SSF: Shakti Standard Format Guide

Akshar Bharati
Rajeev Sangal
Dipti M Sharma

Language Technologies Research Centre
International Institute of Information Technology
Hyderabad, India
{sangal,dipti}@iiit.ac.in
30 September 2007

Abstract

Shakti Standard Format (SSF) is a highly readable representation for storing language analysis. It is designed to be used as a common format or common representation on which all modules of a system operate. The representation is extensible in which different modules add their analysis. SSF also permits partial analysis to be represented and operated upon by different modules. This leads to graceful degradation in case some modules fail to properly analyze a difficult sentence.

SSF also helps in debugging, and allows the modules to be located on different machines, if necessary.

1 Introduction

Shakti Standard Format (SSF) is a common representation for data on which all modules of a system operate. This is specially designed to keep the system architecture extremely simple.

SSF allows information in a sentence to be represented in the form of one or more *trees* together with a set of *attribute-value* pairs with nodes of the trees. The attribute-value pairs allow features or properties to be specified with every node. *Relations* of different types across nodes can also be specified using an attribute-value like representation. The representation is specially designed to allow different levels and kinds of linguistic analyses to be stored. The developers use APIs to store or access information regarding structure of trees and attribute value pairs.

If a module is successful in its task, it adds a new analysis using trees and attribute values to the representation. Thus, even though the format is fixed, it is extensible in terms of attributes or analyses. This approach allows ready made packages (such as, POS tagger, chunker, and parser) to be incorporated easily using a wrapper (or a pair of converters). In order to interface such pre-existing packages to the system, all that is required is to convert from (input) SSF to the input format required by that package and, the output of the package to SSF format. The rest of the modules of the system continue to operate seamlessly.

The format allows both in-memory representation as well as stream (or text) representation. They are inter-convertible using a *reader* (stream to memory) and *printer* (memory to stream). The in-memory representation is good in speed of processing, while the stream is good for portability, heterogenous machines, and flexibility, in general.

SSF promotes the dictum: "Simplify globally, and if unavoidable, complicate only locally." However, if the number of modules is large and each module does a small job, the local complexity (of individual modules) remains under tight control for most of the modules. At worst, the complexity is introduced only locally.

2 Text Level SSF

2.1 Background

A text or document has a sequence of sentences with some structure such as paragraphs and headings. It also includes meta information related to title, author, publisher, year and other information related to origin of the text or document. Usually, there is also the information related to encoding, and version number of tagging scheme, etc. All this information is coded in SSF.

The text level SSF has two parts, header and body:

```
<document docid="..." docnumber="...">
<header>
...
</header>
<body>
```

```
...
</body>
```

The header contains meta information about the title, author, publisher, etc. as contained in the CML (corpus markup language) input. The body contains sentences, each in SSF.

2.2 Example for Text Level SSF

Here is an input example text with paragraphs and headings.

```
The Story of My Experiments with Truth
From Wikipedia, the free encyclopedia

Spiritual angle

In his own words Gandhi ...

The spiritual angle becomes ...
```

Here is the same text with meta tags in CML scheme. The initial set of tags encode the meta tags, and the body contains the actual text. CML Document:

```
<document docid="gandhi-324" docnumber="2">

<title>The Story of My Experiments with Truth </title>
<author> ... </author>

<distributor>
http://en.wikipedia.org/wiki/The_Story_of_My_Experiments_with_Truth
</distributor>

<body>

<tb number="1" segment="yes" bullet="no">
<text> Spiritual angle </text>
<foreign language="select" writingsystem="LTR"> </foreign>
  </tb>

<tb number="2" segment="no" bullet="no">
<text>
In his own words Gandhi ...
</text>
  </tb>

<tb number="3" segment="no" bullet="no">
<text>
```

```
The spiritual angle becomes ...
</text>
  </tb>
</body>

</document>
```

The above example is now shown represented in SSF with header and body. The header has the meta-tags followed by body which contains the analysis of each of the sentences:

```
<document docid="gandhi-324" docnumber="2">
<header>
<title>The Story of My Experiments with Truth </title>
<author> ... </author>
  ...
<distributor>
http://en.wikipedia.org/wiki/The_Story_of_My_Experiments_with_Truth
</distributor>
</header>

<body>
<tb number="1" segment="yes" bullet="no">   <-- Note: Value of segment attribute
<sentence>
1 Spiritual
2 angle
</sentence>
</tb>

<tb number="2" segment="no" bullet="no">
<sentence number="1">
1 In
2 his
3 own
   ...
29 him
30 .
</sentence>
<sentence number="2">
1 Going
2 through
   ...
19 follow
20 .
</sentence>
</tb>
```

```
<tb number="3" segment="no" bullet="no">
  ...
</tb>
</body>

</document>
```

For full details of the above example, see Appendix A.

2.3 Specifications

The text level SSF has two major parts: header and body. The header has information related to origin, creation, and distribution of the text. For example, information related to title of source from which the text is taken, author name, creation date, publisher, year of publication, data entry operator, name of checker, etc. Specific tags defined for keeping each of these pieces of information are taken as they are from the corpus markup scheme CML. Please consult the relevant document for details of the scheme.

The body contains the actual sentences along with paragraph structure etc.

2.3.1 Text Level SSF Tags

There are two tags at the outer most level: header and body.

```
<document docid="..." docnumber="...">
<header>
...
</header>
<body>
...
</body>
```

2.3.2 Header Tag

The beginning and end of the header is marked by:

```
<header>
```

and

```
</header>
```

respectively.

2.3.3 Body Tag

The beginning and the end of the body is marked by tags:

```
<body>
  ...
</body>
```

The body tag has several options which indicate the encoding, and version number of sentence-level SSF. Here is an example,

```
<body encode="UTF-8" SSF-version="2.0">
  ...
</body>
```

Here are the details of the two options both of which are compulsory:

1. *encode*: Encode option indicates the encoding being used for the storing the token or lexical item in the SSF (under property TKN_ to introduced later). Some example values are: ISCII, UNICODE, UTF-8, wx, etc.

2. *SSF-Version*: SSF-version indicates the version being used. There are two existing versions. New versions might also come out in the future, as the standard evolves.

2.3.4 Structure of Body

The body of a text in SSF contains text blocks given by the tag *tb*.

```
<body encode= ... >
<tb>
...
   </tb>
<tb>
...
   </tb>
</body>
```

A text block (tb) contains a sequence of sentences. The structure of a document in text level SSF is indicated by Fig. **??**.

Figure 1: Document Structure in SSF

Each sentence can be marked as a *segment* (to indicate a heading, a partial sentence, etc.) or not (to indicate a normal sentence).

```
<body encode= ... >
<tb number=1>
<sentence segment="yes" number=1>
Spiritual angle
</sentence>
   </tb>

<tb number=2>
<sentence segment="no" number=1>
1 In
```

```
2 his
   ...
</sentence>
<sentence segment="no" number=2>
1 Going
2 through
   ...
</sentence>
   </tb>

<tb number=3>
...
   </tb>
</body>
```

3 Sentence Level SSF

3.1 Background

Sentence level SSF is used to store the analysis of a sentence. It occurs as part of text level SSF. The analysis of a sentence may mark any or all of the following kinds of information as appropriate: part of speech of the words in the sentence; morphological analysis of the words including properties such as root, gender, number, person, tense, aspect, modality; phrase-structure or dependency structure of the sentence; and properties of units such as chunks, phrases, local word groups, bags, etc. Note that SSF is theory neutral and allows both phrase structure as well as dependency structure to be coded, and even mixed in well defined ways.

Several formalisms have been developed for such descriptions but the two main ones in the field of NLP are Phrase Structure Grammar (PSG) and Dependency Grammar (DG). In PSG, a set of phrase structure rules are given for the grammar of a language. It is constituency based and order of elements are a part of the grammar, and the resulting tree. DG, on the other hand, is relational and shows relations between words or elements of a sentence. It, usually, tries to capture the syntactico-semantic relations of the elements in a sentence. The resulting dependency tree is a tree with nodes and edges being labelled. The difference in the two approaches are shown below with the help of the following English example:
Example: Ram ate the banana. The phrase structure tree is drawn in Fig. 2 using a set of phrase structure rules. Fig. 3 shows the dependency tree representation. SSF can represent both the formats.

Figure 2: Phrase structure tree

Figure 3: Dependency tree

Though the SSF format is fixed, it is extensible to handle new features. It also has a text representation, which makes it easy to read the output. The following example illustrates the SSF. For example, the following English sentence,

```
Children are watching some programmes on television in the house. --(1)
```

contains the following chunks (enclosed by double brackets),

```
((Children)) [[are  watching]] ((some  programmes))
((on television)) ((in the house))
```

All the chunks are noun phrases, except for one ('are watching') which is a verb group and is shown enclosed in square brackets. If we mark the part-of-speech tag for each word, we have the following:

```
((Children_NNS)) [[are_VBP watching_VBG]]
((some_DT programmes_NNS)) ((on_IN television_NN))
((in_IN the_DT house_NN))
```

The representation above is shown in SSF in Fig. 5.

```
Address   Token         Category
 ---------------------------------------------------
 1        ((              NP
 1.1      children        NNS
          ))

 2        ((              VG
 2.1      are             VBP
 2.2      watching        VBG
          ))

 3        ((              NP
 3.1      some            DT
 3.2      programmes      NNS
          ))

 4        ((              PP
 4.1      on              IN
 4.1.1    ((              NP
 4.1.2    television      NN
          ))
          ))

 5        ((              PP
 5.1      in              IN
 5.2      ((              NP
 5.2.1    the             DT
 5.2.2    house           NN
          ))
          ))
 ---------------------------------------------------
```

Fig. 5: Towards Shakti Standard Format

As shown in Fig. 5, each line represents a word/token or a group (except for lines with '))' which only indicate the end of a group). For each group, the symbol used is '((' . Each word or group has 3 parts. The first part stores the tree address of each word or group, and is for human readability only. The word or group is in the second part, with part of speech tag or group/phrase category in the third part.

The example below shows the SSF for the first noun phrase where feature information is also shown, as the fourth part on each line.

```
1       ((              NP      <fs root=child cat=np gend=m num=p pers=3>
1.1     children        NNS     <fs root=child cat=n gend=m num=p pers=3 case=0>
        ))
```

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

```
1       ((              NP
1.1     children        NNS     <fs af='child,n,m,p,3,0,,' >
                                        |    | | | | |
                                        |    | | | |  \
                                      root   | | |pers |
                                             | | |    case
                                    category | number
                                             |
                                          gender
```

The field for each attribute is at a fixed position, and a comma is used as a separater. Thus, in case, no value is given for a particular attribute the field is left blank, e.g. last two fields in the above example.

The representation in SSF of sentence 1 with feature structures is given in Fig. 5 (abbreviated attribute 'af' is used).

```
Address   Token         Category        Attribute-value pairs
  ------------------------------------------------
  1       ((              NP
  1.1     children        NNS     <fs af=child,n,m,p,3,0,,>
          ))
  2       ((              VG
  2.1     are             VBP     <fs af=be,v,m,p,3,0,,>
  2.2     watching        VBG     <fs af='watch,v,m,s,3,0,,' aspect=PROG>
          ))
  3       ((              NP
  3.1     some            DT      <fs af=some,det,m,s,3,0,,>
  3.2     programmes      NNS     <fs af=programme,n,m,p,3,0,,>
          ))
  4       ((              PP
  4.1     on              IN      <fs af=on,p,m,s,3,0,,>
  4.1.1   ((              NP
  4.1.2   television      NN      <fs af=television,n,m,s,3,0,,>
          ))
          ))
  5       ((              PP
  5.1     in              IN      <fs af=in,p,m,s,3,0,,>
  5.2     ((              NP
  5.2.1   the             DT      <fs af=the,det,m,s,3,0,,>
  5.2.2   house           NN      <fs af=house,n,m,s,3,0,,>
```

```
        ))
        ))
-----------------------------------------------
        Fig. 5: Shakti Standard Format
```

3.2 Specifications

The SSF representation for a sentence consists of a sequence of *trees*. Each tree is made up of one or more related *nodes*.

A node has *properties* which are given by prop-name and prop-val. For example, a node may have a word 'she' associated with it along with gender 'f'. These may be stored or accessed using prop-name TKN_, and gender attribute, respectively.

Every node has four "system" properties:

- Address - referred to by property name ADDR_
- Token - acccessed by attribute name TKN_
- Category - accessed by attribute name CAT_
- Others - used to store user-defined features which are accessed through their feature names or attribute names.

A property has a prop-name and prop-val. Here are a few examples for the system properties:

Property name	*Example property values*
ADDR_	1.2.4, 3.2, 2
TKN_	'((', she, eating
CAT_	NN, VB, NP, VG

Example: Given below are two nodes (or trees with a single node each), marked by address labels 1 and 2, having their respective tokens as 'children' and 'played', and their categories as NN and VB:

ADDR_	TKN_	CAT_
1	children	NN
2	played	VB

Elements in each row are separated by a single tab character.

Corresponding to the above SSF text stream, an in-memory data structure may be created using the APIs. (However, note that value of the property ADDR_ is not stored in the in-memory data structure explicitly. It is for human reference and readability only, and is computed when needed.)

Notation used for describing the values is given next. A value is given by any of the following including their combination:

- Address. Consider the example,

```
<nnumber> (.<nnumber>) *
```

  where $< nnumber >$ is natural number (including zero) given by:

```
<nnumber> ::= <digit> <digit>*
<digit> ::= 0|1|2|3|4|5|6|7|8|9
```

  and where '*' indicates repetition zero or more times, '(' and ')' act as delimiters.

- Lexical value. Consider the example,

```
'((' | <alpha-num-tok>
```

  where either the two opening parentheses or $< alpha - num - tok >$ is present. $< alpha - num - tok >$ is defined as follows:

```
<alpha-num-tok> ::= <alpha-num-u>* <alpha-num>
<alpha-num-u> ::= <alpha-num>_
<alpha-num>   ::= (<alphabet>|<digit>) (<alphabet>|<digit>)*
<alphabet>    ::= A|B| ... |Z|a|b| ... |z
```

- Category names. For the property CAT_, the permissible values are given by a dictionary of terms.

```
Property name      Permitted Property values

CAT_            {NP,VG, ...} if CAT_ is a phrasal category
                {NN,JJ, ...}  if CAT_ is a part of speech category
```

  The above can be written as:

```
Property name      Property value format

CAT_            pcat(CAT_) -> {NP,VG, ...};
                lcat(CAT_) -> {NN,JJ, ...}
```

  where *pcat* is a procedure which gives the list of phrasal categories if value of CAT_ is a phrasal category, and *lcat* gives the list of lexical category if it is a lexical category.

3.2.1 Attributes and Values

A node may have one or more features. A feature consists of attribute-value pair.

Example: Two nodes with attributes 'root':

ADDR_	TKN_	CAT_	OTHER_
1	children	NN	$< fsroot = childnum = pl >$
2	played	VB	$< fsroot = playtense = past >$

Note that the feature 'root' has values 'child' and 'play' for the respective nodes. Similarly, features 'num' (number) with value 'pl' (plural), and 'tense' with value 'past' are also shown.

To give the specification of the format of attributes and values, some definitions are introduced first::

- Alpha-numeric $AN = [a-zA-ZO-9]+

  '[a-z]' stands for a character out of the lower case range: 'a' to 'z'. (In standard Unix notation for regular expressions, square brackets indicate that any of the characters is permitted, and the hyphen ('-') shows the range. Thus, 'a-z' means any lower case alphabetic character, etc. A '+' indicates one or more repetitions, whereas '*' means zero or more repetitions.)

- Alpha-numeric single-underscore:

  Alpha-numeric strings possibly separated by single underscores (without two underscores in a sequence and not ending or beginning with an underscore)

  ```
  $ANSU = $AN(_$AN)*
  ```

- Alpha-numeric double-underscore: Alpha numeric single underscore strings separated by two possible double underscores

  ```
  $ANDU = $ANSU(__$ANSU)*
  ```

Now,

(i) An *attribute* is defined by $ANSU or $ANSU followed by underscore.

```
<attribute> ::=  $ANSU | $ANSU _
```

(ii) A *value* is defined by $ANDU (which includes $ANSU).

```
<value>     ::=  $ANDU
```

A simple value is defined by $ANSU.

Attributes

There are two types of attributes - user defined or system defined. The convention that is used is that a user defined attribute should not have a underscore at the end.

System attribute may have a single underscore at its end.

Values

Values are of two types: simple and structured. Simple values are represented by $ANSU. Structured values have progressively more refined values separated by double underscores. For example, if a value is:

```
vmod__varg__k1
```

it shows the value as 'vmod' (modifier of a verb), which is further refined as 'varg' (argument of the verb) of type 'k1' (karta karaka).

A value X covers another value Y, if Y is structured and more refined than X. In other words, Y is of type X, and is more specified, or refined than X.

For example, a value B covers value C in the following:

```
B = vmod__varg
C = vmod__varg__k1
```

B says that something is an argument of a verb (vmod__varg), and C says that it is an argument of type k1. This indicates that C is a refined or detailed form of B.

If a constraint says that the value must be of type X, then Y also satisfies the constraint. For example, if value B is constrained to be of type 'vmod__varg' then clearly both B and C satisfy the constraint. Thus, value X is covered by a value Y if the following holds:

case (i): X = Y

case (ii): X is a prefix of Y, followed by two underscores and $ANDU.

3.2.2 Interlinking of nodes

Nodes might be interlinked with each other through directed edges. Usually, these edges have nothing to do with phrase structure tree, and are concerned with dependency structure, thematic structure, etc. These are specified using the attribute value syntax, however, they do not specify a property for a node, rather a relation between two nodes.

For example, if a node is karta karaka of another node named 'play1' in the dependency structure (in other words, if there is a directed edge from the latter to the former) it can be represented as follows:

1	children	NN	$< fs\ drel =' k1 : play1' >$
2	played	VB	$< fs\ name = play1 >$

The above says that there is an edge labelled with 'k1' from 'played' to 'children' in the 'drel' tree (dependency relation tree). The node with token 'played' is named as 'play1' using a special attribute called 'name'.

So the syntax is as follows: if you associate an arc with a node C as follows:

```
<treename>=<edgelabel>:<nodename>
```

it means that there is an edge from $< nodename >$ to C, and the edge is labelled with $< edgelabel >$. Name of a node may be declared with the attribute 'name':

```
name=<nodename>
```

(All the words in angle-brackets may be substituted with appropriate user-defined names.)

4 Cross Linking across Sentences

4.1 Background

There is a need to relate elements across sentences. A well known case is that of co-reference of pronouns. For example, in the following sentences:

```
Sita saw Ram in the house. He had come all by himself.
```

the pronoun 'he' in the second sentence refers to the same person as referred to by 'Ram'. Similarly 'himself' refers to same person as 'he' refers to. This is show by means of a coreference link from 'he' to 'Ram', and from 'himself' to 'he'. SSF allows such cross-links to be marked.

The above text of two sentences is shown in SSF below.

```
<document docid="gandhi-324" docnumber="2">
<header> ...  </header>
<body>
<tb>
  <sentence num=1>
  1 Sita              <fs name=R>
  2 saw
  3 Ram
  4 in
  5 the
  6 house
  7 .
  </sentence>
  <sentence num=2>
  1 He                <fs coref="..%R"  name=he>
  2 had
  3 come
  4 all
  5 by
  6 himself           <fs coref=he>
  7 .
  </sentence>
</tb>
```

Note that 'himself' in sentence 2 corefers to 'he' in the same sentence. This is shown using attribute 'coref' and value 'he'. To show coreference across sentences, a notation is used with '%'. It is explained next.

Name labels are defined at the level of a sentence: Scope of any name label is a sentence. It should be unique within a sentence, and can be referred to within the sentence by using it directly.

To refer to a name label in another sentence in the same text block (paragraph), path has to be specified:

..\%R

To refer to a name label R in a sentence in another text block numbered 3, refer to it as:

..\%..\%3\%1\%R

One can refer to a registered corpus C using ' C'. This will be expanded in time to come.)

5 References

Bharati, Akshar, Rajeev Sangal, Dipti M Sharma, Shakti Natural Language Analyzer: SSF Representation Unpublished manuscript, LTRC, IIIT Hyderabad. (Available on http://ltrc.iiit.ac.in/ILMT)
(Contains a longer description with many examples, but also with the choice of dependency structures and how SSF helps in system building is available.)

A Example: Text in CML and SSF formats

Here is an input example text with paragraphs and headings.

```
The Story of My Experiments with Truth
From Wikipedia, the free encyclopedia

Spiritual angle

In his own words Gandhi takes us through some of the experiences in his
life, with each chapter forming at least one important learning lesson to
him. Going through the introduction section of the autobiography may
suggest what to expect during the five parts that follow.

The spiritual angle becomes evident when Gandhi says, "...What I want to
achieve - What I have been striving and pining to achieve these thirty
years - is self-realization, to see God face to face, to attain Moksha
(Salvation). I live and move and have my being in pursuit of this goal."
```

Here is the same text with meta tags in CML scheme. The initial set of tags encode the meta tags, and the body contains the actual text. CML Document:

```
<document docid="gandhi-324" docnumber="2">

<title>The Story of My Experiments with Truth </title>
<author>
<firstname>Mohandas</firstname>
<middlename>Karamchand</middlename>
<lastname>Gandhi</lastname>
</author>

<creation creationdate="03/07/2007" institutename="IIIT, Hyderabad"></creation>
<distributor>
http://en.wikipedia.org/wiki/The_Story_of_My_Experiments_with_Truth
</distributor>

<language name="En" writingsystem="LTR" script="Roman" />
<projectdesc name="ILMTConsortium"/>
<dateofpublication>1927</dateofpublication>

<body>

<tb number="1" segment="yes" bullet="no">
<text> Spiritual angle </text>
<foreign language="select" writingsystem="LTR"> </foreign>
```

```
  </tb>

<tb number="2" segment="no" bullet="no">
<text>
In his own words Gandhi takes us through some of the experiences in his
life, with each chapter forming at least one important learning lesson
to him. Going through the introduction section of the autobiography
may suggest what to expect during the five parts that follow.
</text>
<foreign language="select" writingsystem="LTR"> </foreign>
  </tb>

<tb number="3" segment="no" bullet="no">
<text>
The spiritual angle becomes evident when Gandhi says, "...What I
want to achieve - What I have been striving and pining to achieve
these thirty years - is self-realization, to see God face to face,
to attain Moksha (Salvation). I live and move and have my being in
pursuit of this goal."
</text>
<foreign language="select" writingsystem="LTR"> </foreign>
  </tb>
</body>

</document>
```

The above example is now shown represented in SSF with header and body. The header has the meta-tags followed by body which contains the analysis of each of the sentences:

```
<document docid="gandhi-324" docnumber="2">
<header>
<title>The Story of My Experiments with Truth </title>
<author>
<firstname> Mohandas </firstname>
<middlename> Karamchand    </middlename>
<lastname> Gandhi </lastname>
</author>
  ...
<language name="En" writingsystem="LTR" script="Roman"/>
<projectdesc name="ILMTConsortium"/>
<dateofpublication>1927</dateofpublication>
</header>

<body>
<tb number="1" segment="yes" bullet="no">   <-- Note: Value of segment attribute
<sentence>
```

```
1 Spiritual
2 angle
</sentence>
</tb>

<tb number="2" segment="no" bullet="no">
<sentence number="1">
1 In
2 his
3 own
   ...
29 him
30 .
</sentence>
<sentence number="2">
1 Going
2 through
   ...
19 follow
20 .
</sentence>
</tb>

<tb number="3" segment="no" bullet="no">
  ...
</tb>
</body>

</document>
```

Appendix - B

B Dictionary of SSF Attributes and Values

B.1 Node Types

Node types can be represented in the feature structure by the attribute 'ntype'. The values for the ntype attribute would be any one of the following : pos, lwg, bag, phr

B.1.1 pos

: When the node has a part of speech as its category. Following Telugu example depicts it -

```
(( PRP <ntype=pos>
baMgArapu NN
avi PRP
  ))
```

B.1.2 lwg

: a word group. For example,

```
(( NP <ntype=lwg>
rAma
ko
))
```

B.1.3 bag

: a group of words where internal dependencies are not known or are not marked. For example,

```
(( NP <ntype=bag>
My
younger
brother
Ram
))
```

The expanded dependency tree for the above 'bag' is given in Figure-**??**

Figure 4:

B.1.4 phr

The value 'phr' would be assigned to a phrasal node within a phrase structure paradigm. For example,

```
((  PP <ntype=phr>
to
(( NP <ntype=phr>
My
younger
brother
Ram
))
))
```

Although, the NP in the above example appears to be the same as the NP in the previous example (**??**), the grammatical framework in which it is analyzed here is different. **??** is within dependency framework and **??** is in the phrase structure grammar framework. The expanded phrasal tree structure for the above example is represented in Figure-**??**.

Figure 5:

B.2 Predicate for Dependency relation type

(Note : This predicate will not be used in the ILMT system as of now)

The predicate for dependency relations is 'drel'. The top level values for this attribute are - nmod, vmod, jjmod, and rbmod. Each of these values may have subtypes, particularly, vmod and nmod. Figure-**??** and Figure-**??** represent the subtypes for nmod and vmod respectively.

Figure 6:

The leaf nodes in the Figures **??** and **??** are the final values for the attribute 'drel'. In case the system fails to get the final value for a given node, the value of the parent node is given. For example, if the deeper level value of the drel attribute is not known for a node, a higher level value can be given. This point is further explained through Figures 8 and 9 below.

```
1. (( NP <fs drel=vmod__varg__k1:watching>
1.1 Children
1.2 ))
2 (( VG <fs name=watching>
2.1 are
2.2 watching
```

```
))
3. (( NP <fs drel=vmod__varg__k2:watching>
3.1 some
3.2 program
))
4 (( PP <fs drel=vmod__varg__k7p:watching>
4.1 on
4.2 television
))
5. (( PP <fs drel=vmod__varg__k7p:watching>
5.1 at
5.2 home
))
Figure 8
```

Figure 7:

Figure-8 shows the fine-grained values of the drel attribute for the NPs and PPs. The NPs 'children' and 'some program' have the kakrak values 'k1' and 'k2'. Both the PPs 'on television' and 'at home' have k7p as their karaka values since both indicate 'place'. However, Figure-9 represents the drel values for both the PPs as 'ky'. This indicates that the more fine grained drel values for these PPs could not be obtained.

```
1. (( NP <fs drel=vmod__varg__k1:watching>
1.1 Children
1.2 ))
2 (( VG <fs name=watching>
2.1 are
2.2 watching
))
3. (( NP <fs drel=vmod__varg__k2:watching>
3.1 some
3.2 program
))
4 (( PP <fs drel=vmod__varg__ky:watching>
4.1 on
4.2 television
))
5. (( PP <fs drel=vmod__varg__ky:watching>
5.1 at
5.2 home
))

Figure 9
```

The value 'ky' indicates that the PPs are kakakas other than k1, k2 and k4. Similarly, if it is known for a node that it is a 'varg' but not exactly which varg, the value can be given as varg and so on.

Although the drel values given in Figures-5 and 6 represent the tree structure, to save typing effort, one can give only the final known value. For example, one can only give the drel value as drel=k1. All the values for the attribute 'drel' are given in Table 1 at the end of this document.

B.3 Predicate for Grammatical Role Type

(Note : This is not being used in the ILMT System as of now)

The attribute for the grammatical role type is 'grole' The values for this attribute are fixed depending on the syntactic properties of a language. For example for English these values would be : Subject, Object, Object2, prep__on, prep__in (etc), comp__that etc. The double underscore in any value indicates a subtype. For example, the value 'prep__on' states 'preposition of the type on' .

B.4 Some other Predicates

(Note : This is not being used in the ILMT System as of now) Certain other attributes which can occur in SSF are : Sentence Type as stype, theta roles as trel, named entities as nlp_sem, sense as SID, class as CLID, Coreference as coref,

C Dictionary of Categories

C.1 Predicates and POS Categories

Predicate is a program that returns true if the listed category is given as argument.

```
   Type Phrases/bags Lexical Head

   nounp NP NN,NNP,NST,PRP,WQ
   vrbp VGF, VGNF,VM
                  VGINF, VGNN
   adjp JJP JJ,QF
   advp RBP RB,QF,WQ
   conjp CCP CC
   negp NEGP NEG
   fragp             FRAGP VAUX,PSP,DEM
   NILp             BLK SYM,INJ,UNK

Predicates for the following will be decided in due course:

QC, QO, RP, CL, INTF, ECH, XC, RDP, UT
```