Functional Specifications of Morphology

Version:1.31

Date	: September 20,2008
Prepared by	: HCU
Reviewed by	: Expert Software Consultants

Revision History

Version	*Date of Revision*	*Section(s) Affected*	*Persons Involved*
1.1	**September 20,2008**	Case name and neg marker are added in optional values	*HCU*
1.2	**June 30, 2007**	Directory Structure	HCU
1.3	**January 18, 2008**	Output SSF With API	HCU
1.4	**June 06,2008**	S/W and Linguistic issues	HCU

Functional Specifications of Morphology

Prof. G. Uma Maheshwar Rao
Mrs. Amba P Kulkarni
M. Christopher
CALTS, Hyderabad Central University
guraohyd@yahoo.com , ambapradeep@gmail.com, efthachris@gmail.com

INDEX

1.INTRODUCTION:

MORPHOLOGY COMPILER/ ANALYSER is a program which compiles and analyses words belonging to a natural language. It works in a language independent way, hence it can learn and recognize words from any language. Morph has two modes of operation, viz the COMPILER mode and the ANALYSER mode. In the COMPILER mode it reads information about the words of a language from paradigm-input files and lexicon-data files and stores the processed information. This process is referred to as "compilation of data". Once this has been done Morph can recognize the words which were present in the data. All the data need not be compiled in one go, fresh data can be fed to morph any time by running it in compiler mode. To recognize words one has to run morph in its second mode of operation, which is the ANALYSER mode.

Morph recognizes only those words which it has been "taught". It outputs all the descriptions of the given word it has read in compiler mode. It produces a diagnostic "Unknown word <given word>" to say that it does not find the word in its list. An important point which should be well understood is the difference between "compilation of data"and compilation of the program "morph". Compilation of data is done by running morph in compiler mode. While "compiling morph" means putting the morph source code through the C-compiler. This may be necessary especially to customize morph program to some specific need. Compilation of morph is not connected with "data compilation" or "compilation of data" in any manner. At times changes may demand recompilation of full data, while for some changes only recompilation of morph may suffice, and for some changes both may be called for.

2. Input- output Specifications:

Input :	TKN
Output:	Morphological Anaysis

Input specifications require that property TKN_ must be defined in the input SSF that is given to the Morphological Analysis

ADD_	TKN_	CAT_
1	rAmA	<UNDEF>
2	sIwA	<UNDEF>

Output specifications required that property of attribute feature as given below will be defined by Morphological Analyzer. So the output SSF must contain Feature structures to all the valid values.
Input specifications require that property TKN_ must be defined in the input SSF that is given to the Morphological Analyser.

Output specifications required that property of attribute feature as given below will be defined by

Morphological Analyser. So the output SSF must contain Feature structures to all the valid values.

Output:

An output SSF from Morph must contain all the **four** columns of SSF.
The first column will have ADDR
The second column will have TKN
The third column will have CAT as `UNDEF`
The fourth column will have the `feature structure`, *`fs`*

The `feature structure` will be in the form of either `abbreviated features, af` and/or attribute-value pairs. If it has two `feature structures` then separate them with “`|`” character and between each column there is a tab that separates the fields.

The possible values of fs is listed below,

Nouns:

A noun is analysed as root+suff+{features(such as gender, number,...)}.
The complete structure is presented below.
<fs af
root = “ Root of the word”
lcat =”Lexical category of the root”
gend =“Gender of the word”
num =”Number coressponding to the word form”
pers =”Person of the word”
case =”Case (Direct / Oblique)”
vibh =”(cm / tam)”
case_name=”case name”
spec =”Specificity Marker”
emph =”Emphatic Marker”
dubi =”Dubitative Marker”
interj =”Interjection Marker”
conj =”Conjunction Marker”
hon =”Honorific Marker”
agr_gen =”Gender of the agreeing noun”
agr_num =”Number of the agreeing noun”
agr_per =”Person of the agreeing noun”
suff =”Form of suffix representing all the above markers”
>

VERBS:

The verb analysis structure is presented below.
<fs af
root =”Root of the word”
lcat =”Lexical category of the root”
tam =”Suffix indicating Tense Aspect Modality”
gend =”Gender of the word”

num =”Number corresponding to the word form”
pers =”Person of the word”
spec =”Specificity Marker”
emph =”Emphatic Marker”
dubi =”Dubitative Marker”
interj =”Interjection Marker”
conj =”Conjunction Marker”
hon =”Honorific Marker”
neg =”verb-neg Marker”
voice =”Voice”
caus =”Whether the verb form is causative or not(y/n)”
finiteness =”Whether the verb form is finite or not (y/n)”
suff =”Suff representing all the above markers”
>

Adjectives:
The feature structure for Adjectives is as follows:
<fs af
root =” Root”
lcat =”Lexical category
gend =”Gender of the word
num =”Number
pers =”Person of the word
degree =”degree
-like =”like
dubi =”Dubitative
interr =”Interrogative
emph =”Emphatic
conj =”Conjunction Marker
?spec =”Specific
suff =”complete suffix
>

Adverbs:
The feature structuer for Adverbs is presented below.
<fs af
root=”Root of the word”
lcat=”Lexical category of the root”
dubi=”Dubitative Marker”
interr=”Interrogative”
emph=”Emphatic Marker”
conj=”Conjunction Marker “
?spec=”Specific”
suff=”complete suffix”
>

Noun Locative:

A locative noun is analysed as root+suff+{features(such as gender, number,...)}.
The complete structure is presented below.
<fs af
root = “ Root of the word”
lcat =”Lexical category of the root”
gend =“Gender of the word”
num =”Number coressponding to the word form”
pers =”Person of the word”
case =”Case (Direct / Oblique)”
vibh =”(cm / tam)”
spec =”Specificity Marker”
emph =”Emphatic Marker”
dubi =”Dubitative Marker”
interj =”Interjection Marker”
conj =”Conjunction Marker”
case_name=”case name”
hon =”Honorific Marker”
agr_gen =”Gender of the agreeing noun”
agr_num =”Number of the agreeing noun”
agr_per =”Person of the agreeing noun”
suff =”Form of suffix representing all the above markers”
>

Avaya:

A Avaya is analysed as root+suff+{features(such as gender, number,...)}.
The complete structure is presented below.
<fs af
root = “ Root of the word”
lcat =”avy”
gend =“”
num =””
pers =””
case =””
vibh =””
suff =”Form of suffix representing all the above markers”
>

PostPosition:

A PostPosition is analysed as root+suff+{features(such as gender, number,...)}.
The complete structure is presented below.
<fs af
root = “ Root of the word”
lcat =”Lexical category of the root”
gend =“Gender of the word”

```
num =”Number coressponding to the word form”
pers =”Person of the word”
case =”Case ( Direct / Oblique )”
vibh =”(cm / tam)”
spec =”Specificity Marker”
emph =”Emphatic Marker”
dubi =”Dubitative Marker”
interj =”Interjection Marker”
conj =”Conjunction Marker”
hon =”Honorific Marker”
case_name=”name of case”
agr_gen =”Gender of the agreeing noun”
agr_num =”Number of the agreeing noun”
agr_per =”Person of the agreeing noun”
suff =”Form of suffix representing all the above markers”
>
```

Number :

A Numeral is analysed as root+suff+{features(such as gender, number,...)}.
The complete structure is presented below.

```
<fs af
root = “numer”
lcat =”num”
gend =“”
num =””
pers =””
case =””
vibh =””
suff =””
>
```

Punctuation :

A Punctuation is not analysed just give the NCR.
The complete structure is presented below.

```
<fs af
root = “NCR”
lcat =”punc”
gend =“”
num =””
pers =””
case =””
vibh =””
suff =””
>
```

Unknown :

A Unknown is not analysed and it is just repeatd.

The complete structure is presented below.

```
<fs af
root = “word”
lcat =”unk”
gend =“”
num =””
pers =””
case =””
vibh =””
suff =””
>
```

The structure of feature structure of Morphological Anlyser is given below.

'fs' is feature structure which contains 'af' is a composite attributes consisting of root, lcat, gend, num, pers, case, vibh, tam .

Some frequently occurring attributes (such as root, lcat, gend, etc.) may be abbreviated using a special attribute called 'af' or abbreviated attributes.

These eight cases are mandatory for the morph output :

<fs af = 'root,lcat,gend,num,pers,case,vibh,suff' >

The rest are optional:

<fs af = 'root,lcat,gend,num,pers,case,vibh,suff' spec=”” emph=”” dubi=”” case_name”” interj=”” conj=”” hon=”” arg_gen=”” arg_num=”” arg_per=”” voice=”” caus=”” finiteness=”” ?spec=”” >

Some of the possible values for the attributes list is given below:

I) . **lexical category (lcat)** will have all possible values like:

1) noun	= n
2) verb	= v
3) adjective	= adj
4) adverb	= adv
5) pronoun	= pn
6) Nlocative	= nst
7) avya	= avy
8) postposition	= psp
9) number	= num
10) punctation	=punc
11) unknow	=unkn

II). The possibles values for **Gender** :

m, f, n , mf , mn , fn, any .

III). The possibles values for **Number** :

sg, pl, dual, any.

IV). The possibles values for **Preson** :

1, 2, 3, any.

V) The possibles values for **Case** :

d,o

VI) The possibel **case marker** :

ex **(Telugu)** : dir, obl, ki , ku, ni , nu, lo, wo, yoVkka etc ...

VII). The possibles values for **Vibhakt**i :

(Ex Telugu) du,mu,vu,lu,ce,wan,ceVn....

vibh has the value either “cm /tam” depending on the lcat value it takes the **cm** or **tam**.

vibh = case marker (cm) in case of noun.

vibh = tam (Tense Aspect Modularity) in case of verb.

VIII). The possible feature structre for the word which is **unknown** to morph is **<fs af='word,unkn,,,,,,'>**

IX). The possible feature structre for the punctation mark which is unknown to morph is

<fs af='&sdot;,punc,,,,,,'> for \. .

X). The possible feature structure fot the number is **<fs af='88,num,,,,,,'>**

XI). The possible values for **case name :**

ex: nom, acc, dubi, etc **or** 1, 2, 3

3 . Flow Chart of Morph:

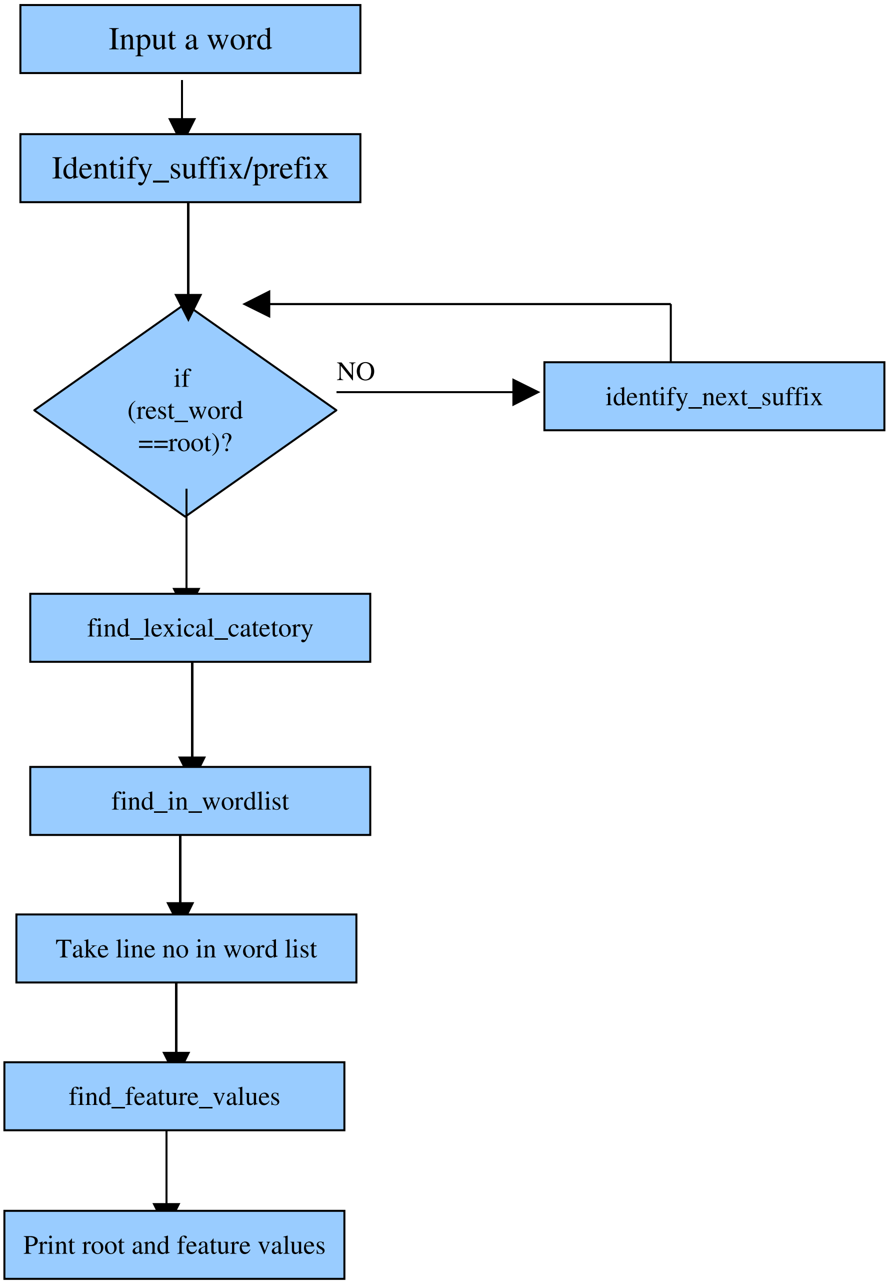


Fig: 1: Flow chart for the Morphology

Flow Chart of Morph will show the overview of the program control from one module to the other. It shows the decision points in the program.

4. Process Descriptions:

Morphological Analyser:-

In Morphological analysis we take words and we try to identify the suffix or prefix, and check weather this suffix is a valid suffix. If this suffix is present in the word-list then it is a vialed if not its a unknown word for Morphological Analyzer. If the suffix is valid then check weather the stem is valid or not just, by converting it to root word, by adding and deleting is done. If the suffix and root words are valid then take the line number of the word in word-list then get the feature structure (like gnp, tam, case, case marker value). Add root word and feature structure to API- wrapper to print in the data tree.

API and their Implementation:-

API's (Application Programming Interface) are used to avoid usage of excess memory by calling input and output in one file using Data Structures. And this is used to get the structure of SSF with out any wrappers.
Advantages:

1) It is easy to locate the words in the text by traversing the tree. Reduces Storage and retrieving efforts .
2) We can retrieve a single feature value with out any change in the code of Morph Engine SSF representation designed to keep both rule based as well as statistical approaches together.We use APIs in reading input in SSF format and executing output in SSF format.
3) The Unrecognized by morph are stored in uword file and they have also been displayed in the morph output.

5. Data Flow Diagram:

1)

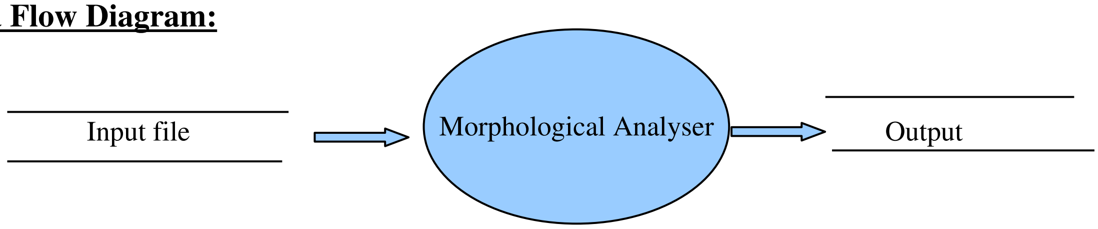


Fig-5 : Level 0 DFD

2)

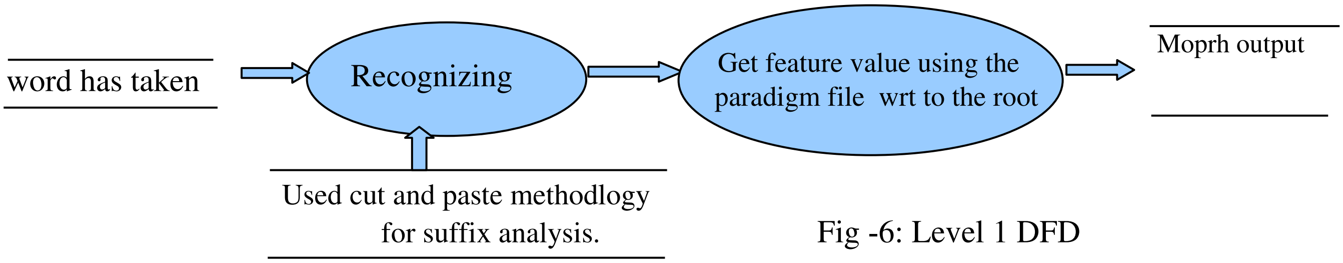

word has taken
Recognizing
Get feature value using the paradigm file wrt to the root
Moprh output
Used cut and paste methodlogy for suffix analysis.

Fig -6: Level 1 DFD

The above DFDs explains about data flow of a word.
Data Flow Diagram in another way:

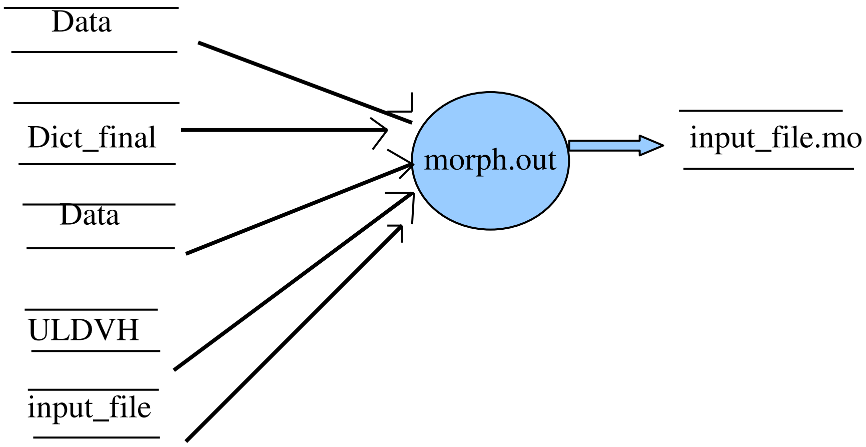


Fig-7: LEVEL-0 DFD

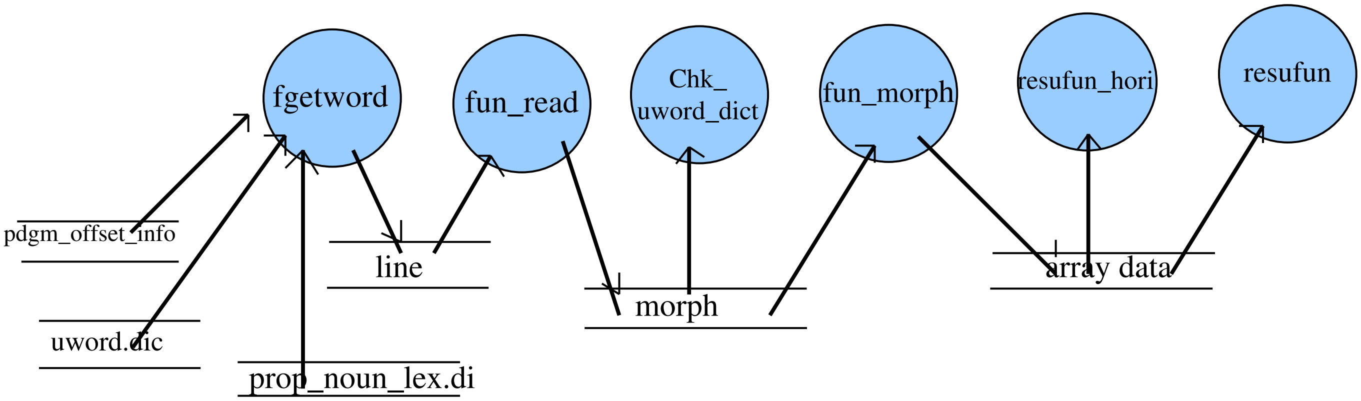


Fig-8: LEVEL-1 DFD

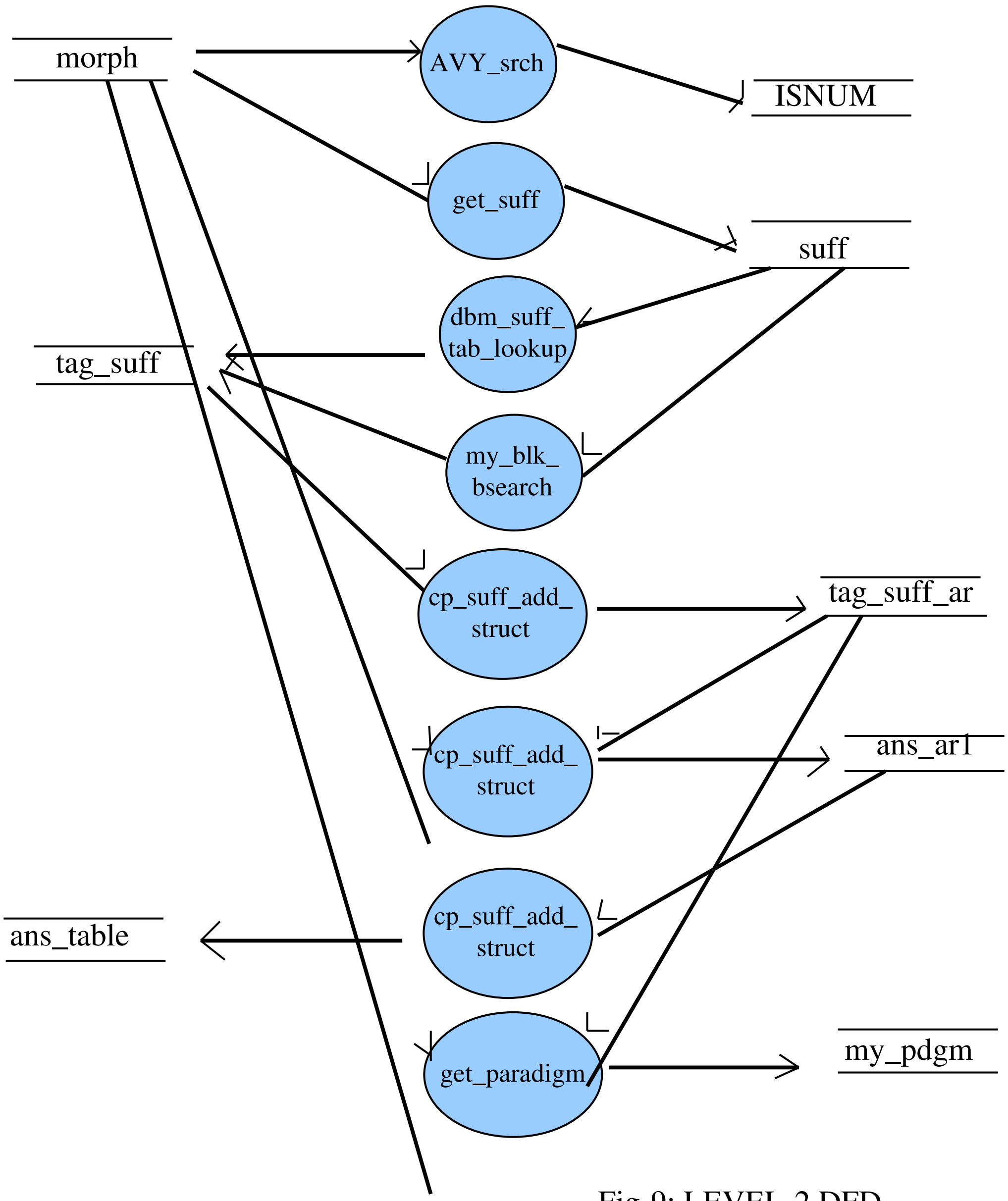


Fig-9: LEVEL-2 DFD

Here the above DFDs are represented as per the programmning concept.How data flows was shown from LEVEL 0 to 3

APPENDEX

INPUT-OUTPUT Specifications:

(Telugu Example)

ADDR_	TKN_	CAT_
1	ceVppAdu	<UNDEF>
2	navvadAniki	<UNDEF>
3	rAmudivaMka	<UNDEF>
4	Avida	<UNDEF>
5	kAni	<UNDEF>
6	aMxamEnavAdu	<UNDEF>
7	pEna	<UNDEF>
8	viSiRtamEnavAdu	<UNDEF>
9	aMxamEnavi	<UNDEF>

Output :

(Telugu Example)

```
ADDR_   TKN_        CAT_         others_
1        ceVppAdu    <UNDEF>     <fs af='ceVppu,v,m,s,3,,A,A'>
2        navvadAniki  <UNDEF>    <fs af='navvu_adaM,n,any,3,,,ki,ki'>
3       rAmudivaMka <UNDEF>     <fs af='rAma,adj,m,3,,,vEpu,vEpu'>
4       Avida           <UNDEF>     <fs af='Avida,pn,,,,d,,'>|<fs af='Avida,pn,,,,o,,'>
5       kAni            <UNDEF>     <fs af='kAni,avy,,,,,,'>|<fs af='kaM,n,any,3,,o,,'>|<fs
af='avvu,v,any,any,any,,ani,ani'>
6       aMxamEnavAdu <UNDEF> <fs af='aMxaM,pn,m,sg,2,,Ena_vAdu,Ena_vAdu'>
7       pEna            <UNDEF>      <fs af='pEna,n,any,,3,o,,'>|<fs af='pEna,n,any,,3,o,,'>|<fs
af='pEna,n,,,,o,,'>|<fs af='pEna,adv,,,,,yoVkka,yoVkka'>|<fs af='pE,n,any,,3,,na,na'>
8       viSiRtamEnavAdu <UNDEF>          <fs af='viSiRta,adj,,,,,Ena_vAdu,Ena_vAdu' >
9       aMxamEnavi  <UNDEF>      <fs af='aMxaM,pn,fn,pl,2,,Ena_xi,Ena_xi'>
```

INPUT:

(Hindi Example)

ADDR_	TKN_	CAT_
1	bahuwa	<UNDEF>
2	acCI	<UNDEF>
3	aMgrejI	<UNDEF>
4	jAnawe	<UNDEF>
5	We	<UNDEF>
6	Ora	<UNDEF>
7	mAnawe	<UNDEF>
8	We	<UNDEF>
9	ki	<UNDEF>
10	baDiyA	<UNDEF>

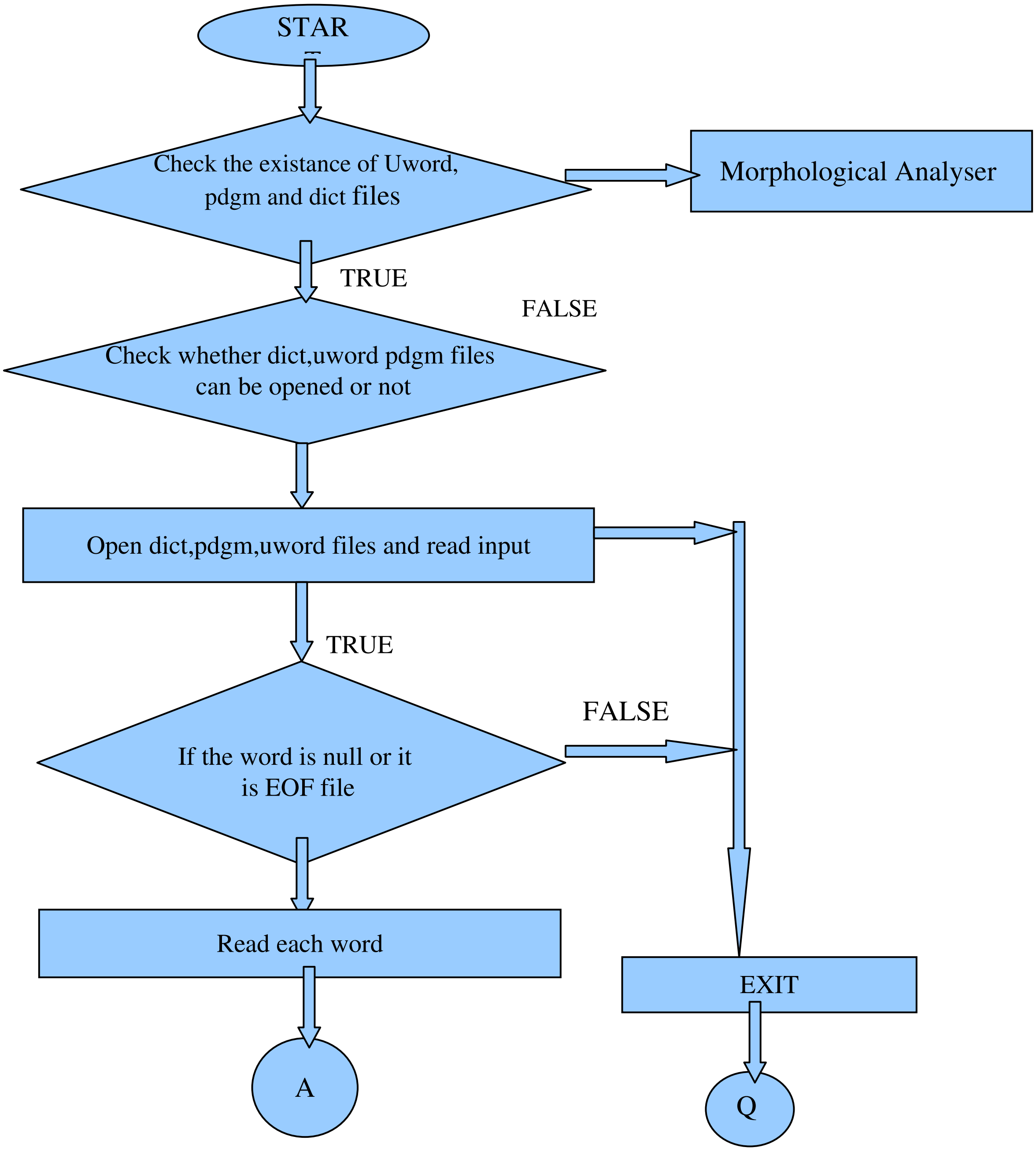
Output:

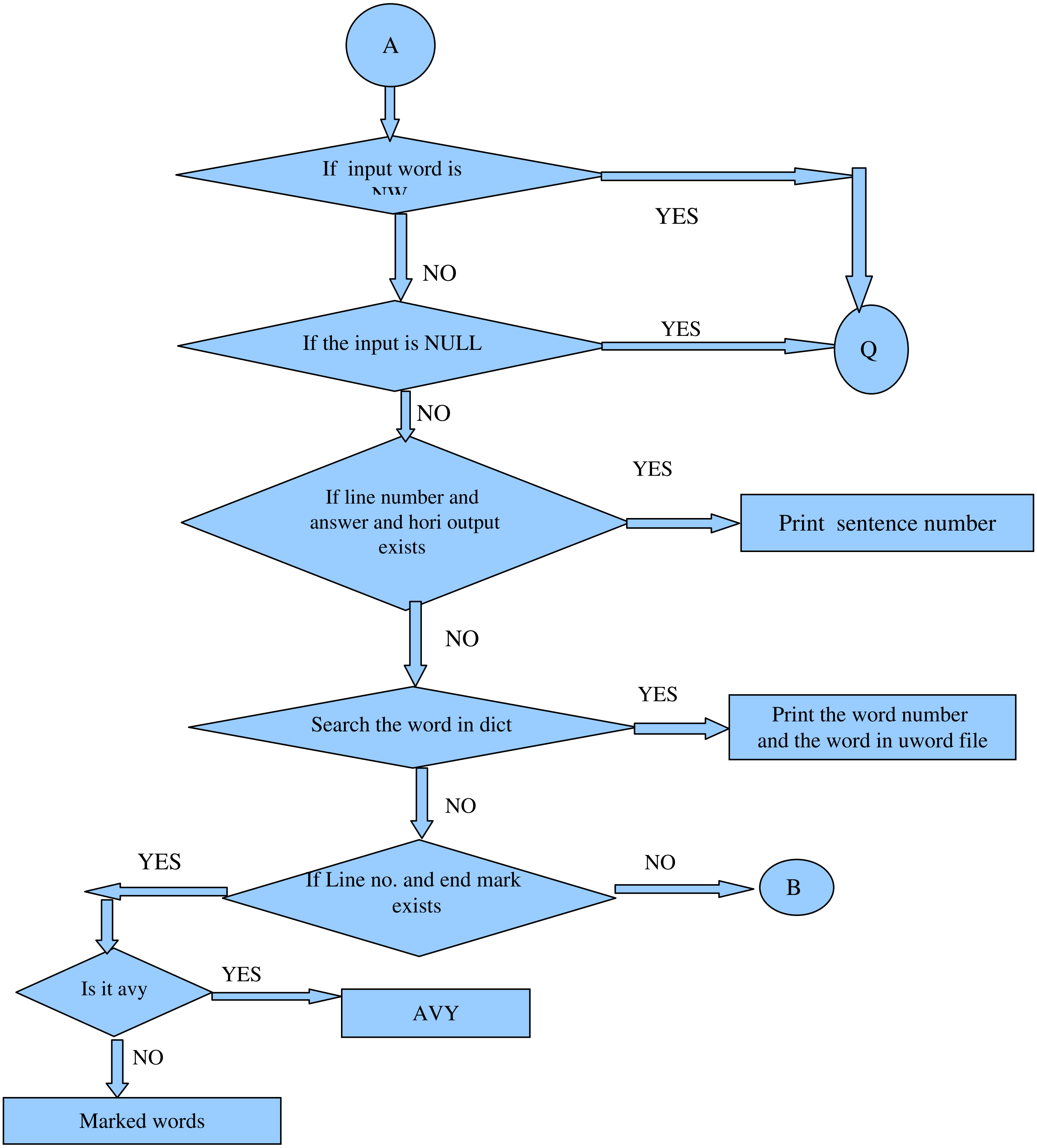
```
ADDR_   TKN_      CAT_               others_
1       bahuwa        <UNDEF>     <fs af='bahuwa,n,m,sg,,d,,'>|<fs af='bahuwa,n,m,pl,,d,,'>|<fs
af='bahuwa,n,m,sg,,o,,'>|<fs af='bahuwa,adv,,,,,,'>|<fs af='bahuwa,adj,any,any,any,,,'>
2       acCI    <UNDEF>     <fs af='acCI,adj,f,any,any,,,'>
3       aMgrejI       <UNDEF>     <fs af='aMgrejI,n,f,sg,,d,,'>|<fs af='aMgrejI,n,f,pl,,d,,'>|<fs
af='aMgrejI,n,f,s,,o,,'>|<fs af='aMgrejI,n,f,pl,,o,,'>
4       jAnawe        <UNDEF>     <fs af='jAna,v,m,pl,any,,wA,wA'>
5       We      <UNDEF>     <fs af='WA,v,m,pl,any,,WA,WA'>
6       Ora     <UNDEF>     <fs af='Ora,adv,,,,,,'>|<fs af='Ora,Avy,,,,,,'>
7       mAnawe        <UNDEF>     <fs af='mAna,v,m,pl,any,,wA,wA'>
8       We      <UNDEF>     <fs af='WA,v,m,pl,any,,WA,WA'>
9       ki      <UNDEF>     <fs af='ki,avy,,,,,,'>
10      baDiyA        <UNDEF>     <fs af='baDiyA,unk,,,,,,'>
```

II) The possibles values for **Vibhakt**i :

(Ex Telugu) du,mu,vu,lu,ce,wan,ceVn....

WORD FLOW IN MORPH ENGINE


STAR
T
Check the existance of Uword, pdgm and dict files
Morphological Analyser
TRUE
FALSE
Check whether dict,uword pdgm files can be opened or not
Open dict,pdgm,uword files and read input
TRUE
FALSE
If the word is null or it is EOF file
Read each word
EXIT
A
Q


A
If input word is NW
YES
NO
If the input is NULL
YES
Q
NO
If line number and answer and hori output exists
YES
Print sentence number
NO
Search the word in dict
YES
Print the word number and the word in uword file
NO
If Line no. and end mark exists
YES
NO
B
Is it avy
YES
AVY
NO
Marked words

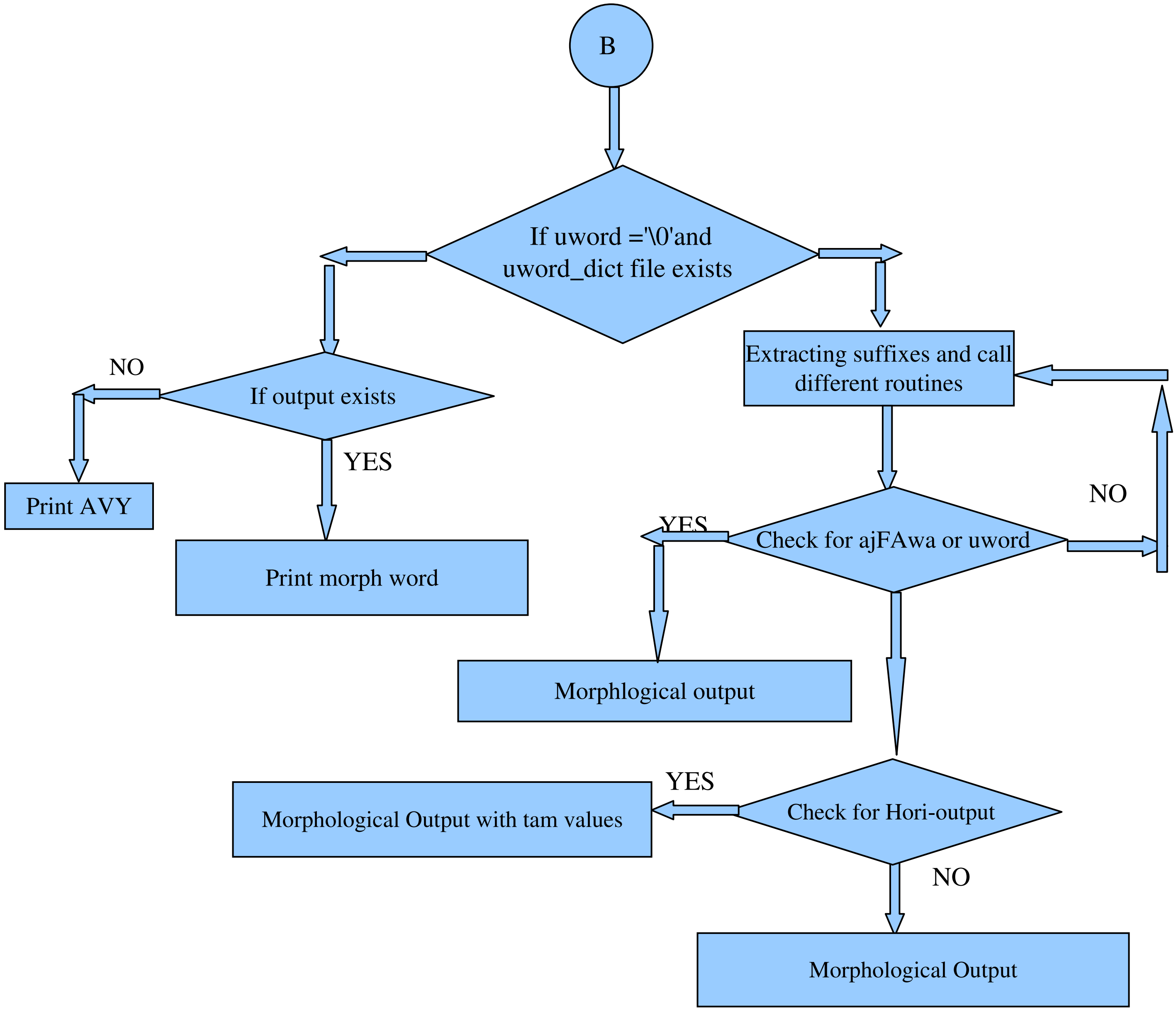
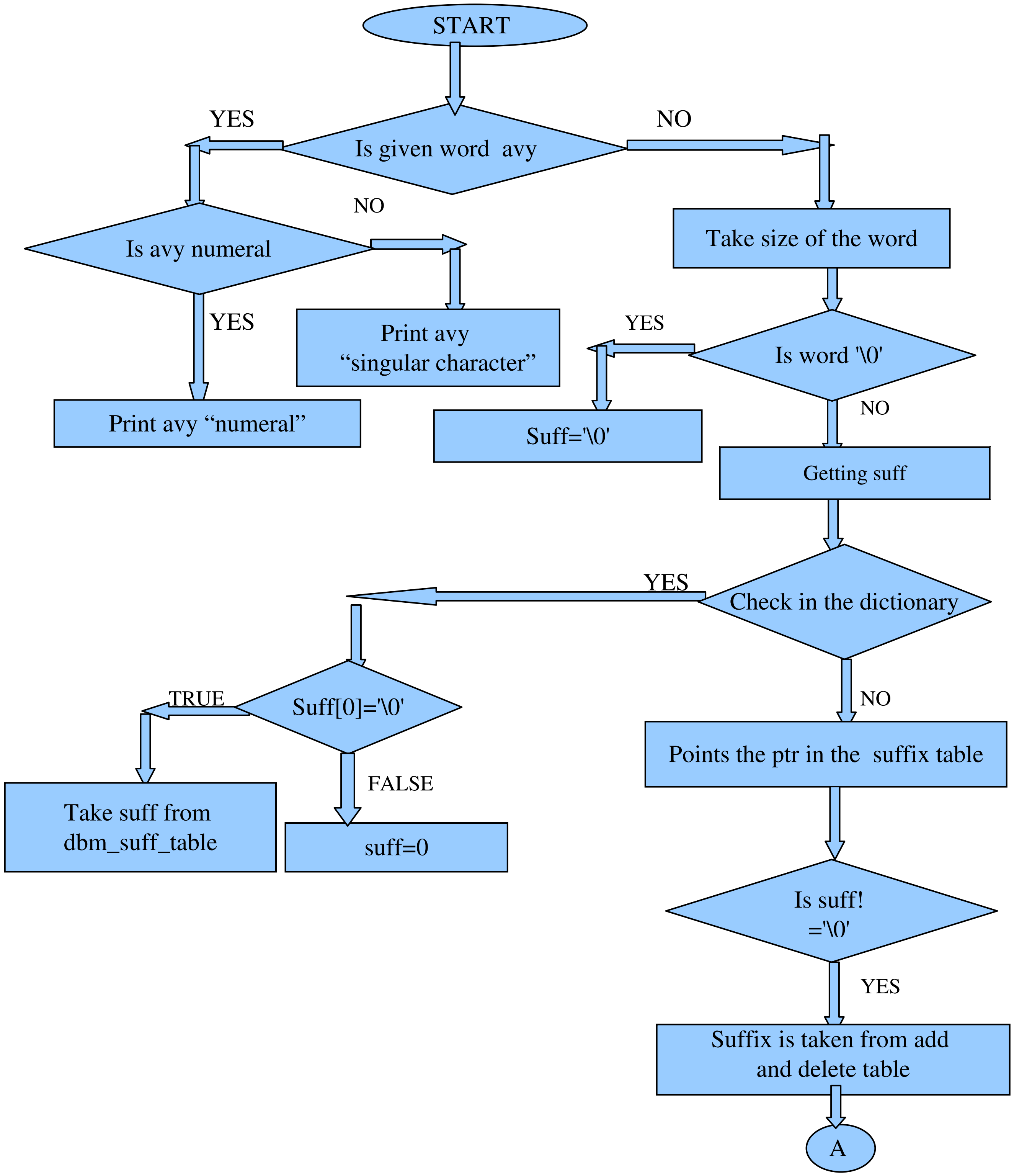

B
If uword ='\0'and uword_dict file exists
If output exists
NO
Print AVY
YES
Print morph word
Extracting suffixes and call different routines
Check for ajFAwa or uword
YES
NO
Morphlogical output
Check for Hori-output
YES
Morphological Output with tam values
NO
Morphological Output

Fig-3: Flow Chart for the word flow of morph

FLOW CHART FOR TELUGU MORPH



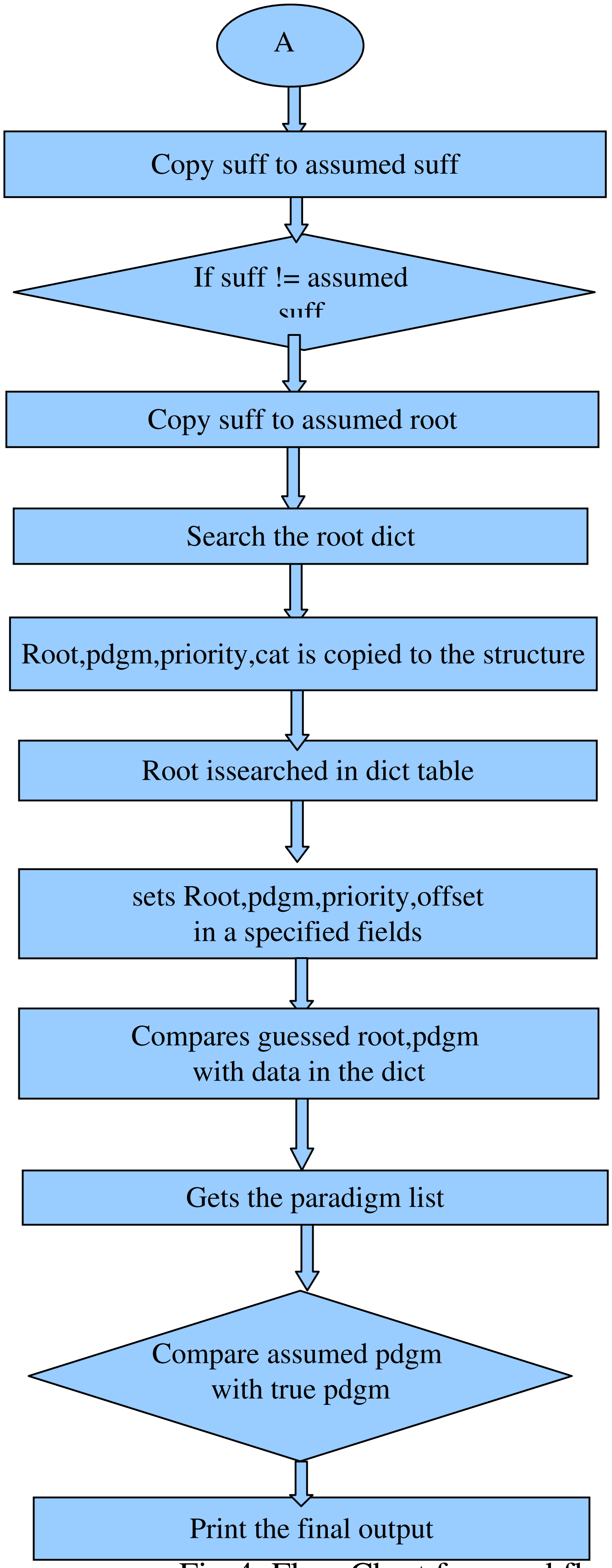

A
Copy suff to assumed suff
If suff != assumed suff
Copy suff to assumed root
Search the root dict
Root,pdgm,priority,cat is copied to the structure
Root issearched in dict table
sets Root,pdgm,priority,offset in a specified fields
Compares guessed root,pdgm with data in the dict
Gets the paradigm list
Compare assumed pdgm with true pdgm
Print the final output

Fig-4: Flow Chart for word flow

NOTES

Note 1:

- The code can be separated to into parts (subroutines) like – command line parsing, program intialization, error handling and logging, and the main application.
- All the function/subroutines have been defined in the defn.h, struct.h struct1.h files. It would be better if,
  - Functions/subroutines are defined in the separate program files, .cpp file.
  - In the main program they must be called as subroutines.
- There is lot of complex/obfuscated code which could be restructured in to small functions.
- The programs should be designed so that modules can be called in a pipeline one after another, like Unix programs.
- This kind of program interface design will also help when we plan to run them via a dashboard.

Note 2:

- Large number of global variables are defined (not declared) in defn.h file. These variables are being accessed by functions as and when required.
- The DFDs (derived from the code) does only give a vague picture of how the data flows inside Morphological Analyzer.
- Restructuring the program codes in line with basic software engineerig principles will increase the readability and maintainability of the software.

NOTE 3:
Linguistic descripiton of feature structure of different lexical categories.

Nouns:
A noun is analysed as root+suff+{features(such as gender, number,...)}.
The complete structure is presented below.
<fs af
root = Root of the word
lcat = Lexical category of the root
gend = Gender of the word
num = Number coressponding to the word form
pers = Person of the word
case = Case (Direct / Oblique)
vibh = (cm / tam)
spec = Specificity Marker
emph = Emphatic Marker
dubi = Dubitative Marker
interj = Interjection Marker
conj = Conjunction Marker
hon = Honorific Marker
case_name=name of case (nom, dubi, gen)
agr_gen = Gender of the agreeing noun
agr_num = Number of the agreeing noun
agr_per = Person of the agreeing noun
suff = Form of suffix representing all the above markers
>

Example:
Input word: manuRulake
output:
manuRulake < fs af='root=maniRi,lcat=n,gend=any,num=pl,pers=3,case=obl, vibh=ki,suff=ki_e'
spec=,emph=e,dubi =,interj=,conj=,hon=,agr_gen=,agr_num=,agr_per=>

VERBS:

The verb analysis structure is presented below.

<fs af
root = Root of the word
lcat = Lexical category of the root
tam = Suffix indicating Tense Aspect Modality
gend = Gender of the word
num = Number coressponding to the word form
pers = Person of the word
spec = Specificity Marker
emph = Emphatic Marker
dubi = Dubitative Marker
interj = Interjection Marker
conj = Conjunction Marker
hon = Honorific Marker
neg=negative marker of verb
voice = Voice
caus = Whether the verb form is causative or not(y/n)
finiteness = Whether the verb form is finite or not (y/n)
suff = Suff representing all the above markers
>

Example:
Input word : pagalagoVttAnu
output:
pagalagoVttAnu < fs af='root=pagalagoVttu,lcat=v,gend=any,num=s,per=1,case=a,tam=a,suff=A'spec=,emph=,dubi=,interj=,conj=,hon =,voice=,finiteness=y>

Adjectives:

The feature structure for Adjectives is as follows:

<fs af
root = Root
lcat = Lexical category
gend = Gender of the word
num = Number
pers = Person of the word
degree = degree
-like = like
dubi = Dubitative
interr = Interrogative
emph = Emphatic
conj = Conjunction Marker
case_name=name of case (nom, dubi, gen)
?spec = Specific

```
suff = complete suffix
>
```

Example:
Input word : lewaxi
Output:

```
lewaxi          <fs af= 'root=lewa,lcat=adj,gend=nm,num=s,pers=3,case=,vibh=,suff=xi' -like= “”
degree=”” dubi=”” interr=”” emph=”” conj=””>|<fs af= 'root=
xi,lcat=p,gend=nm,num=,pers=3,case=3,vibh=,suff='spe=,emph=,inter=,conju=,nm=,agr_gen=,agr_num=,a
gr_pers=>
```

Adverbs:
The feature structuer for Adverbs is presented below.

```
<fs af
root=Root of the word
lcat=Lexical category of the root
dubi=Dubitative Marker
interr=Interrogative
emph=Emphatic Marker
conj=Conjunction Marker
?spec=Specific
suff=complete suffix
>
```

Example:
Input word : bayataxi
output:

```
bayatixi        <fs
af='root=bayati_xi,lcat=adv,gend=any,num=s,,pers=3,case=1,vibh=0,suff=nu'like=--,spec=--,emph=,dubi=,
interj=,conj=,nm=,agr_gen=any,agr_num=,agr_pers =>
```

In the following **Telugu example** we gave all possible categories.

INPUT:

```
1        ceVppAdu
2        navvadAniki
3        rAmudivaMka
4        Avida
5        kAni
6        aMxamEnavAdu
7        pEna
8        viSiRtamEnavAdu
9        aMxamEnavi
```

OUTPUT: Output is of two forms in SSF format.

1) Output in SSF format describing attributes in detail.

```
                    1      ceVppAdu              < fs af ='root=ceVppu,lcat=v,gend=m,num=sg,pers=3, tam=A'>
2      navvadAniki           < fs  af='root=navvu_adaM,lcat=n,gend=any,num= any,pers=3,case = ki'>
3      rAmudivaMka          < fs  af='root=rAma_adj_m,lcat=n,gend=any,pers=3,case=vEpu'>
4      Avida                       < fs  af='root=Avida,lcat=pn,case=0'>|< fs af='root=3_Avida,lcat=pn,case=obl'
>
5      kAni                         < fs  root = kAni  lcat = Avy >|< fs  af='root=kAni,lcat=Avy'>|< fs
af='root=kAni,  lcat=Avy'>
6      aMxamEnavAdu       < fs  af='root=aMxaM_Ena_vAdu,lcat=pn,case=0' >
7       pEna                        <fs af='root=pEna,lcat=n,gend=any,pers=3,case=0'>|<fs
af='root=pEna,lcat=n,gend=any,pers=3,case=obl'>|<fs af='root=pEna,lcat=n,case=adv_0'>|<fs
af='root=pEna,lcat=n,case=adv_yoVkka'>|<fs af='root=pE ,lcat=n,gend=any,pers=3,case =na'>
8        viSiRtamEnavAdu     <fs af='root =viSiRta_adj_n_Ena_vAdu,lcat=pn,case=0' >
9        aMxamEnavi             <fs af='root=aMxaM_Ena_xi,lcat=pn,case=0'>
```

NOTE 4:

PREPARING DATA FOR MORPH

This chapter contains the definitions of terms used in this document to describe morph usage. Syntax of the files which morph takes from the user is also explained in this chapter. A proper grasp of these is essential for any morph user.

FEATURE : Any lexical-attribute which characterizes a word, or describes a word's meaning, is referred to as a "feature", e.g.:

"vibhakti" is a feature which characterizes nouns in Sanskrit.
"number" is a feature for nouns in English.
"tense" is a feature related to verbs in English.

A word can have more than one feature associated with it, or it may not have any feature describing it. Different languages have different features associated with them. In this document both "feature" and "feature-name" have been used to refer to a feature. Morph's output consists of a listing of features with their proper values for the given word. There are only two pre-defined features in morph which apply to all words. They are "Root-Info" and "Category-Name-Map".
They shall be described in detail later. All other features have to be enumerated explicitly by the user.

FEATURE VALUE : Meanings, one or more, of a FEATURE, are called feature-values, e.g. :

Feature "gender" takes three meanings "male", "female" and "neuter".
Therefore, "neuter" is a feature value of feature "gender" and so are "female" and "male".

CATEGORY : A group of words described by the same set of features, and collectively referred to by the name of their category.
e.g.:

Words of category "noun" have features "gender", "number" and "person" in common. Here "noun" is the "category-name" of the above mentioned category of words. Similarly verbs have their own category which has the feature "tense" along with other features associated with it.
A category is characterized only by user-defined features it depends on since default features "Root-Information" and "Category-Name-Map" are common to all categories.

CATEGORY-NAME-MAP : This is a device to allow the user manipulate the category-name field of the output generated by morph. The value of this feature stays constant across all the words of a category. It enables different categories to have the same label in the category-name field in the output. Such a need arises when we want all noun-words,which have been entered into more than one category, to have the same category-name "noun" in the category-name field of the output of the morphology analyser. The values of this feature are read from the CATEGORY-NAME-MAP-FILE.
FEATURE-ENUMERATOR-FILE : This file is required in all morph sessions. It has to be created only once in the very beginning and must stay the same way, unaltered, in all subsequent sessions of morph. A change in it may call for a recompilation of Paradigm-data and Lex-data. It has as many lines as there are feature definitions, i.e., each feature definition occupies a line. No feature definition should extend across more than one line. A feature definition consists of a feature-name followed by a list of the feature-values it

takes. The number of feature-values following a feature-name in the same line is referred to as the "length" or "feature-length" of that feature. No two definitions in the same file are allowed to have the same feature-name, or in other words, no two lines in the file can have the same first entry. In case of multiple definitions of a feature-name the outcome is undefined, morph does NOT produce any messages when it encounters such a situation. The list after the feature-name must contain at least one element, it must not be empty, in other words features with a feature-length of zero are illegal. For example, a valid feature definition is:

```
number  sing  pl
```

The above definition implies that feature "number" takes two values "sing" and "pl". Feature-Enumerator-File is nothing but a collection of such definitions.
A Feature-Enumerator could look like :

```
TAM  tA yyA nA
num  p s
GENDER male female
person u m a
g_f  feminine
```

If feature definition without any feature-values is encountered morph exits after indicating an error.
Invalid feature definitions may look like :

```
num
p
s
```

CATEGORY-ENUMERATOR-FILE : This file tells morph how your categories depend on the features. There are no built in categories in morph. Any category must be defined in this file before use. A category definition consists of a category-name followed by a list of feature-names the category depends on. All the feature-names must have been defined in the accompanying Feature-Enumerator-File. In this case the list of feature-names following a category-name could be an empty. For words in such categories the word-root is assumed to be the only form of the word. A category-enumerator-file based on the sample Feature-Enumerator-File given above may look like :

```
noun GENDER num  person
noun_f  g_f num person
verb TAM GENDER num person
avyaya
```

Morph ignores invalid (undefined or misspelt) featurenames and moves on after displaying the corresponding warning. Like the feature-enumerator-file, the category-enumerator-file is referred to in all morph sessions and any modifications of which often lead to recompilation of all data.

CATEGORY-NAME-MAP-FILE : This file has two entries in each line. The first entry is the category-name of a validly declared category in the category-enumerator-file. The second entry is the string(consisting of "letter" followed by any number of letter/digit/"_") which is to be displayed in the category-name-map field of the analyser output. Any category not mentioned in this file automatically gets a "" assigned to it as a category-name-map. In case the category-name is itself its own category-name-map then it should be entered twice on the same line. A sample category-name-map-file based on the examples given in the previous definitions is given below.

```
noun    n
noun_f  n
```

```
    verb    v
    avyaya  avy
```

The above file, unlike the feature-enumerator-file and the category-enumerator-file, can be edited at will and does not call for any recompilation of either the data or the program.

PARADIGM-INPUT-FILE : This file is read by morph in compiler mode during paradigm-data input. The information collected from this file is stored in three files called Trie, Info and Rlex in a coded form. The syntax of this file is explained below. "m" is the number of word-forms prescribed for the category whose name appears in the first line of the file, and is the product of the feature-lengths of all the features associated with the category . The line numbers given in the beginning of each line are only for reference and must not be keyed in.

```
Line No.   Line contents.
1          category-name
2          word-root
3          word-form-1 , alternate-word-forms, ...
4        word-form-2 , alternate-word-forms, ...
.          .
.          .
.          .
m+3        word-form-m , alternate-word-forms, ...
```

Line 1 has only the category-name on it. Line 2 has the root on it. Note that multiple entries in either line 1 or line 2 would render the paradigm-input-file invalid. Only the root mentioned in line 2 of a paradigm-input-file can be given as a "model-root" in LEX-DATA-FILE (also see the section under LEX-DATA-FILE). The word-forms are to be listed in "last varies fastest" order. All "synonyms" should be entered in the same line separated by white-space or "," . Line no 3 should be left blank always in all such files. Line 3 is followed by m lines of word-forms and their synonyms. An empty line in any of m lines of the file after line 3 indicates the lack of a wordform for the set of feature-values corresponding to the empty line .The number of paradigms per PARADIGM-INPUT-FILE is limited to one(see utility "load"). More than one paradigm definitions in the same file will lead to the particular file being rejected by morph for having improper no of wordforms (since morph always expects only one paradigm per PARADIGM-INPUT file).

LEX-DATA-FILE : This file is to provide "lex-input" to morph. The syntax of the file is explained below. As before the line numbers are for reference only.

```
Line no         Line contents
1          n   category-1  category-2 category-3  ... category-n
2          new-root-1
3        root-info 1
4          root-info 2

.          .
.          .
.          .
```

n+2 root-info n

n+3 model-root-1 , model-root-2 , ..., model-root-n

n+4 ..

n+5 new-root-2

"n" is an integer giving the number of categories in the cat-group. It must always be followed by "n" category-names in the same line, i.e., the first line. Leave the first line of the LEX-DATA-FILE blank to indicate an empty category- group (i.e. n = 0). The number of root-info-lines should be equal to the number of categories in the category-group. The number of model-roots should also equal the number of categories present in the category-group. To indicate a "null" model-root use the NULLWORD as defined in "../nmv2/morph.h" as place marker. The place marker has to be stated explicitly wherever needed. In case no category-group is specified there should be only one root-info line. But there is now no restriction on the number of model-roots which can be listed in the n+3 th line of the lex-data for the given new-root. More than one new-root can be listed under the given category-group definition in the same file(line no n+5 and onwards). During compilation of Lex-data if more than one paradigm corresponds to the given model-root and category combination, or to the model- root alone (in case of empty category-group) a warning is issued. The given Lex-data is compiled only after the user explicitly directs the program to ignore the warning(s) and carry on. A model-root is any word root which has been entered as the root of at least one paradigm-word and therefore figures on line no 2 of at least one PARADIGM-INPUT-FILE .

The files described in the remaining portion of this chapter do not belong to this chapter if one goes srictly by the title of the chapter. The user does not have to "prepare" them in any way, or have any familiarity with their internal structure. Thes files are created and maintained by morph program by itself. The have been mentioned briefly only because they are an integral part of morph and must always be present in the user's working directory.

Using MORPH
(HINDI Example)

Sample outputs of both types, verbose and terse, for Hindi words "pioge" and "bAlaka" are given below for comparison. They are followed by relevant excerpts from the corresponding "Fe", "Ce" and "Ca" files. The definitions contained in these files apply to both "bAlakA" and "pioge".

```
VERBOSE OUTPUT FOR "bAlaka"          TERSE OUTPUT FOR "bAlaka"
     bAlaka                          #
     CATEGORY : Adj_m_s              input_word
     CNAMEMAP : adj                  adj
     ROOT     : bAlaka               bAlaka
     RINFO    :                      m
     g_m      : m                    s
     n_s      : s                    any
     ANY      : any                  #
     SUFFIX   :

VERBOSE OUTPUT FOR "pioge"           TERSE OUTPUT FOR "pioge"
```

```
     pioge                                    #
   CATEGORY : Future                          input_word
     CNAMEMAP : v                             v
     ROOT     : pI                            pI
     RINFO    :                               future
     future_tam : future                      m
     gender   : m                             pl
     number   : p                             m
     person   : m                             #
     SUFFIX   : ioge
```

"Fe" LISTING (common to both "bAlaka" and "pioge"):

```
number ,        pl,sg
gender ,        f,m
person ,        a,m,u
g_m ,           m
ANY ,           any
n_s ,    sg
future_tam ,    future
```

"Ce" LISTING (common to both "bAlaka" and "pioge"):

```
Future ,       future_tam,  gender,  number,  person
Adj_m_s ,       g_m,  n_s,  ANY
```

"Ca" LISTING (common to both "bAlaka" and "pioge"):

```
Future          v
Adj_m_s         adj
```

By manipulating the category-definition one can always control the order in which the feature-values(defined in the feature-enumerator) are listed in the output. Any information common to all words in a category can be had in the output by declaring single valued features (e. g. g_m, ANY, n_s, and future_tam) in the feature-enumerator-file and including them in the definition of the corresponding category in the category-enumerator-file.
CAUTION : Any changes in the category-enumerator-file or the feature-enumerator- file renders all previously compiled-data useless. A recompilation of the complete-data (paradigm and lex) becomes a must(see CHAPTER 1). Even paradigm-input-files may have to be modified because their structure is also connected to category-definition.

COMPILATION:

When a new .p file is added, or in an existing .p file # of lines is changed:
Do the following things.
a) .p file should be placed in pc_data sub dir.

b) Relevant info regarding the category, features & its values should
be entered in Ca, Ce & Fe files in test area.

Reference : Aksher Bharathi, Rajeev Sangal, et.al, *”Natural Language Processing: A Paninian Perspective”*

Contacts for more information

Prof. G. Uma Maheshwar Rao
(Head) CALTS, HCU.
Email : guraohyd@yahoo.com

Mrs. Amba P Kulkarni
(Head) DSS, HCU.
Email : ambapradeep@gmail.com

M. Christopher (Programmer/ IL-ILMT)
Email : efthachris@gmail.com

CALTS, Hyderabad Central University