through the following function:

```
int primitiveLexAnalysis( char          *word,
                                              \\ Input word
                          LexPrimitive *lexPrim
                                               \\ Returned object
                        ); // Returns TRUE
                           //         if parse successful,
                           // FALSE otherwise
```

7. Discussions

The formalism proposed here has been tried out for a medium sized lexicon in Bangla consisting of about three hundred verb stems, one thousand nominal stems and a few stems of some other classes. There are about forty verb declensions and about ten case declensions capable of generating 2,400 VERBs (causated and non-causated) and 10,000 NOUNs. The results of parsing obtained were very satisfactory, with near linear recognition time complexity.

The lexical projection of a compound stem produced as a result of *euphony* of two stems can be derived from the conjoining stems. As a result, euphony is a more attractive subject of study than prefixes. If an AFSA is used to perform rule-based de-euphonization, it may have too many self-loops, resulting in reduction of efficiency. However, at the level of sentential syntax analysis, considerable advantages may be derived from rule-based de-euphonization. We have made some initial studies (Panda 1992). However, it is too early to report any major achievement.

The biggest advantage of our formalism is the compactness and lucidity of representation. The recognizers are finite-state networks — a well-studied formalism. The representation scheme is easy to understand and quite flexible. The underlying LFG formalism permits a broad generalization across morpheme boundaries as in the last example taken up in § 4.5 (i.e. **mara'y**). Comparing our formalism with Koskenniemi's two level approach, we find that the latter does not incorporate morpho-syntactic restrictions in the automata itself.

Finally, the potential power of the AFSA as the building block of a spelling corrector has been amply demonstrated.

References

Das M 1994 *Implementation of a spelling checker for Bengali language*. M Tech dissertation, Indian Statistical Institute, Calcutta

Gazder G 1985 Review article: Finite state morphology. *Linguistics* 23: 597–607

Kaplan R M, Kay M 1981 Phonological rules and finite state transducers. *ACL/LSA Assoc. Comput. Linguistics* (New York: Linguistic Soc. Am.)

Kaplan R M, Bresnan J 1982 Lexical fuctional grammar: A formal system for grammatical representation. In *The mental representation of grammatical relations* (ed.) Joan Bresnan (Cambridge, MA: MIT Press) pp 173–281

Koskenniemi K 1983 Two level model for morphological analysis. In *Proceedings of the International Joint Conference on Artificial Intelligence* Karlsruhe, West Germany, pp 683–685

Panda H R 1992 *Rule based "Sandhi Bicched" (de-euphonization) of Bengali.* M Tech dissertation, Indian Statistical Institute, Calcutta

Paul J 1986 Bangla verb morphology: A reconsideration. *Indian Linguistics* 47(1–4): 73–79

Ritchie G D, Pulman S G, Black A W, Russell G J 1987 A computational framework for lexical description. *Comput. Linguistics* 13: 290–307

Sengupta P, Chaudhuri B B 1993 A morpho-syntactic analysis based lexical sub-system. *Int. J. Pattern Recogn. Artif. Intell.* 7: 595–619

Sengupta P, Chaudhuri B B 1996 Projection of multi-worded lexical entities in an inflectional language. *Int. J. Pattern Recogn. Artif. Intell.* (in press)

Sengupta P 1994 *On lexical and syntactic processing of Bangla language by computer.* Ph D dissertation, Indian Statistical Institute, Calcutta