M.Tech (Computer Science) Dissertation Series

Rule Based *"Sandhi Bicched" (de-euphonization)* of Bengali

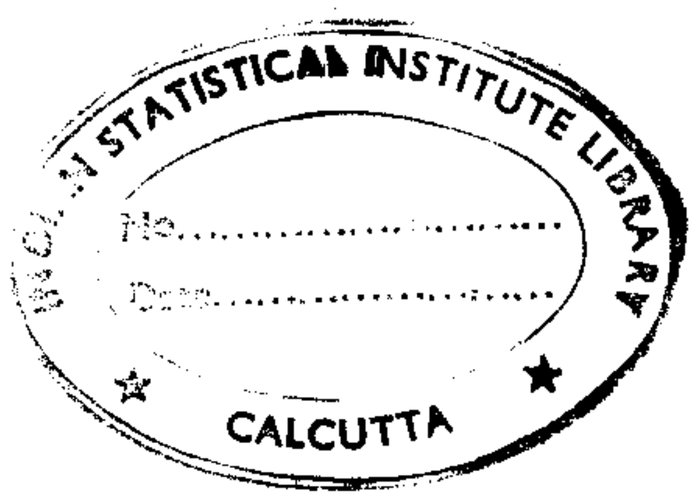
a dissertation submitted in partial fulfilment of the requirements for M. Tech. (Computer Science) degree of the Indian Statistical Institute

By

HEMANTA RANJAN PANDA

under the supervision of

Probal Sengupta
B.B.Chaudhuri



INDIAN STATISTICAL INSTITUTE
203,Barrackpore Trunk Road, Calcutta-700 035

ABSTRACT

In an inflectional language, words are formed as a result of conjoining of more primitive linguistic entities called *morphemes*. Meanings of words are *derived* from the meanings of the constituent morphemes. Since most Indian languages are richly inflectional, efficient morphological level processing is necessary in Natural Language Processing(NLP) systems for Indian languages. The major responsibility of a morphological sub-system in an overall NLP system is to 'parse' a word in to its constituent morphemes. Quite often, a stem morpheme may be considered to be constituted as a result of combining two simpler stems. Sometimes the stems that combine undergo "euphony" or 'Sandhi', i.e., the final portion of the left stem and the initial portion of the right stem undergo a deformation in spelling. The deformations along the boundary are guided by well defined morpho-rules. The system described here augments an earlier morphological processor for Bengali in incorporating ability to perform fragmentation of stems. By incorporating this feature, euphonized stems need not be explicitly stored. As a result, a high degree of redundany is achieved in the storage of the lexical database. The technique involves imparting additional power to finite state automata.

ACKNOWLEDGMENTS

I am indebted to Mr. Probal Sengupta and Dr. B.B. Chaudhuri for introducing me to the areas of Natural Language Processing(NLP) and guiding me throughout this work. Interests of Suresh and Jyoti in the present work are thankfully acknowledged.

CONTENTS

1 INTRODUCTION

Natural Language Processing (NLP) and an inter-disciplinary field called Computational Linguistics(CL) emerge from research in Artificial Intelligence (AI). NLP is concerned with building of intelligent models of production and comprehension of natural languages like English, Hindi, Bengali, Oriya etc. The primary motivations for this type of research lie in:

1. Building intelligent computer systems capable of human-like interfaces as in natural language query system for databases, automatic machine-translation(MT) from one language to another, text and speech analysis/understanding systems, computer-aided-instruction(CAI) systems etc.

2. Gaining a better understanding of how humans communicate using natural languages which is still an open area of research for psycholinguists and theoretical linguists.

NLP research can be broadly classified in the following few (possibly overlapping) levels depending upon the extent of knowledge they use:

MORPHOLOGICAL LEVEL: It concerns with how words are constructed out of more basic meaning units called *morphemes.*

SYNTACTIC LEVEL: It concerns with how words can be put together to form sentences that look correct in language and how a word is related to another in a sentence.

SEMANTIC LEVEL: It deals with how word meanings can combine to form sentence/discourse meanings.

PRAGMATIC LEVEL: Deals with how sentences are used in different contexts and how context affects the interpretation of the sentence.

For a computer based NLP system to work reasonably, there must be a sub-system concerned with interaction with the *lexicon.* The lexicon is

the global knowledge base for words. For simple languages like computer programming languages, the lexicon has a simple structure. However, in inflectional languages, words are formed as a result of conjoining of more primitive linguistic entities called *morphemes*. Meanings of words are *derived* from the meanings of the constituent morphemes. As a result, the lexical sub-system requires to be quite complex. The morphological level processing normally performs lexical interaction in inflection languages. Since most Indian languages are richly inflectional, efficient morphological level processing is necessary in NLP systems for Indian languages. The major responsibility of a morphological sub-system in an overall NLP system is to "parse" a word into its constituent morphemes. Parsing a word is a non-trivial problem because:

- There are no conspicuous delimiters between morphemes in a word unlike in sentences where words are clearly delimited.
- Quite often the participating morphemes get deformed at the conjoining boundary. As a result, a word is not always a simple concatenation of the conjoining morphemes.

A major advantage of having a morphological sub-system is that the lexical properties of words can be *derived* from the constituent morphemes. As a result, a high degree of redundancy is achieved in the lexicon. Physically, the redundancy is manifested in lower space requirement for storing large lexicons.

In classical lexical analysis systems, as in compilers for programming languages, the recognizers are *Finite State Automata* [1]. Even for natural languages, use of finite state automata (or finite state transition networks) is very much in vogue. A good review of application of finite state transition networks(FSTNs) or deterministic finite state automata (DFAs) to word recognition may be found in [3]. Some authors [5,6], have even come up with powerful enhancements in classical DFAs which have been applied for recognizing words in languages with complicated inflectional structures.

Words in Bangla and other Indian languages are composed of a mandatory *stem* morpheme, followed by other morphemes. The stem morpheme, or

simply the stem, provides the basic meaning of the word. Depending upon the *syntactic category* (like noun, verb, etc.) to which the stem belongs, certain *morpho-syntactic* rules are provided in the language which determine what other morpheme(s) can follow the stem in the formation of the word. Normally, the morphemes following the stem, called *inflections*, have no meaning in isolation but play their roles in determining the overall linguistic perspective of the word.

Example: Word **KARaIBEN** is formed from **KAR:a:IBEN**, where : indicates morphemic boundary. In this example, morpheme **KAR** is a verb stem ("Dhatu") meaning 'do', **a** is the verb causational affix ("Sadhita Pratyaya") and **IBEN** is a verb inflection ("Kriya-Bibhakti") indicating second/third person honoured, simple future tense. The word **KARaIBEN** is a verb meaning '(you or someone whom the speaker honours) will make do'. Thus, the meaning of the verb is derived from the meanings of the constituent morphemes.

The problem of parsing Bangla words have been taken up at the ISI some time ago. A solution based on an automata which is more powerful than a classical DFA and is organized in the form of a forest of Directed Acyclic Graphs (DAGs) have been reported in [8]. In the formalism described in the above paper, morphemes are clustered into belonging to different classes and morphemes from a cluster are stored in a DAG representing the cluster. However, all stem morphemes are stored in a single DAG called STEM. A major assumption that has been made in the above system is that every DAG is indeed acyclic.

Quite often, a stem morpheme may be considered to be constituted as a result of combining two simpler stems. Sometimes the stems that combine undergo "euphony" or 'Sandhi', i.e., the final portion of the left stem and the initial portion of the right stem undergo a deformation in spelling. The resulting stem will be called a "euphonized stem" in the present article.

The present work is an extension of the work described in [8] to incorporate 'Sandhi' rules in word level parsing. Simply speaking, the present work is involved with rule-based de-euphonization or 'Sandhi-Bicched' of stems of words. It may be pointed out that any pair of simple stems may not combine by euphony due to various restrictions. Since the above restrictions are almost

always from semantic considerations, they are very difficult to capture in a computerized system. In the present work, we assume that no restrictions are there for stems to be euphonized except spelling restrictions as described later.

2 STATEMENT OF THE PROBLEM

2.1 Definitions

Before we make a formal statement of the problem, certain terms need to be defined.

STEM: Stems are root form of simple words carrying meaning.
Example: SIMHA and aSANA are stems.

AFFIXES: Affixes in isolation do not carry any meaning but provide proper linguistic perspective for words for which they are joined. Depending on the position of joining,an affix is prefix, an internal affix or a declension.

Words are formed as a result of several conjoinings:
a)between a stem and an affix or vice versa.
b)among two or more affixes.
c)between two stems,thus producing compound stems.
d)by some combination of a) to c).
Example: IBEN is is a verb inflection of stem morpheme KAR.

SANDHI or JOINING RULE: When two stems are joined, there is a resulting deformation at the boundary of joining. A set of rules, most of which have been inherited from Sanskrit by most Indian languages, govern the spelling deformities. These rules are called 'Sandhi Rules'. A 'Sandhi Rule' has the form $left + right \Rightarrow resultant$, where $left$, $right$ and $resultant$ are strings of symbols from the alphabet of the language. As a result of application of the rule $a + b \Rightarrow c$ during joining of the stems **Xa** and **bY**, the joined stem is **XcY**.

JOINING: The operation by which two stem morphemes combine with the application of a 'Sandhi Rule' governing spelling deformity during the combination, is defined as "joining". We shall use the notation $X + Y$ to denote joining of stems X and Y. The notation $X + Y = Z$ denotes that the result of joining X and Y is Z.

FRAGMENTATION: The operation for extracting the joining stems of a compound stem alongwith the joining rule which was applied during joining, is called fragmentation or "Sandhi Bicched". Fragmentation is the inverse process of joining. For example, fragmentation of

MAHaTMa will yield **MAHa** + **aTMa** as the joining operation and $a + a \Rightarrow a$ as the joining rule involved.

COMPOUND STEM: The stem resulting out of "joining" two simpler stems will be called a "compound stem". A compound stem derives its meaning from the meanings of the joining morphemes. For example, in **SIMHA** + **aSANA** ⇒ **SIMHaSANA**, the compound stem **SIMHaSANA** derives its sense from both **SIMHA** and **aSANA**.

COMPLEX STEM: Sometimes, these are stems which are formed by mere concatenation of two or more simpler stems without application of any joining rule. For example, **MAHaPRABHU** is a complex stem formed as a simple concatenation of **MAHa** and **PRABHU**. Complex stems have not been handled in the present work.

CRACK POINT: This is the point where the compound stem breaks into constituent morphemes. It is point at which the string ahead is a resultant string of a *Sandhi Rule*.
EXAMPLE:For the compound stem MAHaTMA the crack point is 4. RaJENDRaLAY has two valid crack points at 4 and 8 respectively.

DETERMINISTIC FINITE AUTOMATA (DFAs) is defined as a five tuple $(Q, q_0, F, \Sigma, \delta)$, where,
Q=Set of states in the automation,
q_0=Starting state of automation. In the case of a DAG q_0 is the root.
F=Set of final states. This set refers to nodes which are arrived at after consuming last alphabet of a morpheme.
Σ=Set of characters constituting the alphabet. Small letters and capital letters are members of this set. The cardinality of this set is 52.
δ= Transition function which maps on to a state $\in$ Q from a state $\in$ Q after consumption of an alphabet $\in \Sigma$.

Given below are some examples of Sandhi:

1. HITA+UPADESH=HITOPADESH originating from the rule A+U⇒O.
2. JANA+EKA=JANeKA derived from the rule A+E⇒e.
3. MAHa+OSHADHI=MAHoSHADHI obtained from the rule a+O⇒o.
4. similarly, PO+AK=PaBAK originating from the rule O+A⇒aBA.

5. NOu+iK=NAABIK derived from Ou+i⇒AABI.

For a language, in general there could be multiple fragments of a compound stem. For example, RaJENDRaLAY is a compound stem originating from RaJa+INDRA+aLAY by virtue of rules a+I⇒E and A+a⇒a .

The possibility of null string participating in euphonization is not ruled out. For example if X+' '⇒Y then fragmentation of PYQ is PX and Q.

Recursive de-euphonization involves subsequent fragmentation of the compositions of a rule. Thus, if ABC+DEF⇒=GHIJ and X+Y⇒=E are Sandhi rules, the de-euphonization of PGHIJQ is PABC+DX+YFQ, owing to the above two rules, provided PABC, DX, YFQ are morphemes. Recursive de-euphonization is not seen in Bengali.

The present dissertation work is involved with fragmentation or 'Sandhi-Bicched' of Bangla Stems in a DAG Based Morphological Sub-system as described in [8].

2.2 The Problem:

In a word parsing environment, if a string XcY of symbols otherwise satisfy the properties of being a compound stem and if there exist a 'Sandhi Rule' a + b ⇒ c, i.e. there exists a crack point just after the string X is scanned. Then string XcY will be recognized as a stem which is formed as a result of euphony between simpler stems Xa and bY.

2.3 Motivation :

The major motivation behind taking up of the problem is computational — that of further exploitation of redundancy in the word formation system of

Indian languages. There are a few other motivation.*First*, there is the linguistic motivation in trying to have a basic tool using which the semantics of compound word formation can be studied. *Secondly*, a base is created for study of the behavior of the original system when certain amount of cyclic transitions are introduced. As we shall show later, incorporation of 'Sandhi Rules' most definitely introduce cyclic transitions. With many cyclic transitions, there may be considerable degradation in performance of the original system. A balance has to be reached between throughput and redundancy exploitation. The present work can act as a starting point from which a study of optimal morphological representation for an inflectional language may be carried out.

Thus, the problem of fragmentation of stems of an Indian language is a real and interesting one to solve. The solution will not only permit compact storage of large lexicons but also act as a starting point of studies in certain important fields of linguistics and natural language processing.

3 APPROACH TO THE PROBLEM

The first criterion lies in storing the lexicon within limits of availability of resources. The system maintains a database of morphemes from which the DAG is created. The DAG of morphemes is stored in the main memory of the system. The amount of storage space required for storing the lexicon depends upon the data structuring schemes employed. The scheme should be such that, the search of a morpheme in the DAG during execution stage of the system is minimized.

The system operates in two stages.

1. CREATING THE DAG OF MORPHEMES: This stage creates the DAG from the morphemic database and stores it in the main memory for the next stage. The number of links emanating from a node decreases as we go towards the leaf level. Sparsity of the DAG is reflected after fourth level due to the fact that number of morphemes having first four characters identical is negligible. For instance number of links emanating from a node at second level is around one to four [7]. Hence, more than 95% of the links are null in a lexicon storage environment when 52 links emanate from each node. We can get rid of null links by constructing link whenever the requirement arises. We have taken care to refine the data structures by employing binary tree representation scheme for a DAG, discussed in detail in next section .

2. EXECUTION STAGE: This stage on receiving a compound stem as an input returns all possible morphemic fragments of the input. After building the lexicon in the execution stage, we start scanning the compound stem from the root of the DAG until we arrive at some *resultant string* of the *Sandhi Rule* fetched from the knowledge base. Once a possible crack point is encountered ahead of the present scan point, it is checked whether on scanning the *left string* of the *Sandhi Rule* along the DAG, a final state is arrived. If a final state is not arrived, scanning is continued from the point where scanning was stopped last, until a crack point is found. Suppose a final state is arrived; root is revisited and scanning for the *right string* of the *Sandhi Rule* is done. Scanning is once again continued from the same state for the substring after the *resultant string* of the compound stem.

3.1 LEXICAL ORGANIZATION :

Before execution the morphemes contained in the lexicon is to be organized in the form of DIRECTED ACYCLIC GRAPHS (DAGs), also called as FINITE STATE TRANSITION NETWORKS(FSTNs). At the time of building the lexicon, of our system WG builds up a DAG of morphemes starting from a common root and tagging a boolean variable indicating the end of the morpheme. Pointers from the morpheme DAG might backtrack once it encounters a crack point(point a which the word might split owing to the presence of a resultant of joining rule).

A DAG for recognizing a class of morphemes is the pair (N, E) where N is a set of nodes and E is a set of edges. An edge is a triple $(n1, n2, token)$, and represents a transition from node $n1$ to node $n2$ on consuming the token(character) from input string of characters. The token c is called the label of the edge. A path from node $n1$ to node $n2$ is a sequence of edges $e1.e2..ek$, such that $e1$ starts from $n1$ and ends in $n2$. The labels on the sequence traces a sequence of characters which may or may not be meaningful. A special node called ROOT represents the start node of the DAG. Some nodes of DAG are called terminal nodes . A path from the start node to the terminal node traces out a unique morpheme of the class to which the DAG belongs. Thus, every terminal node has a finite numbers of paths from the start node, each path tracing spelling variations of the same morpheme. A terminal node is thus a representation for the morpheme it recognizes. For example, it should be noted that the morphemes 'aSAN ' and 'aSANA' follow same path from root till the end of the word 'aSAN' whereby for 'aSAN', DAG store a boolean value 'true' and thereby emanating another edge consuming the token character 'A' and indicating the end of the morpheme true for the word 'aSANA'.

During the building stage, our system first builds up a DAG of morphemes. The starting character of each emanate from the ROOT of the DAG. As explained above, a terminal node of any DAG is a representation of a unique morpheme (or spelling variations of the same morpheme). If there is morpheme which is an extension of another morpheme existing in the DAG earlier (DEBA is an extension of DEB) then, an intra DAG transition (which we'll call passive edge extension of the shorter word) takes place tagging a boolean variable 'true' when that word is completely stored in the form of a DAG. The general structure of the edges is given below:

ACTIVE EDGES: These are the edges which backtrack from nodes to the root once a crack point is encountered .

PASSIVE EDGES: These are intra DAG transitions.

TERMINAL INFORMATION: Whether this node indicates the valid end of morpheme.

4 DESIGN AND IMPLEMENTATION OF ALGORITHM

An algorithm for creating a DAG of stem morphemes to store the lexicon within allowable space limits is employed. An efficient data structuring mechanism is incorporated to store the lexicon. Once the lexicon is stored in the desired fashion, we fragment any compound stem supplied as input if the morphemes exist in the DAG . The DAG thus created,can be stored in a secondary device for subsequent execution. The fragmentation aspect of the compound stem starts from a *Rule Base* from which all the rules are fetched, and used in the execution stage of system.

4.1 Design Of Data Structures:

The major point to be remembered during implementation is that the lexicon could very well consist of one million entities. The space aspect need to be tackled effectively by employing efficient data structuring schemes for storage of huge lexicons. DFAs are efficient organizers of lexicons.

A DETERMINISTIC FINITE AUTOMATA (DFA) can be implemented by linked lists, array of pointers or doubly linked lists etc, if a fix amount of nodes emanate from the parent node. But handling of variable number of nodes, as it is of of frequent occurrence in NLP systems need to be solved keeping the space and time complexity in point of view. The extent of compromise between the two is decided by sparsity of the DAG at different levels. As the number of links emanating from fourth level onwards drastically reduces, the data structure employing array of pointers are discarded owing to its enormous storage requirement. The amount of wasteful spaces in a tree structure representation is illustrated by the following lemma. The DAG representation of a DFA is given in Fig.1 (in Appendix), for a general string of alphabet till second level.

Assume that the alphabet set contains 'a..z','A..Z'. If we assign 52 pointers from each node to the next level nodes, the space wastage becomes very high as discussed in the following lemma.

LEMMA [4]: If T is a k-ary tree (i.e. a tree of degree k) with n nodes, each having fixed size as follows:

data			
child-1	child-2	.....	child-k

then $n \times (k-1) + 1$ of the $n \times k$ fields are nil, $n \geq 1$.

From the above lemma, for a 52-ary tree (assuming that the alphabet contains $'a'..'z'$, $'A'..'Z'$ characters at most) having $52 \times n$ links, the number of null-links for a n-node tree is approximately $51 \times n$, indicating a high degree of wastage.
In our method, the lexicon is stored in a tree. To reduce this enormous space wastage, binary tree representation is used. In such a tree, it is implicitly assumed that the order of children of a node is not important. Only parent-child relationship is preserved. To obtain binary tree representation, the relationship of leftmost child- next right sibling is preserved. Every node has at most one leftmost child and at most one next right sibling.

Strictly speaking, since the order of the children is not important, only the children of a node could be its leftmost child and any one of its siblings could be its next right sibling.

A typical example of representation scheme of two Bangla morphemes(aMI and AK) one using array of pointers with constant record size and the other using binary tree representation is given in Fig.2 and Fig.3 (see appendix) respectively.

4.2 The Algorithm

The algorithm basically operates in two stages:-

4.2.1 DAG Creation:

In this stage the system builds the DAG of morphemes which is subsequently used in the running stage. Fields of a node are *alphabet, end morpheme, next level edge, next edge.*

STEP 1: Initialize P to root.

STEP 2: For i=1 to length of morpheme do 3, 4, 5.

STEP 3: Search for morpheme[i] in current level. If found then set P to the pointer found in the same level.

STEP 4: Else create a new node and insert it in the same level. Set alphabet to *morpheme[i]* and *next level edge* to nil and *end morpheme* to false. Set address of the new node created to P. End if.

STEP 5: If i=length of the morpheme then set *end morpheme* to true else set *end morpheme to false.* Set P to head of the children list of P. End for.

4.2.2 Execution Stage:

In this stage, the compound stem is taken as an input and returns all possible fragments of the input. This is a recursive procedure with two parameters such as *scan* and *pointer. scan* is an indication of the point at which the word is to be fragmented. *Pointer* representing the backtracking pointer once a *crack point* is arrived at.*Crack point* is the scan point at which the resultant of the sandhi rule is found .

STEP 1: Firstly, we call the procedure fragment with *scan=0* and *pointer*=root.

STEP 2: While *scan* less than length of the input word do

STEP 3: If there is a crack point at the current scan point then for all such crack point do 4, 5, 6, 7 .

STEP 4: Check if on scanning the left portion of the root, we arrive at an accepting state starting from the current state. This is owing to the fact that a portion of the string scanned by currently active procedure concatenated with the left hand portion of the rule is a valid morpheme or not. If we get a final state then check if we arrive at some state in the DAG starting from the root and scanning the right hand portion of rule. This done to see if the right hand portion of rule is a valid prefix of some morpheme of the DAG. In the algorithm *pointer* and state of the DAG are synonymous.

STEP 5: If the second condition is satisfied then we call the recursive procedure with parameters such as *scan* + length of the resultant string and *pointer* we have after scanning the right hand portion of the rule.

STEP 6: scan = scan + 1 and make appropriate transitions. If none exists then exit. End do.

STEP 7: If the current state is an accepting state then output the fragmented string. End do.

STEP 8: Else output "The compound stem do not have a valid fragmentation or constituent stem morphemes do not exist in DAG or check for the correct spelling of the word".

4.3 Computational Complexity

The creation of the DAG of morphemes takes comparatively more computations than running stage of the system. Worst case time complexity in forming the DAG is linear in total length of the morphemes processed in DAG creation stage. This is because of the fact that once a new alphabet at a particular level is found, a new node is created. A new method for minimization of acyclic deterministic automata in linear time is discussed.

Generally, the number of fanouts at a particular level decreases as we go towards the leaf level. Suppose the number of fanouts at the first level from root is 15, then number of links required is 31(two links for each fanout and one for root to first level). Total number of bytes required for storing 15 characters is 15 bytes. Total bytes consumed is 139 bytes, assuming 4 bytes

per link. For incorporating the same scheme as array of pointers in a 52-ary tree we require an enormous 208 bytes.

The space consumed is optimal by employing the binary tree representation of tree data structure. We have taken care of the enormous proportion of the null-link leaves in a 52-ary tree. With this representation 51 morphemes of average length 6 has occupied 1323 bytes in comparison to 34251 bytes while implementing by array of pointers.

The worst case time complexity in parsing a compound stem is exponential in length of the compound stem. This is due to the non-determinisms at various stages of parsing which have to be taken care of in a serial approach by either depth first search or breadth first search. Backtracking during the search may be reduced by using lookahead symbols to predict active transitions.

5 DISCUSSION

5.1 Discussion on the System:

The formalism proposed here has been tried out for a medium sized lexicon in Bengali consisting of one hundred morphemes and fifty rule bases. The results of de-euphonizations obtained were very satisfactory. The formalisms can easily be extended to any Indian languages with subtle modifications. There were two implementation of the same work using array of pointers and trees as data structures.

- The rules governing the conjoining rules are semantic in nature. To highlight one aspect of the semantic problem let's de-euphonize a word SIMHaMI with an existing rule A+a=>a and lexical morphemes as SIMHA and aMI. The system will obviously parse it into SIMHA and aMI,which is not a valid de-euphonization. This is a semantic problem to be tackled as a future extension.

- Moreover,since the noun stems and prefixes reside in the same DAG(STEM), the active transition from an active node recognizing a prefix leads to a root of DAG itself, causing self loops. Too many self loops leads to a rapid degeneration of efficiency..

- 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.

5.2 Discussion on the Algorithm

Acyclic automata is an efficient data structure for lexicon representation. Access time is linear in word size and compression results are excellent, minimization being the best size saving operation. Minimization of n-state DFA $O(n \times logn)$ by Hofcroft is a refinement of $O(n^2)$ algorithm by Moore.

5.3 Minimization of States in a DFA

An efficient algorithm [2] which takes care of the equivalent states of DAG by replacing it with a single state, thus minimizing it in linear time.
Each state of a directed acyclic word graph(DAWG) is labeled with a string describing the reduced automata starting at this state. Working up in increasing labels (labels = longest distance to a terminal state) the algorithm spreads the labels,which are created from the labels of the following states. At each label a lexicographic sort is applied to the list of labels and all the states with identical labels are merged (such states are equivalent whence a minimization). The labeling is done once and only once on each state and the sorting is linear, so overall complexity is linear in the number of transition.

If A is an automation, then there exists a unique automation M minimal by a number of states, recognizing the same language i.e. L($\mathcal{A}$)=L($\mathcal{M}$). Two states of a given automation p and q are said to be equivalent *iff* the automation defined with p and q as initial states are equivalent. If for every word w states $p.w$ and $q.w$ are final then two states are equivalent. An automation with no pair of equivalent states are minimal.

With the above scheme working with huge automata of one million states and 550,000 words are compressed by minimization algorithm. For example:300,000 word dictionary in 5.2MB in text format was compacted to 0.3MB in automation format–0.06 compression ratio.

6 CONCLUSION

The present work can further be extended to Semantic levels by:

- Tagging which are the pairs of simple stems which can not conjoin at the boundary. e.g. SIMHaMI if allowed to parse then the recognizer will return us two constituent SIMHA and aMI which are not valid conjoinings. Hence, we shall have to generalize the impossible pair euphonizations and take care of them as an excluded set.

- envisaging on conveying the sense of a word,from its originating morphemes i.e 'SAMAAS' .

  SAMAAS of MAHATMA is 'one possessing MAHAN ATMA'.

  But DE-EUPHONIZATION of MAHATMA is MAHa+aTMA.

A parallel approach eliminates backtracking completely by using a concurrent state-space method. The parallel method reduces time complexity appreciably but may result in high space complexity. Experimental result suggest that parallel approach yields slightly better results than the serial method because word lengths are normally within 6 to 8 symbols on the average, so that the extra space required is nominal.

References

[1] AHO A.V.;ULLMAN J.D:*Principle Of Compiler Design.* Narosa Publishing House-1990 .

[2] DOMINIQUE REVUZ: *Minimization of deterministic acyclic automata in linear time.* Theoretical computer science.(Jan'92)(181-189)

[3] GAZDER G. AND MELISH C.: *Natural Language Processing in Prolog.* Addison Wesley : 1989.

[4] HOROWITZ AND SAHNI: *Fundamentals of Data Structure.* Galgotia Book Source:1984.

[5] KOSKENNIEMI K.1983. *Two Level Model for Morphological Analysis.* In IJCAI-83. (INTERNATIONAL JOUNT CONFERENCE ON ARTIFICIAL INTELLIGENCE) Karlsruhe, West Germany: 683–685.

[6] KAY M. and KAPLAN R.M;1981. *Phonological Rules And Finite State Transducers.* ACL/LSA (ASSOCIATION OF COMPUTATIONAL LINGUISTICS/ LINGUISTIC SOCIETY OF AMERICA) Paper. New York.

[7] SENGUPTA P.;CHAUDHURI B.B.1989. *A Morphological Verb and Case Analyzer for NLP of a major Indian language.* In National symposium for Natural Language Processing.Vishakapatnam,India, 1989.

[8] SENGUPTA P.;CHAUDHURI B.B.; *A morpho-syntactic analysis based lexical sub-system.* (Under revision).

A Set of Sandhi Rules Used

The following set of rules are tested with the system and is found to give satisfactory results.

- a+R⇒Ar
- A+i⇒E
- a+i⇒E
- A+U⇒O
- A+u⇒O
- a+U⇒O
- a+u⇒O
- A+R⇒Ar
- A+E⇒e
- a+E⇒e
- A+O⇒o
- a+O⇒o
- u+U⇒u
- U+u⇒u
- u+u⇒u
- U+U⇒u
- Ou+i⇒AABI
- o+A⇒b
- A+A⇒a
- A+a⇒a

- a+A⇒a
- a+a⇒a
- A+I⇒E
- a+I⇒E
- E+A⇒AYA
- e+A⇒aYA
- O+A⇒ABA
- o+A⇒aBA
- I+I⇒i
- O+E⇒ABE
- o+I⇒aBI
- o+U⇒aBU

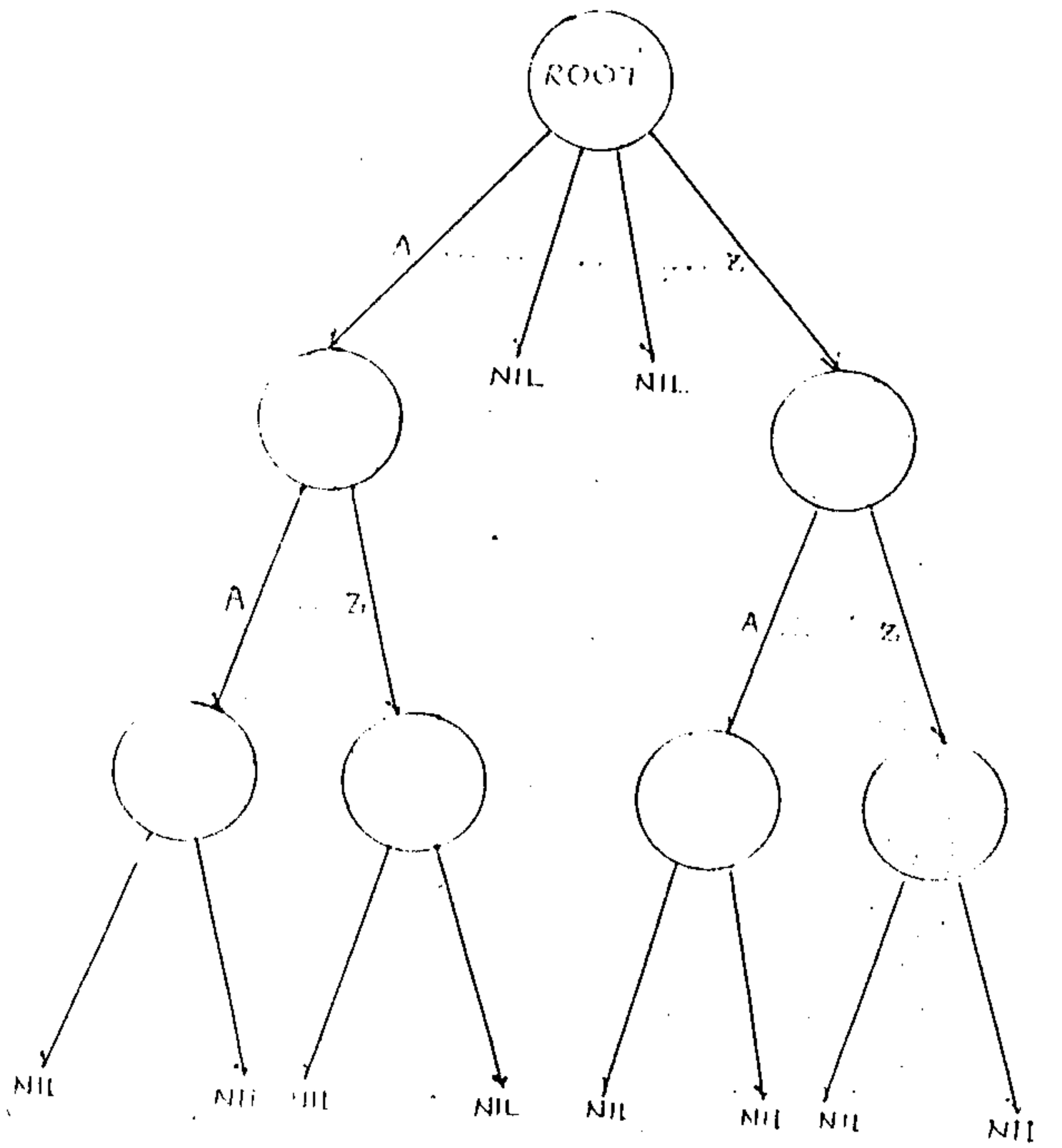


Fig.1. DAG representation for the alphabet set indicating that more than 95% links are null.

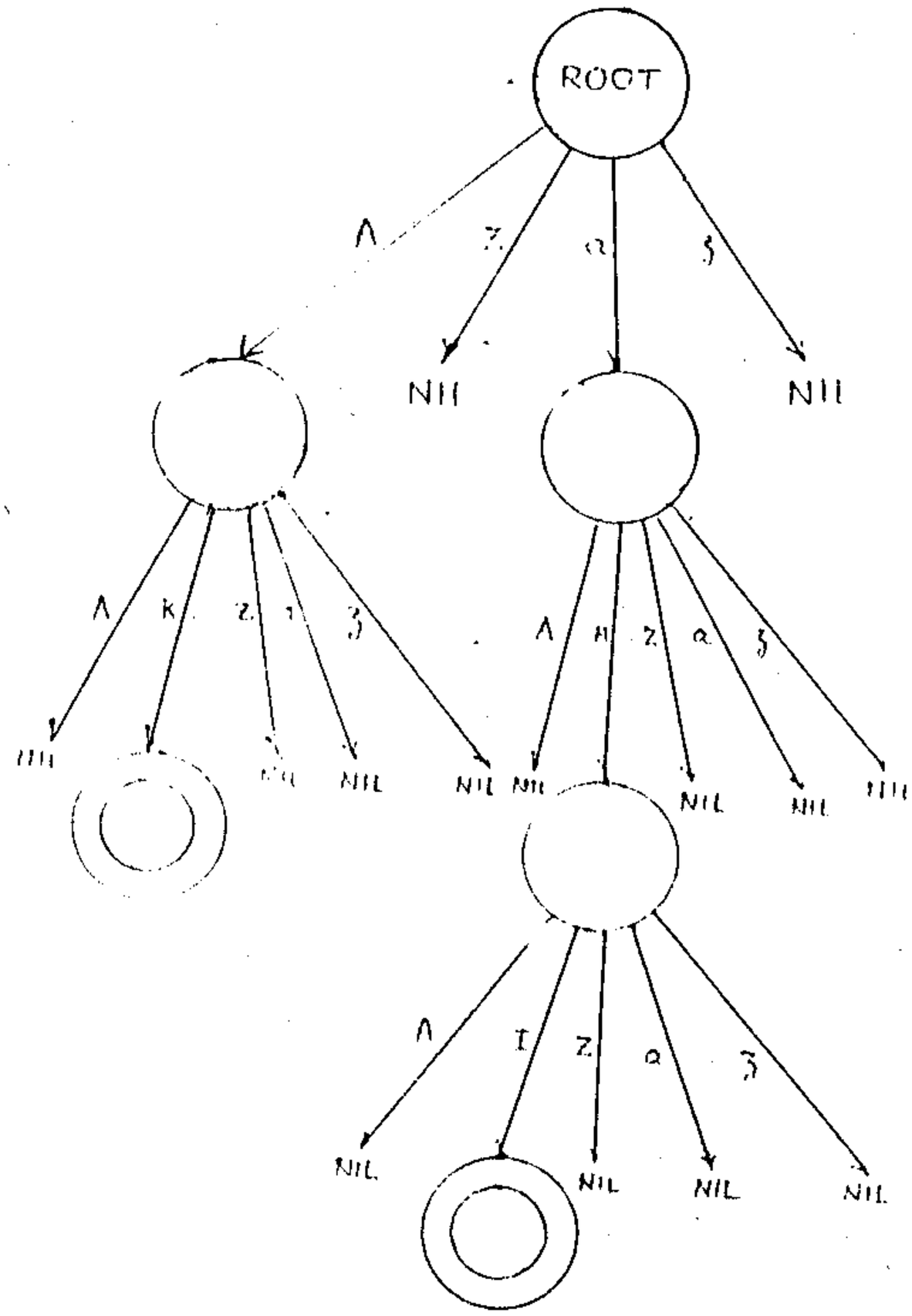


Fig.2. A typical representation of two morphemes AK and aMI as array of pointers in a 52-ary tree. End of a stem morpheme is represented by two concentric circles.

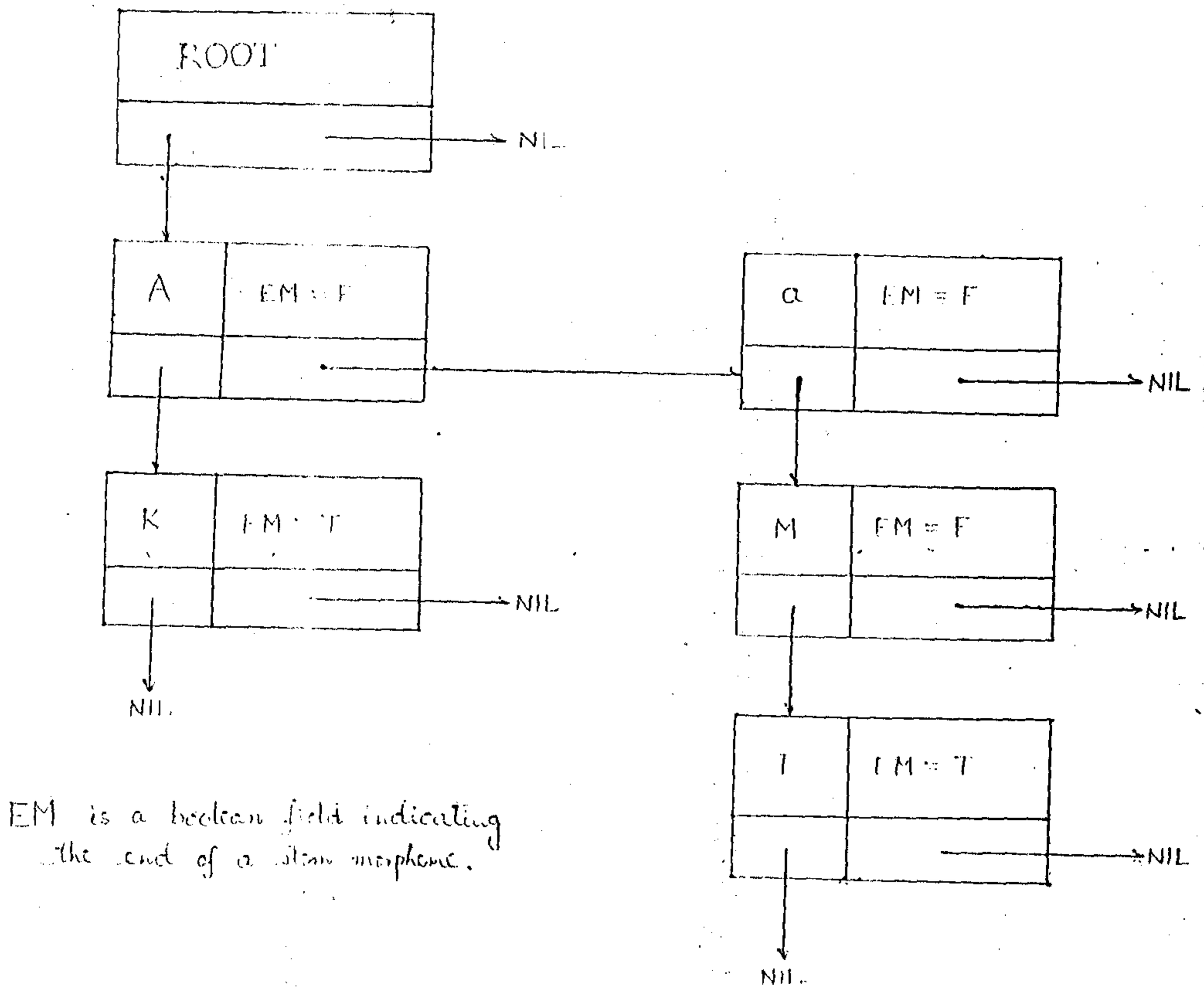


Fig.3. Binary tree representation of two morphemes, **aMI** and **AK** in the present scheme. Refinement is reflected in the number of non-nil links.