AUTOMATIC LFG GENERATION

MS Thesis for the Degree of

Submitted in Partial Fulfillment
of the Requirements for
the Degree of

Master of Science (Computer Science)

at the

National University of Computer and Emerging Sciences

by

Umer Khalid Qureshi

2009

Approved:

_____

Head
(Department of Computer Sciences)

_____ 20 _____

Approved by Committee Members:

Advisor

_____

Dr. Sarmad Hussain
Professor
National University of Computer & Emerging Sciences

Co –Advisor

_____

Mr. Shafiq-ur-Rahman
Associate Professor
National University of Computer & Emerging Sciences

_____

Ms. Nayyara Karamat
Senior Development Engineer
National University of Computer & Emerging Sciences

VITA

Mr. Umer Khalid Qureshi was born in Lahore, Pakistan on December 04, 1983. He received a Bachelor of Science in Computer Science from PUCIT, Lahore in 2005. He had been associated with CRULP-NUCES as a Research Officer from 2007 to 2008. His area of interest includes Natural Language Processing in general and Text Processing in particular.

Acknowledgments

I am most grateful to Allah, who has empowered me to accomplish my all tasks including this research.

I am grateful to my advisor Dr. Sarmad Hussain as well, for his kind considerations, encouragement, guidance, supervision and support throughout the course of this research work. I am also thankful to him for trusting me and for all the facilities he provided me to complete my research. I am thankful to my co-advisors, Mr. Shafiq-ur-Rahman and Ms. Nayyara Karamat, for the guidance and help. Both of them have been a source of inspiration and kind help for me.

I am also thankful to CRULP for providing me an opportunity to develop this research work. I must thank my colleagues at CRULP as well specially Mr. Aasim Ali and Mr. Shahid Iqbal for helping me in the implementation of my system.

Umer Khalid Qureshi

Table of Contents

Table of Figures

1 INTRODUCTION

The demand of the Natural Language Processing is to enable computer to understand the language of human. Syntax of a language plays a primary role in that language. Generative Grammar is an approach to study and compute the syntax of a particular natural language. The generative grammar of a language includes a set of rules that will correctly calculate if a combination of words is grammatically correct. Lexical Functional Grammar is also a variety of generative grammars. The major focus of this grammar is to analyze the syntax of a language with perspective of generally two syntactic structures [35]: (1) the outer structure (C-Structure) deals with the visible hierarchical organization of words into phrases and (2) the inner structure (F-Structure) contains abstract relational information of the outer structure [6].

The aim of this work is to present a system that automates development of Lexical Functional Grammar for English using Templates. It also discusses the issues with different types of grammar development practices. The following sections include the LFG formalism and linguistic analysis needed for the development of unification based functional grammars. Later, we review couple of parsing architecture needed to generate F-Structure of a sentence. We present the problem statement of the thesis followed by the methodology for proposed system and linguistic analysis needed for our grammar development. We also report the results compiled from the development of our proposed system.

2 BACKGROUND

In this section, we review the information needed to understand the following section of problem statement. It includes the linguistic and computational aspects of Lexical Functional Grammars.

2.1 Lexical Functional Grammar

This section discusses a brief overview of a linguistic formalism established for analysis and representation of natural languages particularly with respect to machine translation.

Lexical Functional Grammar [10] is a formal theory of language. The major focus of theoretical linguistics researchers in the linguistic formalism has been syntax [11]. This unification based linguistic formalism is used mostly for computation and syntax processing. LFG has different levels of structures to represent the linguistic information. This thesis covers only two components [11]: C-Structure and F-Structure.

2.1.1 C-Structure

Representation of the hierarchal grouping and sequence of the words in phrases is called constituent structure. This representation also shows the syntactic categories of the words (Part Of Speech). The constituents maintain their linear order in the representation. In other words, this representation shows how phrases are formed with combinations of words; and the sentences as hierarchal combination of phrases. These hierarchical groupings are describable by phrase structure rules commonly represented as a context-free grammar.

Example 1:

Consider the sentence "`he ate apples with me`". The phrase structural rules describing the hierarchal constituent structure[1] of this sentence are as follows.

[1] The output of Collins' Parser [22] [23].

```
S →  NP VP

VP → vbd NP PP

NP → prp | nns

PP → in NP
```

The parsing using the above Context Free Grammar (CFG) can depict the constituent structure of the sentence “he ate apples with me” as follows:

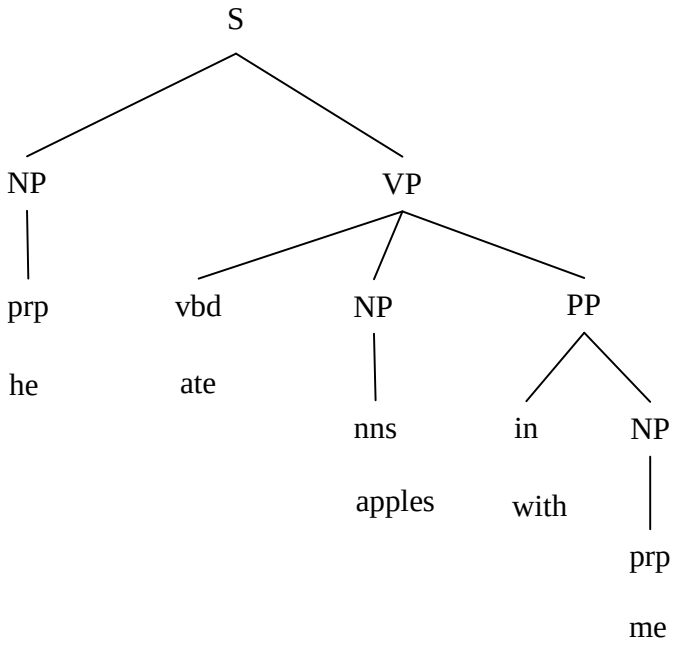


Figure 1: C-Structure of 'he ate apples with me'

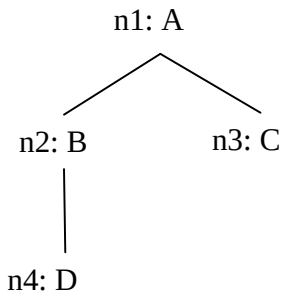
Here, ‘prp’, ‘vbd’, ‘nns’ and ‘in’ are parts of speech and refer to personal pronoun, past participle of verb, plural common noun and preposition respectively.

C-Structure represents information about the part-of-speech in each constituent and the syntactic structure of the sentence. Since the POS is the terminal element in this grammar, hence it is independent of words. C-Structure licenses the constituencies of a language at POS level. The lexicon binds the POS with the words which can aid to analyze the source sentence.

2.1.1.1 Descriptive Representation of C-Structure

The model-based representations of structures (e.g. tree) can also be represented in descriptive and declarative form [12]. The properties of one structure are used to generate formal descriptions of other representations. These formal descriptions are in the form of a collection of constraints on the relations that those structures must possess. The structures that satisfy all the propositions in the description are acceptable [12]. The description of structure implies the writing of defining properties and relation in that structure. We construct a tree and describe it in our descriptive representation in Example 2.

Example 2:




```
M(n2) = n1          λ(n2) = B
M(n4) = n2          λ(n3) = C
M(n3) = n1          λ(n4) = D
n2 < n3
λ(n1) = A
```

Figure 2: Tree with descriptive form

Example 2 describes that parent of 'n2[1]' is 'n1', represented by 'M(n2) = n1'. Similarly, parent of 'n4' and 'n3' are 'n2' and 'n1', respectively. 'n2' is on the left of 'n3', represented by 'n2 < n3'. The value of 'n1' is A, 'n2' is B, 'n3' is C and 'n4' is D, as represented by the 'λ' relation. 'n4' and 'n3' have no direct relation and hence not described in the descriptive form. Similar is the case with relation of 'n1' and 'n4'. As they both do not have a direct arch, so they can't be described unless queried as 'M (M (n4)) = n1'. The symbols 'M', '<' and 'λ' are for representation purpose and may vary. The potential strength of this abstract representation is discussed later in Section 2.3.4.2 .

[1] 'n2' is the representation name of the node with value B in the tree.

2.1.2 F-Structure

It is not sufficient to know the information about the external structure of the sentence; the relation of phrases in which they may occur is also important [26]. C-Structure captures the external aspects whereas F-Structure covers the internal aspects of the sentence. F-Structure represents the higher syntactic information along with functional information in a sentence. The higher syntactic information describes the grammatical attributes of a word for instance ‘he’ is third person singular pronoun and ‘ate’ is past participle of ‘eat’. The functional information of a sentence describes the relations between words and phrases for instance; ‘he’ is the subject of ‘ate’ in Example 1 above. F-Structure can also represent the kind(s) of syntactic function a predicator[1] may have [26]. Usually the higher syntactic and functional information is shown as an attribute-value pair [13]. These pairs form the nodes of an acyclic graph structure. The attribute-value pair in F-Structure is represented such that an attribute can be a grammatical symbol (for instance: Number) or a grammatical function (for instance: Subject) and the value for that attribute can be an atomic symbol (for instance: Singular), a semantic form (as illustrated in Example 3 below) or a subsidiary F-Structure (see Example 4 below) [14]. An atomic value describes the grammatical feature of a constituent. The semantic form represents the semantic interpretation of a predicate. The semantic interpretation is represented in terms of the syntactic functions that a predicator can have. Semantic form is usually represented as ‘PRED’ [26].

Example 3:

Following is the semantic form of ‘ate’.

$$\begin{bmatrix} \text{PRED} & \text{'eat<(↑SUBJECT)(↑OBJECT)>'} \end{bmatrix}$$

The semantic form is more related to sub-categorization frame than representing the exact semantic form in F-Structure [11]. The purpose of semantic form is to encode the number and type of grammatical functions for that particular predicate.

[1] Predicate is usually the head of a phrase and the verb or the preposition in clauses [26].

The subsidiary F-Structure of a syntactic function is represented in Example 4 below.

Example 4:

The Noun Phrase (represented as NP) capturing ‘apples’ is the object of the verb ‘ate’ in Example 1. This can be represented in the F-Structure of the sentence as follows.

$$\begin{bmatrix} \text{OBJ}^{1} & \begin{bmatrix} \text{PRED} & \text{'apple'} \\ \text{NUM} & \text{PL} \end{bmatrix} \end{bmatrix}$$

The subsidiary F-Structure nature of these representations makes them look like tree; however, there can be some cross level links violating the definition of tree (see Figure 12).

The attribute-value pair in an F-Structure is independent of order. The C-Structures shown in Figure 1 above do not carry any functional description about the constituents and hence cannot assign the F-Structures to themselves. Thus, the grammar rules are annotated with the functional information as shown in the Example 5 below.

Example 5:

$$S \rightarrow NP\ (\uparrow \text{SUBJ}^{2} = \downarrow)\ VP\ (\uparrow = \downarrow)$$

The phrase structure rule in Example 5 has been annotated with functional equations to specify the mapping from C-Structure to F-Structure, termed as ‘ø’. The functional equation employ two meta variables, ↑ and ↓. The ↑ refers to the F-Structure associated with the parent node, while ↓ represents the F-Structure associated with the current (self) node. In the functional equations, = is used for unification of the F-Structure attributes [16] [10]. As a consequence, the grammar rule in Example 5 can be described as follows.

[1] Short form of OBJECT.
[2] Short form of SUBJECT.

The node S has a left child NP and a right child VP. The F-Structure associated with S is unified with the F-Structure of VP (↑ = ↓). The value of SUBJ attribute of the F-Structure of S is unified with the F-Structure of NP (↑ SUBJ = ↓).

In LFG formalism, entries in the lexicon (lexical items) also have the functional information with them. The lexical item for words “ate” and “apples” is following.

Example 6:

```
ate: vbd, ↑ PRED = 'eat<SUBJ,OBJ>' ,
          ↑ TENSE = PAST .

apples: nns, ↑ PRED = 'apple' , ↑ NUM = PL .
```

Thus, the lexical items are used to deliver attribute value pairs to the leaf nodes in the parse tree.

2.1.2.1 Unification

Unification is the process of merging the information content of two structures and ruling out the merger of structures that are incompatible with each other [15]. The term ‘structure’ is used as abstraction of semantic form, atomic value and subsidiary F-Structure. The following Example 7 [26] illustrates the results of unification.

Example 7:

(7.1)

$$[\text{NUMBER} \quad \text{PL}] \cup \begin{bmatrix} \text{NUMBER} & \text{PL} \\ \text{GENDER} & \text{F} \end{bmatrix} = \begin{bmatrix} \text{NUMBER} & \text{PL} \\ \text{GENDER} & \text{F} \end{bmatrix}$$

(7.2) $[\text{NUMBER} \quad \text{PL}] \cup [\text{NUMBER} \quad \text{SG}] = \text{Failure}$

(7.3) $[\text{NUMBER} \quad \text{PL}] \cup \text{NULL} = [\text{NUMBER} \quad \text{PL}]$

In (7.1), the unification operator successfully unifies its both operands which are structurally different but compatible. (7.2) shows a failure in unification because of

incompatible or inconsistent information for same attribute [15]. In Example (7.3), unification with 'NULL' results the other argument unchanged.

"The feature structures are represented as directed acyclic graphs (DAGs), where features are depicted as labels on directed edges, and feature values are either atomic symbols or DAGs"[15]. The unification operator is somewhat straightforward recursive graph matching algorithm, customized to carry through the different requirements of unification. The details of unification process are addressed in [15].

2.1.2.2 Building F-Structure Using LFG

The F-Structure can be built by parsing the sentence from the LFG of that language. The building of F-Structure can be shown with the help of Example 8 below.

Example 8:

The relative Lexical Functional Grammar of the sentence '`he ate apples with me`' can be following (using Penn Treebank Tagset [34]).

```
S → NP (↑SUBJ = ↓) VP (↑=↓)

VP → vbd(↑=↓) NP(↑OBJ=↓) PP(↑PREP=↓)

NP → prp(↑=↓)

NP → nns(↑=↓)

PP → in(↑=↓) NP(↑OBJ=↓).
```

And the C-Structure (parse tree) using the above LFG can be as shown in Figure 3. The F-Structure is built using the unification process starting from the lexical level. F-Structure at each node is the result of unification from its child nodes. As a start, the tree can be annotated using the functional information from LFG rules as Figure 4 shows.

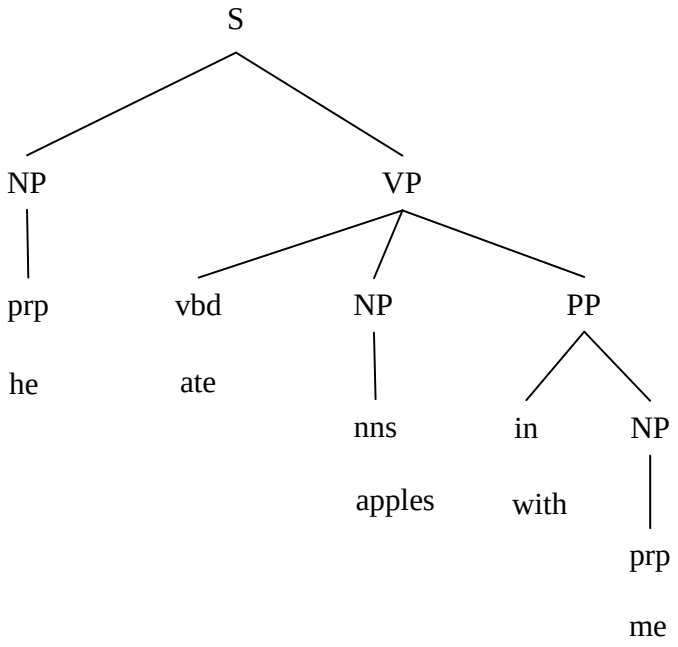


Figure 3: Parse Tree using CFG

.

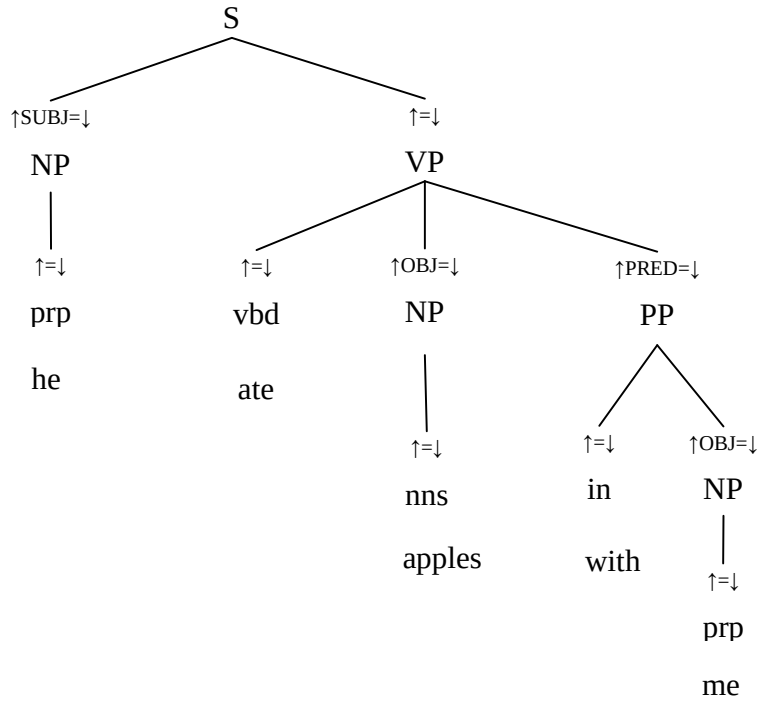


Figure 4 : Parse tree annotated with functional description.

Each node knows its relation with its parent. For instance, the NP is SUBJ of parent's F-Structure and VP unifies all its features to its parent without any subsidiary classification. Similarly, NP as a child node of VP has relation of OBJ with its parent node.

Each word and its POS are coupled within the constituent. The feature description of each word builds F-Structure of corresponding POS in the tree. The F-Structures flow towards root from each node to result a single F-Structure for a sentence. It used unification at each node with its children. The Example 9 illustrates the F-Structure building. A shorter sentence is used to make the process easy to understand.

Example 9:

The C-Structure for '`he sleeps`' is as follows using LFG rules given:

(9.1) `S → NP: ↑ SUBJ = ↓ ;  VP: ↑ = ↓ ; .`

(9.2) `NP → prp: ↑ = ↓; .`

(9.3) `VP → vbz : ↑ = ↓ ; .`

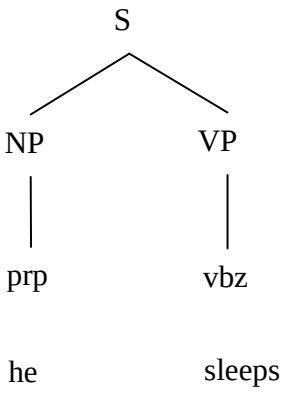


Figure 5 : Parse Tree of 'he eats' with CFG

And the lexical entries for given words

```
he: prp, ↑ PRED ='pro',↑ NUM= SG, ↑ GEND = M ,
         ↑ PERS= 3.
```

```
sleeps: vbz, ↑ PRED = 'sleep<↑SUBJ>' ,
             ↑ TENSE = PRES ,  ↑ NUM = SG ,
             ↑ PERS = 3 .
```

f1:

PRED	‘pro’
NUM	SG
PERS	3
GEND	M

f2:

PRED	‘eat<SUBJ>’
TENSE	PRES
NUM	SG
PERS	3

Figure 6 : The F-Structures of 'he' and 'eats'

To build a single F-Structure for the above sentence, we need to know the relation between both structures. As described earlier, the lexical items, “he” and “eats” delivers their feature description to corresponding POS, ‘prp’ and ‘vbz’. The ‘f1’ and ‘f2’ are the F-Structures of ‘prp’ and ‘vbz’, respectively. Further, the F-Structure building is moved towards root using the C-Structure within LFG as shown below.

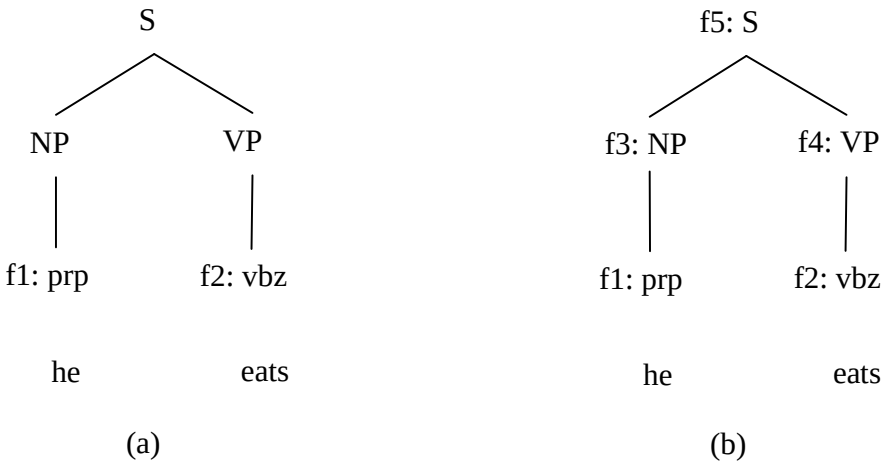


Figure 7 : Hierarchal naming of F-Structure

The Figure 7 (a) describes that the feature structures ‘f1’ and ‘f2’ are brought upward from the words. And 7 (b) shows that there is an F-Structure at each node. The ‘NP’ node receives the F-Structure from its child node ‘prp’ as shown in the rule (9.2), and ‘VP’ also receives F-Structure from ‘vbz’ using rule (9.3). ‘S’ receives two F-Structures from its children (‘NP’ and ‘VP’). The node is updated using the relation as described in rule (9.1). The F-Structure ‘f4’ is unified with ‘f5’ without any subsidiary relation marking. The structure of ‘f5’ will have an attribute named “SUBJ” containing the ‘f3’ as its value. Figure 8 illustrates the building of ‘f3’ and ‘f4’.

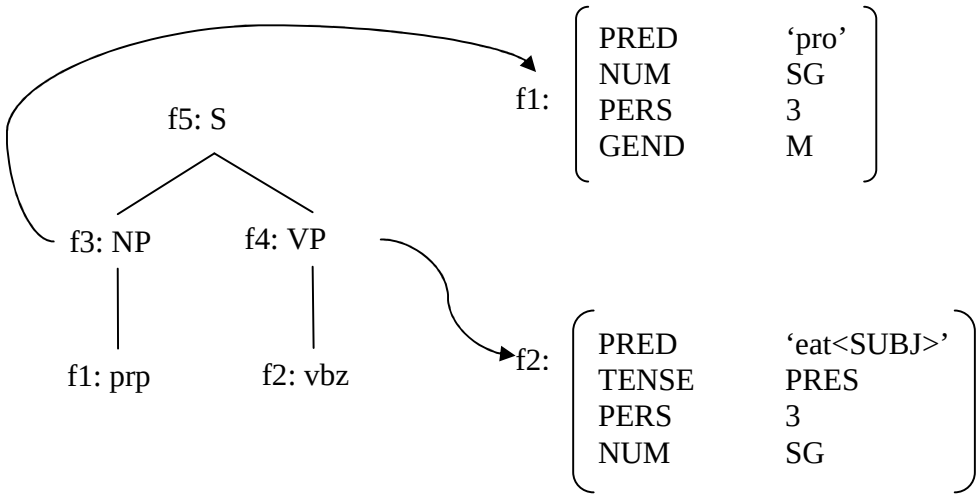


Figure 8 : Mapping F-Structure and tree nodes

Unification of ‘f3’ and ‘f4’ with empty ‘f5’ under rule (9.1) gives the F-Structure of sentence as shown in Figure 9.

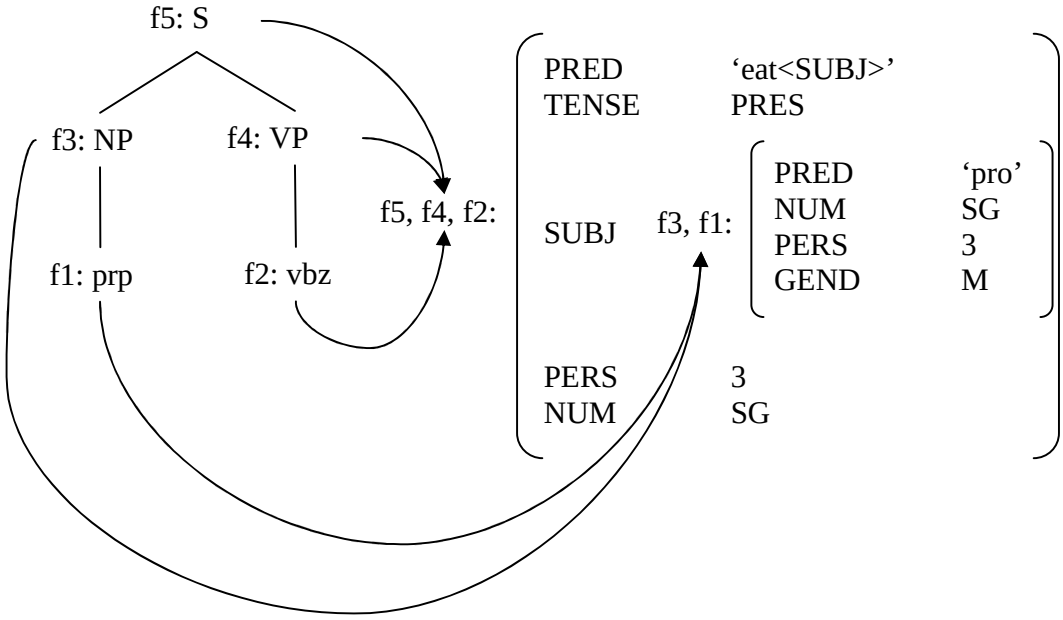


Figure 9 : Unified F-Structure mapped with tree nodes

The figure describes that ‘f1’ and ‘f3’ refer to the same F-Structure, as the LFG rule (9.2) requires. The LFG rule (9.3) requires the F-Structure ‘f4’ to be same as ‘f2’. LFG rule (9.1) requires ‘f5’ and ‘f4’ to be equal and hence with the associative property, ‘f2’ and ‘f5’ are equal. The reason for having rule (9.1) and ‘f5’ same as the ‘f2’ is the semantic form ‘eat<SUBJ>’ requiring an attribute ‘SUBJ’ to be assigned value. This attribute can be assigned value by using rule (9.1) only. In order to say our F-Structure a complete one, we have to give a value (atomic, semantic or subsidiary) to all the arguments of a semantic form and the attributes.

2.1.2.2.1 Constraints based parsing in LFG

Non-transformational theories of syntax are constraint-based [16]. They require satisfaction of static concurrent constraints to determine the grammaticality. These constraints add control on the generation of F-Structure in LFG. Satisfaction of constraint is also required, in addition to successful unification, for F-Structure building. Example 10 elaborates the construction of F-Structure with constraints.

Example 10:

LFG rules of Example 9 are amended to include constraints in the grammar such that ‘person’ and ‘number’ features of verb and its subject must match.

(10.1) S → NP: ↑ SUBJ = ↓, ↑ PERS =C ↓ PERS ; VP: ↑ = ↓,
↑ NUM =C ↑ SUBJ NUM;.

(10.2) NP → prp: ↑ = ↓; .

(10.3) VP → vbz : ↑ = ↓ ; .

The constraint is satisfied if the value of ‘person’ from (10.2) and (10.3) is same i.e. “sleeps” is the type of verb allowed with third person singular subject only. Graphical representation of the tree is described in Figure 10:

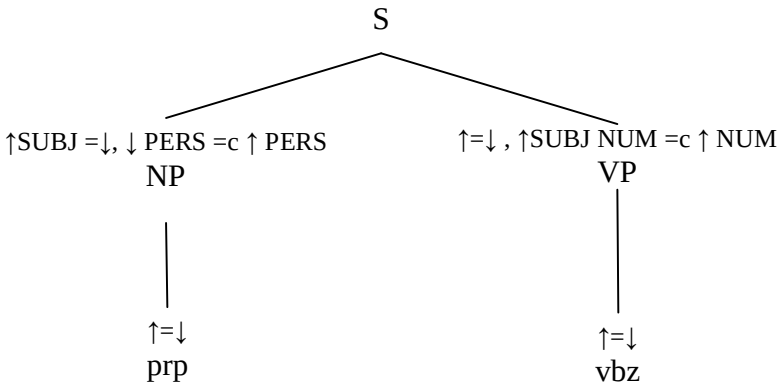


Figure 10 : F-Description annotated Parse tree

The above feature description annotated C-Structure describes the F-Structure building along with its constraints. Constraint succeeds: (1) if one argument is a subset of the other, in case both arguments are non-null or (2) if both arguments are null. It fails: (1) if any one of the arguments is null or (2) if one argument cannot be unified with other, in case both arguments are non-null.

2.1.2.3 Descriptive representation of F-Structure

A structure can be used to generate another form of structure, a descriptive, declarative or model based representation [12]. In order to represent the graphical representation of F-Structure as descriptive form, we first define the LFG's parenthetic notation [12] as:

$$(f\ a) = v \text{ iff } \langle a\ v \rangle \in f$$

Here 'f' is an F-Structure, 'a' is an attribute and 'v' is a value (atomic, semantic or subsidiary F-Structure). The parenthetic notation implies that an attribute 'a' in F-Structure 'f' has the value 'v' if any only if the attribute value pair '<a v>' is a member of 'f'.

Example 11:

$$f1:\begin{bmatrix} p & x \\ q & f2:\begin{bmatrix} s & t \\ u & v \end{bmatrix} \end{bmatrix}$$

The descriptive form of above F-Structure is as follows.

```
(f1 p) = x                (f1 q) = f2

(f2 s) = t                (f2 u) = v
```

2.1.2.4 F-Structure Properties

The sentences are parsed through LFG to result a final F-Structure. These F-Structures must hold the following three properties to fulfill the well formed-ness [10].

2.1.2.4.1 Completeness:

An F-Structure is locally complete if and only if it contains all the governable grammatical functions that it's predicate governs and an F-Structure is complete if and only if all of its subsidiary F-Structures are complete [13].

This condition requires that all the grammatical functions for which the predicate subcategorizes must have a value. The completeness property of an F-Structure does not hold if any argument is missing. For instance, the clause, 'We like' does not hold the completeness attribute because the grammatical function OBJ of predicate 'like' is not assigned a value.

2.1.2.4.2 Coherence

An F-Structure is locally coherent if and only if all the governable grammatical functions it contains are governed by a local predicate and an F-Structure is coherent if and only if all its subsidiary F-Structures are coherent [13].

Coherence requires every semantic form in the F-Structure to be assigned to a grammatical function i.e. every 'PRED' in the F-Structure should be direct or subsidiary part of the value of a grammatical function. For instance, "*he died the book*" being ill-formed because, *the book* can neither be associated as the object of the verb nor it can be

added as an adjunct in the relation to main verb. Hence, it cannot be assigned to any grammatical function and results a structure that does not hold the coherence property.

2.1.2.4.3 Consistency

In a given F-Structure, an attribute can have at most one value. However, there is a possibility that multiple values unify and build a set of values (which never violates the unification principles) [13]. For instance, in an F-Structure of English sentence, the TENSE feature cannot have values both PRESENT and PAST [13].

2.2 Analysis of Verbal Elements

The predicate is the element, containing information about the relationship in a sentence. [26]. *A grammatical unit containing one predicate and its participants is called a simple sentence or a clause* [18]. Verbal elements include predicates which require argument(s) for a clause to be grammatical [13]. The identification of a verb's sub-categorization frame (the grammatical functions) plays important role in the development of any natural language grammar. The following sections discuss different type of grammatical functions that are usually used in development of grammars; for instance LFG. These grammatical functions cover Subject (SUBJ), Object (OBJ), Secondary Object (OBJ2), Complementary clauses (COMP), open complementary clauses (XCOMP).

2.2.1 Subject (SUBJ)

It is assumed that all verbs subcategorize for subject, but some languages like German and Hindi challenge this assumption [13]. A noun phrase in the clause acts as a subject of clause. One of the identification rules for subject is the agreement with verb (or auxiliary verb). Properties of subjects vary from language to language [26]. The other clue can be the nominative case marking of the noun phrase. Different case markings can help in identification of subject in different languages. In German, nominative case marking helps in identifying subject [13]. The noun phrases as subject are highlighted in Example 12 below.

Example 12:

(12.1) **He** eats.

(12.2) **John** gave me a book.

(12.3) **Mary** can drive.

(12.4) **The worker union** protested.

(12.5) **Barking dogs** seldom bite.

(12.6) **Swimming** is fun.

(12.7) **Both of them** joined the board.

(12.8) A couple of years ago, **Ahmad** graduated.

(12.9) **The men, four of whom are ill**, were indicted for fraud.

The sentence (12.5) in Example 12 includes a gerund verb as a modifier of noun, but it is still included in the subject categorization. The subject in (12.6) is itself a gerund verb yet representing a noun phrase and hence a subject. Sentence (12.7) is an example of a subject with prepositional phrase. The sentence (12.9) includes another sentence clause within a subject frame.

There is another definition which is in general more understandable but not practical. The subject is the noun phrase that is a simulator, an initiator or an actor or sometimes subject experiences an action. The later case is most likely to occur in the case of passive voice. For instance in Example 12, sentence (12.2) shows an action performed by "*John*".

2.2.2 Object (OBJ)

Usually the second argument of transitive verbs is an object [13]. The object is usually recognized by its position. For instance, it appears following and adjacent to the verb in English. However, mostly in free order languages, like Germen, Hindi etc, the case

markers identify the object. For example, the following two sentences in German have the same meanings. The accusative case helps in identifying object in both sentences.

Example 13:

As described in [26].

(13.1)	Der	Fahrer	startet	den	Traktor.
	The.Nom	driver	starts	the.Acc	tractor
(13.2)	Den	Traktor	startet	der	Fahrer.
	the.Acc	tractor	starts	the.Nom	driver

(The driver is starting the tractor.)

The case marking test in English and French works only for pronouns such as he vs. him (object) and il vs. le (object French) [13].

Cross linguistically, passivization can be a good test to identify object. The noun phrases of subject and object are inverted in passivization such that, object become subject and vice versa. The active subject is realized as NULL in the passive sentence. The nullification is referred to argument suppression [13] An English example is represented in the following Example 14 with the semantic forms of main verb.

Example 14:

(14.1) He stole the money. (↑ PRED) = ‘steal<SUBJ, OBJ>’.

The money was stolen. (↑ PRED) = ‘steal<SUBJ >’.

(14.2) (Reproduced from [26])

He went home.

Home was gone.

In Example 14 (14.2), if a noun phrase is not object, it cannot be passivized correctly.

2.2.3 Secondary Object (OBJ2)

Ditransitive verbs subcategorize for three arguments as subject, object and secondary object [13], for instance, the verb 'give'. In English secondary objects can be identified by their position. It must be adjacent to and followed by the object (the primary object) [26].

In the sentence, "He gave me a pen", direct object is "me" and secondary object is "a pen". The secondary object in any language requires the existence of first object no matter what other test is used.

2.2.4 Oblique (OBL)

In English, the ditransitive sub-categorization frame for 'verbs of giving' alternates (the dative alternations) with a ditransitive frame whose third argument is an oblique [13]. Oblique class is difficult to define. They are the arguments other than subject and are not appropriate morph-syntactic form to be object. They also do not undergo the syntactic processes which affect object such as passivization in English. Generally, in English, Prepositional phrases stand as oblique. For example following sentence is represented with semantic form of verb. [13]

Example 15:

(15.1) She gave the pen to ahmad.

(↑ PRED) = 'give<SUBJ, OBJ, OBL>'.

(15.2) The pen was given to ahmad.

(↑ PRED) = 'give<SUBJ, OBL>'.

The sentence (15.1) in Example 15 shows the sentence with active voice and (15.2) is the passivization of (15.1). The object is removed but oblique persists.

2.2.5 Closed and Open Complementary Clauses

Arguments of a verb are not only noun or prepositional phrases. An entire clause may also be the compliment of a verb. Sometimes they may be replacing a noun phrase. [13]

Example 16:

(16.1) `Ahmad knows that Asif cheated.`

The verb ‘knows’ has a closed complementary clause ‘that Asif cheated’. In LFG, the closed complementary clauses have their own subject. As shown in example, there is a whole clause under sub-categorization of ‘know’ which we call as ‘COMP’.

There is a possibility that a complementary clause does not have its own subject rather its subject is functionally controlled from outside the clause.

(16.2) `Asif refused to come.`

The clause ‘to come’ is an open complimentary clause of ‘refused’ because it has a verb (the predicate) for the clause. This open complimentary clause is marked as XCOMP of ‘refused’. This implies that COMP has an explicit subject whereas XCOMP does not [13]. The F-Structures of the sentences (16.1) and (16.2) from Example 16 are shown in Figure 11 and Figure 12.

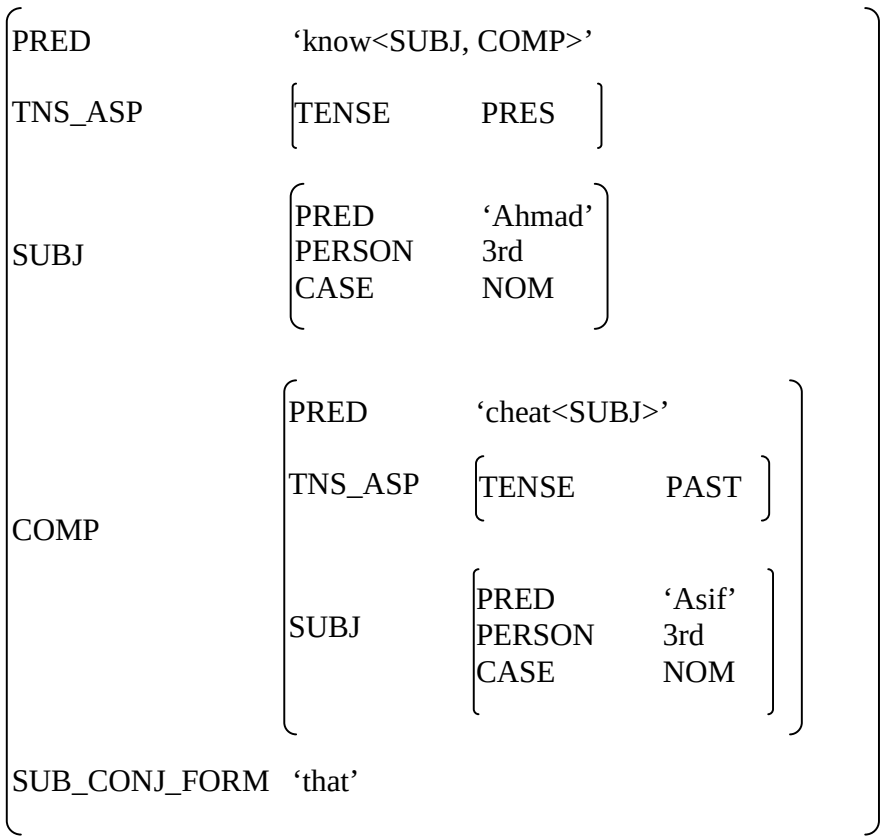


Figure 11 : F-Structur of 'Ahmad knows that Asif cheated'

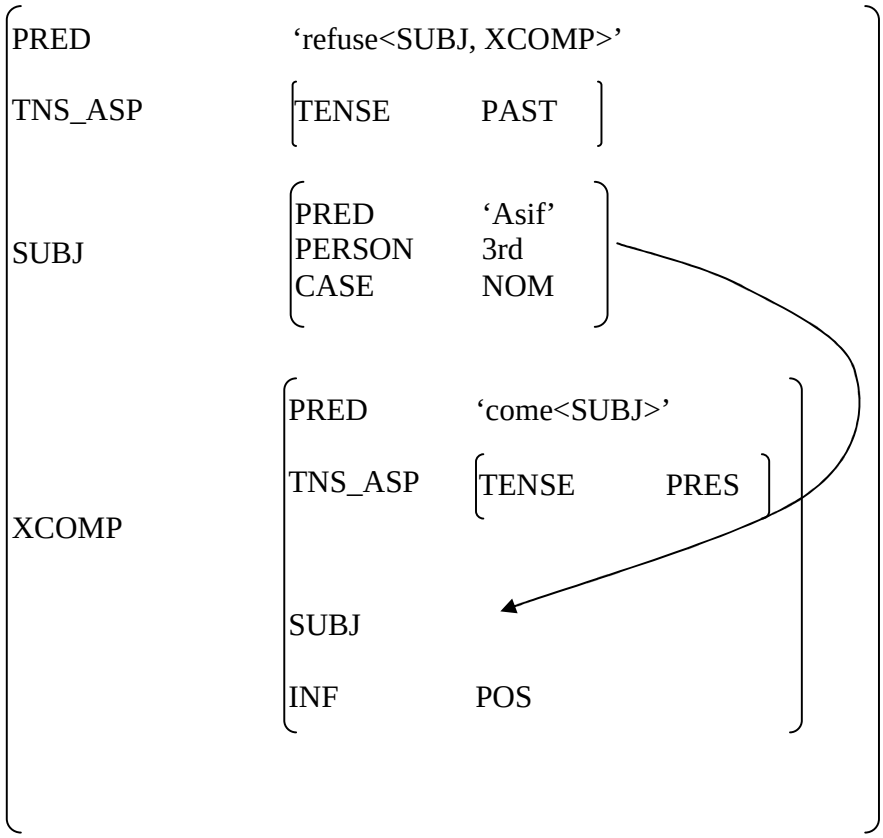


Figure 12 : F-Structure of 'Asif refused to come'

The link in the Figure 12 shows the subject sharing among parent and subsidiary F-Structures (in other words: clauses).

2.2.6 ADJUNCT

The grammatical functions ‘Adjunct’ is not subcategorized for by the verb [13]. They include a large number of different items for instance adverbs, prepositional phrases etc. These grammatical functions are analyzed as belonging to a set which can occur with any PRED [13]. Following examples shows a sentence with adjuncts.

Example 17:

(17.1) He did not come before me.
(↑ PRED) = ‘come<SUBJ>’

(17.2) I went to the city with my brother yesterday.
(↑ PRED) = ‘go<SUBJ>’

In (17.1), ‘did’ is the auxiliary verb adding only tense and aspectual information to main verb. ‘not’ and ‘before me’ are the adverbial and prepositional adjuncts respectively. In (17.2), ‘to the city’ and ‘with my brother’ are prepositional adjunct and ‘yesterday’ is adverbial adjunct.

2.2.7 Grammar development

There are 2 major reported types of grammars [2].

1. Hand-crafted grammars.

2. Automatically acquired grammars.

The type of a grammar can affect the level of abstraction needed for a grammar. For instance, a hand-crafted grammar can achieve more syntactic abstraction than automatically acquired grammar but with the increase in training corpus and increase in the size of developed grammar, maintenance issue becomes primal [2] [4]. *The development of large coverage, rich unification based grammar resources are not only time consuming and expensive but also requires considerable linguistic expertise* [13]. Small and medium sized grammars do not fulfill the requirement for a real world application and a large hand-crafted grammar is not easy to maintain.

A reasonable suggestion to avoid the problem of size of corpus and acquired grammar is to compact the grammar of a corpus [3] [4]. The compaction has been reported with quite a good reduction in size of grammar with gain in recall but decrease in precision [3]. The development of Lexical Functional Grammar for a natural language is still an issue. A solution to this problem is to automatically acquire the Context Free Grammar from the Treebank and manually annotate it with the feature description [5] [6] [19]. This solution is acceptable as far as there is a human involvement to manually annotate the grammar to build an LFG. However, it becomes impractical if a large Treebank is to be annotated

with feature description. *Some degree of automation in grammar development is unavoidable for any real world applications* [2].

2.3 Parsing Techniques

This section presents the architectures for the purpose of parsing and making F-Structure of a source sentence [1]. The two simple but useful techniques, pipeline and integrated model [1] are discussed in the following sub-sections with their potential pros and cons. Section 2.3.3 discusses the different issues with these models and in Section 2.3.4 we review the two major techniques to build Lexical Functional Grammar.

2.3.1 Pipeline model

In the pipeline model [1], first the PCFG (probabilistic context free grammar) is extracted from the un-annotated Treebank. Then, the input sentence is parsed so that we may have the most appropriate C-Structure according to PCFG we have extracted. The C-Structure is then annotated with feature description. Further, the annotated C-Structure is sent to constraint solver so that we may get an F-Structure in the end.

System flow diagram is illustrated in following figure 13.

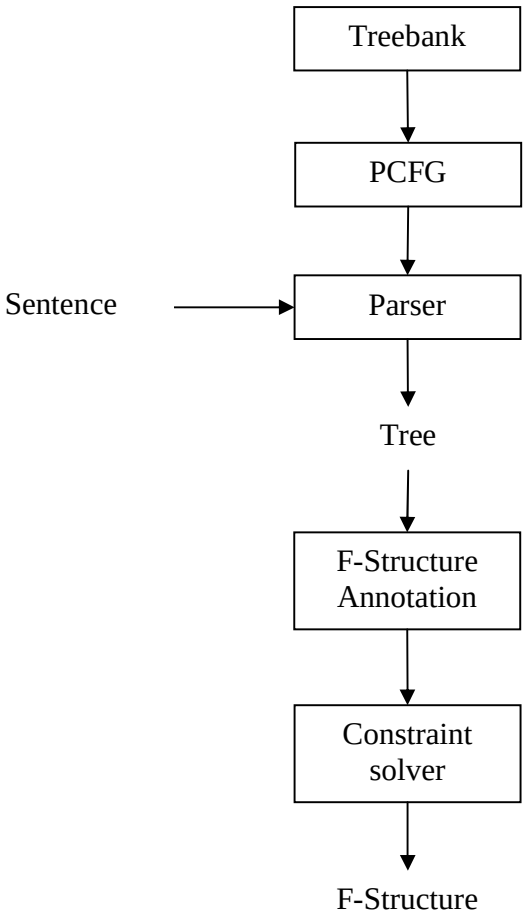


Figure 13 : System flow diagram of Pipeline model

There are two major phases in the pipeline architecture. The first phase includes extracting the PCFG from Treebank and parsing the input text according to PCFG. The accuracy is dependant on the size and coverage of Treebank. We have tree(s) as input to the second phase which is F-Structure annotation. The annotation process can be either manual or automated. The feature description annotated tree is sent to constraint solver. Constraint solver resolves all possible F-Structures from leaf to root node and chooses one of them.

2.3.2 Integrated model

In the integrated model, we first annotate the whole corpus (Treebank) with feature description. From annotated corpus, we extract ‘Annotated PCFG’ (APCFG). The

APCFG is somewhat like probabilities assigned LFG with constituent structure of real examples. Then the input sentences are parsed to give the annotated tree. The f-description annotated tree is further sent to constraint solver to generate the final F-Structure of the input sentence.

The architecture is as following figure 14.

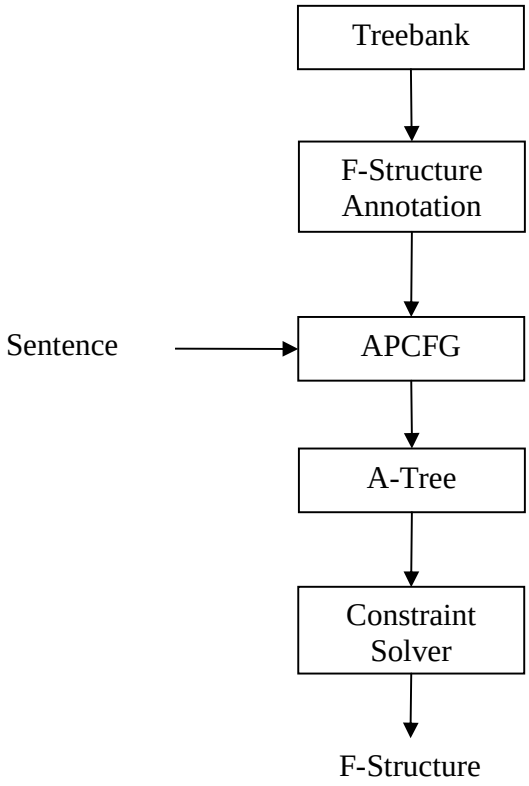


Figure 14 : System flow diagram of Integrated model

The major difference between pipeline and integrated model is the corpus annotation. This annotation process is the same as in pipeline model and hence either manual or automatic. From the annotated PCFG, we parse the sentence(s) and get an annotated tree. This annotated tree is further sent to constraint solver which results a single F-Structure randomly among all possible F-Structures.

2.3.3 Comparison of both models

Probabilistic parser and tree annotation are the two major systems needed in both models. In pipeline model, parser is trained on un-annotated trees. As name shows, integrated model uses the integration of parsed trees from corpus and the feature description. In integrated model, the probabilistic parser is actually influenced by the feature and functional description that becomes the part of training data. This positive aspect may also put expectation off the track for the results of parser because the parser may be distressed by the extra annotation it has. However, in the pipeline model, the parser is relatively trained only for the grammar it finds in corpus. We find more probability that pipeline model may give a more relevant C-Structure for the testing sentence than integrated may do. However, in the tree annotation phase, we may have more chances of mistakes than integrated model. There is some trade off between both models for sequencing the parser and annotation. Any probabilistic parser can be used under conditions such as it should not be biased while training for different 'type' of corpora. [1] [11]

It is notable that manually annotating the C-Structure dramatically differentiates the systems. Pipeline model always needs a grammar expert to annotate the parsed tree. This can increase time cost for instance in batch processing. However, the integrated model uses one time cost to manually annotate the corpus and later it can be easily used for purposes like batch processing or online parsing. Also, annotation system can be used as *direct* or *indirect* approach [1]. The direct approach is where one can convert annotated corpus to F-Structure. On the other hand in indirect approach, one has to first annotate the corpus and later convert it into F-Structure.

2.3.4 Annotation of CFG

Annotation system holds the key to generate Lexical Functional Grammar from CFG. A very obvious way to make the annotation system is the manually building Lexical Functional Grammar. However, this section discusses the automatic Lexical Functional Grammar development and automatically annotating the C-Structure to eventually result as F-Structure.

There exists a couple of techniques [1] [20] [21] [7] to automatically develop the Lexical Functional Grammar.

2.3.4.1 Regular Expression based technique

The first technique is to make Meta rules [7] manually such that they can be used to annotate the context free grammar's rule. These rules are somewhat like regular expressions. We align the CFG rules with these Meta rules (or template[1] in literature as well) and annotate the CFG rules. Templates are formed as follows [7].

LHS > RHS @ Annotation

There are three components in a template. It has LHS (Left Hand Side) and RHS (Right Hand Side) (like CFG) and the third component is the 'Annotation' we want to perform for this LHS and RHS. LHS can be only a non-terminal whereas RHS may have combination of terminals and non-terminals. Consider the following example for the annotation process.

Example 18:

We have a CFG rule as:

(18.1) `S → NP VP`

Where LHS is 'S' and RHS has 'NP' and 'VP'. We write the Template as following

(18.2) `S > * NP * VP * @ [Annotation].`

Here, the symbol * (Kleene star) can be aligned to any symbol(s). Here 'any symbol(s)' implies that this can be replaced with a null, a singleton or more than one symbol. We perform the mapping of CFG with this template such that the matching symbols are mapped (for instance NP to NP and VP to VP in both rules) without overlapping i.e.

(18.3) `S → NP VP`

[1] Template is equal to Meta rules. In further discussions, we use word 'template' instead of Meta rules.

(18.4) `S > * VP * NP * @ [Annotation] .`

The above CFG and Template can never relate to each other and hence we cannot use mapping in this case.

On success of mapping, we can apply annotations. The 'Annotation' part in template describes the functional structure of Right Hand Symbols. We rewrite the template as:

(18.5) `S > * NP * VP * @ [S:↑===VP,VP:SUBJ===NP] .`

The annotation part of the above system describes two annotations. The above template rule describes that, VP has a relation ↑ = ↓ with her parent S and NP is the SUBJ of its parent VP. As VP copies itself to the S so it implies that NP is the SUBJ of S. Each annotation is written as the following rule describes.

(18.6) `[Parent symbol: Relation === Child symbol ]`

Now consider the following case:

(18.7) `S > * NP NP * VP * @ [S:↑===VP, VP:SUBJ===NP] .`

In the (18.7), we are unable to identify which NP is exactly the SUBJ of VP. To resolve the problem, we use specific symbols called variables and modify the template as follows.

(18.8)
```
S > * NP:n1 NP:n2 * VP:v1 *
    @ [S:↑===v1,v1:SUBJ===n2].
```

Each variable is representing a Right hand Symbol and is responsible to express the relation with its parent. The only additional task to be done is to resolve the relations. We perform it by appending the current relation with the parent relation to get the absolute relation with Left Hand Side. In the above template, 'n2' shows a relation with 'v1' but 'v1' already had a relation with 's'. To get the absolute relation of 'n2' we resolve the relation of its parent (which is '↑'). As a result we can concatenate the two strings '↑' and

‘SUBJ’ as ‘↑ SUBJ’. This process returns the relation to be annotated with symbol of ‘n1’. We resolve relations of all symbols and use the mapping to annotate on CFG and this annotation returns us the LFG.

From (18.5) we have modified template as:

(18.9) `S > * NP:n1 * VP:v1 * @ [S:↑===v1, v1:SUBJ===n2] .`

From above mentioned process, we resolve the relations and map it onto (18.1) and get the CFG rule (18.1) annotated and restructured into LFG as following

(18.10) `S →  NP:↑SUBJ=↓ ;  VP:↑=↓; .`

2.3.4.2 Flat Tree based technique

A comparison paper [20] [21] of the above technique presents even more generalization. The basic idea is to describe tree in descriptive form (flat set representation). Then the templates are made in the same flat set representation and hence, those templates are applied to the trees.

The method is more general because it can consider arbitrary tree fragments instead of covering local CFG rule. The other reason is that these templates can be order-dependant and order-independent unlike regular expression based technique where order does matter [20] [21] [1].

Considering the issues listed above, we show the annotation as following.

Example 19:

We use Example 16 and rewrite the CFG rule (16.1) of Example 16 as following. The s1, n1 and v1 are the variable to refer S, NP and VP respectively.

Tree description:

```
dom (s1, n1)

dom (s1, v1)
```

```
pre (n1, v1)

cat (s1, S)

cat (v1,VP)

cat (n1, NP)
```

Template Description:

```
dom (X, Y), dom (X, Z), pre (Y, Z), cat (X, S),
cat (Y,VP), cat (Z, NP)
```

Implies

```
SUBJ(X, Y) , eq (X, Z).
```

‘dom(1, 2)’ implies that first argument dominates the second one. ’pre(1, 2)’ means that first argument occurs before second in the CFG rule or first is on the left of the second in a tree. ‘cat(1, 2)‘ represents that category of first argument is shown in the second argument. ‘eq(1, 2)’ means that first argument is equal in feature description to the second (or ↑ = ↓ in LFG notation). ‘SUBJ(1, 2)’ shows that subject of first argument is the second argument.

Here in the example, the tree description shows that S dominates NP and VP and NP is on the left of VP. In Template description, if we have an S dominating NP and VP and NP occurs before VP then VP equals S and NP is the subject of S.

By just removing ‘pre (Y, Z)’ condition in the Template, we can make the template order independent [20] [21].

3 PROBLEM STATEMENT

So far we have discussed about the constructs of predicate argument structures, Lexical Functional Grammar, parsing with this grammar and a couple of parsing schemes. Following discussion focuses the core purpose of this thesis.

The problem so far have been seen is the development of Lexical Functional Grammar (Section 2.2.7). The manual process of grammar development takes too long even for a grammar expert [13]. Hence, there is need to automate the process of lexical functional grammar building to get resultant F-Structure. Following is the problem statement of this thesis.

> "To build Annotation System that can convert a Context Free Grammar to Lexical Functional Grammar for English language."

The focus of the system is to take input a C-Structure and result a Lexical Functional Grammar which can be used to build the F-Structure. The annotation process is obviously abstract such that it can be used in any of the pipeline or integrated model.

3.1 Motivation

Machine Translation system[1] has been built to translate English sentences into Urdu using the F-Structure correspondence mechanism. A sentence from English language is parsed using pre-defined CFG rules and similar kind of LFG is used to build F-Structure. This F-Structure is further transformed into corresponding Urdu F-Structure and from that Urdu F-Structure, Urdu sentence is generated. Both English CFG and LFG are hand written and ambiguous. These ambiguous grammars can generate many parses. The time complexity becomes exponential and thus the space complexity to store all possible parses also becomes exponential.

[1] See www.crulp.org for Machine Translation System.

In order to prevent the problem, the grammar should be very specific to the test sentence. If there is a statistical parser used to generate that CFG, the time and space complexities can be significantly reduced. The new design of MT uses Collins' statistical parser [22] [23] to parse the test sentence and generate the CFG that is very specific to parse the given sentence. From this point on, the proposed annotation system generates the LFG corresponding to CFG. As a result, both grammars remain un-ambiguous. The new MT then re-parses sentence using the new CFG and LFG to generate F-Structure with no more exponential space and time complexity.

3.2 Scope

The thesis covers two major phrases, the Template development and Mapping System Development. Template development is performed manually followed by the Annotation system development to automate the process of sentence specific LFG generation.

3.2.1 Template Development

Template development is the part of linguistic and computational analysis in thesis. The templates will be developed manually. Input data for this phase will be 100 parsed trees from Penn Treebank and 200 parsed trees of English news (from BBC and CNN).

3.2.2 Annotation System Development

The Annotation system will be developed to use the templates, map them to CFG of a test sentence and finally result an LFG. System will be tested over 105 sentences of English. Sources of testing sentences will be same as that of training sentences.

3.3 Methodology

The methodology we have adopted for the purpose of automatically generating Lexical Functional Grammar from C-Structure is based upon the technique described in Section 2.3.4.1 . However, there has been slight modification.

The architecture of the Machine Translation system is given in Figure 15.

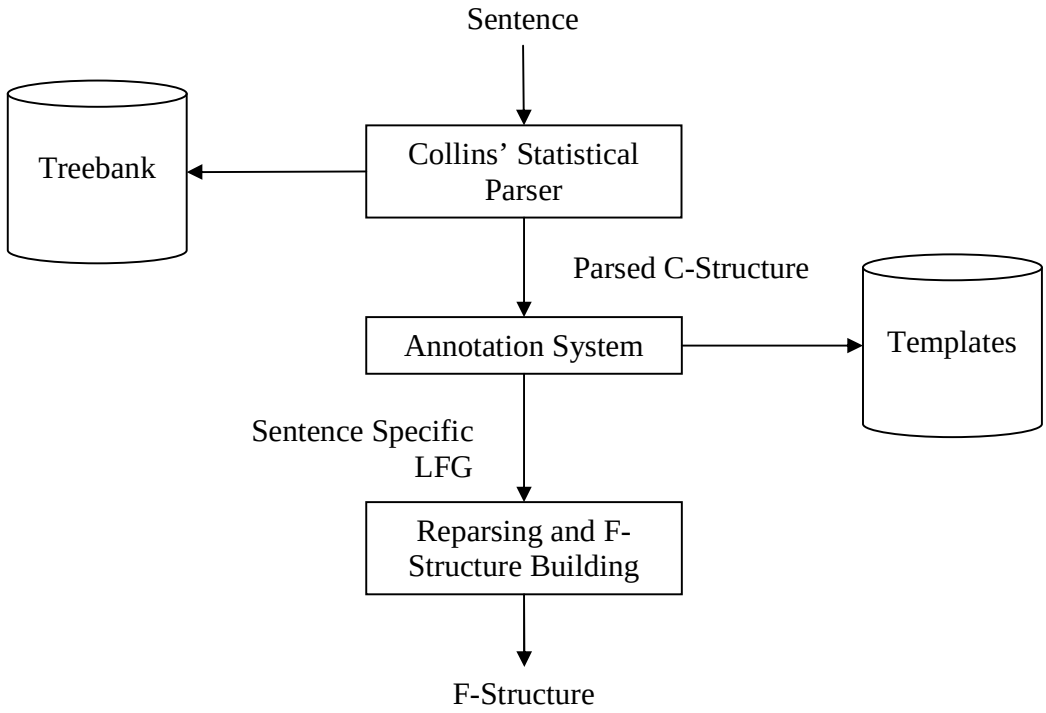


Figure 15 : Machine Translation System architecture

We are using the pipeline model for the system architecture as described in Section 2.3.1 and Collins' parser [22] [23] for the purpose of parsing and obtaining C-Structure from an English sentence. The parsed structure is then passed on for F-Structure building which uses the annotated grammar to reparse the sentence and build F-Structure using constraint solver. Our focus is to produce Lexical Functional Grammar so we are using the third party parser (Collins' Parser) to generate the C-Structure. We have F-Structure building system which uses the LFG generated by annotation system. The purpose of the system is to reparse the sentence using the Lexical Functional Grammar as shown in Section 2.1.2.2 .

As mentioned above that the annotation system is the main objective of this thesis, so we look at the proposed system diagram [24] within annotation system.

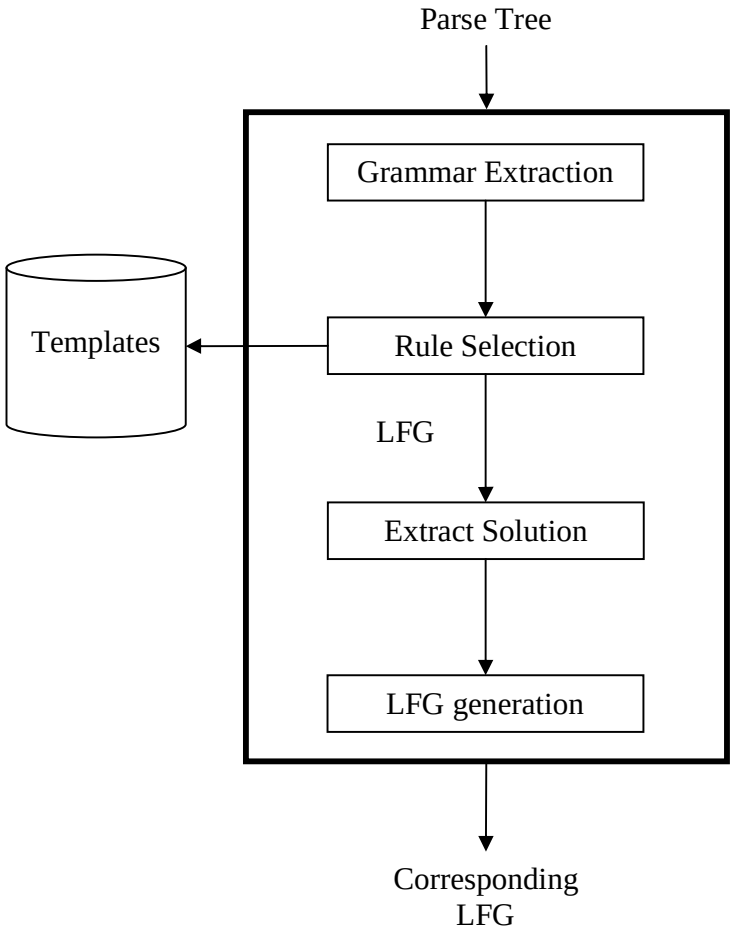


Figure 16 : Proposed architecture of Annotation System

In the diagram, the outer thick box shows the of Annotation system boundary. Input for the system is the most suitable parse tree and the output of the system is the Lexical Functional Grammar. The output contains the grammar that can parse only the under-process sentence i.e. should be unique and unambiguous (at-least no constituent ambiguity). First of all, the 'Templates' are being built manually. The description of the sub-systems is as follows.

3.3.1 Grammar Extraction

In first phase we discuss the grammar extraction [4] from the parsed tree. The process takes the normalized output of Collins' parser [22] as the input and extracts the CFG rules from that parse tree. Following example shows a simple tree and its extracted grammar.

Example 20:

The C-Structure of '`he ate apples with me`' is following.

```
(S
    (NP-A
        (NPB he/PRP ) )
    (VP  ate/VBD
        (NP-A
            (NPB apples/NNS ) )
            (PP  with/IN
                (NP-A
                    (NPB me/PRP ) ) ) ) )
```

Extracted Grammar:

(20.1) `S` → `NP-A` `VP`

(20.2) `NP-A` → `NPB`

(20.3) `NPB` → `prp`

(20.4) `VP` → `vbd` `NP-A` `PP`

(20.5) `NP-A` → `NPB`

(20.6) NPB → nns

(20.7) PP → in NP-A

(20.8) NP-A → NPB

(20.9) NPB → prp

In a simple sentence like shown above, we have three ambiguous rules i.e. (20.2), (20.5) and (20.8). Similarly, in longer sentence we may even have ambiguity in larger sub structures while looking only at the CFG. In order to tackle this problem, we add a number on every non-terminal such that the C-Structure can uniquely re-parse the sentence given the grammar. Following is the conversion we can perform.

(20.10) S-0 → NP-A-1 VP-2

(20.2) NP-A-1 → NPB-3

(20.3) NPB-3 → prp

(20.4) VP2 → vbd NP-A-4 PP-5

(20.5) NP-A-4 → NPB-6

(20.6) NPB-6 → nns

(20.7) PP-5 → in NP-A-7

(20.8) NP-A-7 → NPB-8

(20.9) NPB-8 → prp

3.3.2 Rule Selection

This sub-system performs the selection of most appropriate Meta rules that can annotate the under-process CFG rule. The algorithm runs as given below.

```
For each CFG Rule
      For each Template such that LHS of template = LHS of CFG
      rule
            If (template's RHS matches CFG rule's RHS)
                  Add template in the list attached with CFG
                  rule.
```

The rule selection uses the manually built templates and runs the above algorithm. All the CFG rules undergo the rule selection process. This system eventually results with all the CFG rules annotated with all possible f-descriptions. This system is not responsible for consistency of F-Description. For example, we select the template (34.20) in Example 34:. The reason of this selection is the match of the symbols on Left Hand Side and a successful mapping of symbols on Right Hand Side. In0, we select template (35.36) instead of (35.35) to map with CFG rule (35.1) because of extra symbols we have in template (35.35) . In order to completely understand the process, we refer to the structured walk through in Example 34 and Example 35.

3.3.3 Extract Solution

The purpose of this system is to use the output of Rule Selection and make it consistent within a CFG rule [24]. For instance, a CFG rules is possibly annotated with f-description which comes from two templates and both templates are inconsistent with each other. This sub-system selects the template (or annotation) that annotates maximum number of symbols in a CFG rule.

There is also a possibility that a CFG rule is annotated by many templates. In such case, we perform the grouping and make groups of templates that do not clash with each other. The group with most coverage is then selected and is used to annotate that CFG rule. Consider the following Example 21 [24].

Example 21:

Assume a rule as:

S	→	WP	XP	YP	ZP

Consider the output of Rule selection as:

S	→	WP	XP	YP	ZP
(a)		A	B		
(b)			C	D	
(c)		A	C		E
(d)		A		D	E

Here, the (a), (b), (c) and (d) are the template numbering (manually built templates) and ‘A’, ‘B’, ‘C’, ‘D’ and ‘E’ are the f-description annotations that a rule may add to the CFG rule.

The output of Rule selection shows the annotation for instance (a) marks ‘WP’ as ‘A’ and ‘XP’ as ‘B’. Now resolving the rule we can group (a) and (d) together as they do not clash with each other and similarly (b), (c) and (d) can also be grouped together as they are consistently annotating the CFG rule. We select the second group because it has the most coverage in this scenario. If more than one group can annotate with same coverage, we select the resultant group arbitrarily. Although, this selection may or may not be linguistically correct.

3.3.4 LFG Generation

LFG generation outputs the formal LFG syntax based on the rules resolved by ‘Extract Solution’. By reparsing the input sentence from this generated LFG, we get the corresponding F-Structure. The LFG generated by this sub-system is specific to the input sentence. LFG parser requires completely annotated rules to build F-Structure whereas it is possible in the annotation process mentioned in sections above that there are still some

symbols left un-annotated in the CFG rules. As mentioned in Section 2.2.6 , ADJUNCT are the verbal attributes not classified in sub-categorization of a predicate. Therefore, we annotate the un-annotated symbols as ADJUNCT.

The LFG generated by this sub-system is still not deterministic (see Example 34:, CFG rule (34.13) and (34.17)) and can generate multiple C-Structures in reparsing phase. In order to make it deterministic, we add a unique number (as described in Section 3.3.1) on each non-terminal symbol [24] as shown in Example 34 and Example 35. This addition makes the grammar deterministic and unambiguous. Example 34 and 35 describes this process in details.

4 ANALYSIS

This section covers different aspects appeared while developing the templates. The analysis is based on the linguistic as well as computational concerns. We also discuss the additions and variations we made from the original proposed technique [7].

4.1 Template Syntax Variation

There are certain variations from original model in Section 2.3.4.1 [7]. The existing regular expression scheme is supposed to annotate the grammatical functions but cannot add any other lexicalized feature descriptions. So the addition is made to original model [7] such that there are two annotation parts. The syntax is supposed to follow the expression given below.

```
LHS > RHS @ [Annotation] (@ [F-descriptions & constraints]).
```

The second part after RHS (F-descriptions and constraints) is to add the additional features descriptions or to apply some constraints. These constraints are evaluated later in the constraint solver. The annotation process only adds them while generating an LFG rule. Moreover, this part is an optional part of a template. As an example, see the following rule.

```
VP:vp > * VBD:v1 * ADVP:a1 *
      @ [v1:ADJUNCT ADV$===a1, vp:↑===v1]
      @ [v1:↑ INF = NEG].
```

This template describes that the ADVP is the ADJUNCT ADV of main verb and the main verb has an attribute-value pair as INF = NEG.

We have added a relation symbol ‘_’ which can be used in annotation part of a template. The purpose of this symbol is to only define the parent and child relation and leaving the type of relation to ‘unknown’. This ‘unknown’ implies that there are some computation

left to resolve this relation for instance in constraint solver. To understand consider the following template.

```
VP:vp > * VBD:v1 * S-A:s1 *
      @ [vp:_===s1,vp:↑===v1]
      @ [ s1:[      [↑XADJUNCT=↓ & ↓INF =c NEG]  ]
                    ||
                    [↑=↓ & ↓XCOMP INF =c POS ] ] ] .
```

The above template describes that, initially, ‘s1’ has its parent ‘vp’ and same does ‘v1’. However, ‘v1’ defines its relation with parent as ↑ = ↓ but ‘s1’ does not.

In the 2nd annotation part, ‘s1’ describes that it has two types of relation possible. As mentioned in the previous sections, the constraint solver resolves which path can be followed on basis of unification. The relation can be either ‘↑ XADJUNCT =↓‘ or ‘↑ = ↓’ depending upon the success and failure of respective constraint. The symbol ‘&’ forces all of the conditions and attributes to satisfy and unify.

4.2 Verbal Analysis

We discuss here the grammatical functions observed while developing the templates.

4.2.1 Subject (SUBJ)

The sentence marking (clausal level) non-terminals includes ‘S’, ‘S-A’ etc in Penn Treebank tagging guide [34]. Usually, subject appears on the sentence level discrimination between noun phrase and a verb phrase such that the noun phrase completes itself before the start of verb phrase.

For instance, consider the following template.

```
S:s > * NP-A:n1 * VP:v1 *
    @ [v1:SUBJ===n1,s:↑===v1] .
```

This template describes that in the domination of “S”, we have a noun phrase (with argument marked) before a verb phrase. If this is the case, we can mark it as subject of the verb phrase. There is a possibility that two noun phrases occur before the verb phrase and one of them is the subject and the other is a modifier of the subject. Following Example 22 shows the sentence level subject modifier.

Example 22:

PRETORIA, South Africa (AP) Adam Gilchrist hit the fastest half-century…

As we can clearly see, that ‘PRETORIA’ cannot be classified as the subject of the verb ‘hit’. The C-Structures of sentence is as following.

```
(S
     (NP  PRETORIA)
     (NP-A     South Africa (AP) Adam Gilchrist)
     (VP  hit the fastest half-century…)
)
```

The CFG rule for S is as follows.

```
S → NP    NP-A VP
```

We have template for this observation [25] as follows.

```
S:s > * [NP:n1|NP-A:n1] * NP-A:n2 * VP:v1 *
      @ [n2:ADJUNCT MOD$===n1,s:↑ SUBJ===n2,s:↑===v1] .
```

The template annotates as: the first noun phrase ‘n1’ (whether it is NP or NP-A) is the adjunct modifier of subject ‘n2’. The ‘n2’ (which is already marked as the verbal argument and is closer to verb phrase at sentence level) is the subject of the verb. Hence, the template annotates CFG rule and results as follows.

```
S → NP: ↑ SUBJ ADJUNCT MOD  = ↓ ; NP-A:↑ SUBJ =↓ ;
    VP: ↑ =↓;
```

Similarly, in SBAR, SBARQ etc (subordinate clausal level non-terminal) can have a ‘WH’ word as subject. The template rule is shown as;

```
SBAR,SBARQ,SQ:sbar > * WHNP:w1 * [SG-A:s1|SQ:s1|S-A:s1]
                    * @ [s1:SUBJ===w1,sbar:↑===s1] .
```

Consider the following Example 23 that has a ‘WH’ word as subject.

Example 23:

```
Who knows about the CSD and its works?
```

The C-Structure is as follows.

```
(SQ
     (WHNP     Who)
     (SG-A     knows about the CSD and its works)
)
```

CFG rule for SQ is as follows.

```
SQ → WHNP SG-A
```

And it is annotated as given below.

```
SQ → WHNP: ↑ SUBJ =↓ ; SG-A: ↑ =↓ ; .
```

Another form of sentences was observed. The subject can also move and be placed after the verb in an active declarative sentence [25].

Example 24:

"Ponting's team looks a good one, but it carries no aura," wrote the former England bowler Mike Selvey in the Guardian

The main verb ‘wrote’ in the above sentence has the subject ‘the former England bowler Mike Selvey’. This subject is not at the position where usually is occurs. In such case, Collins’ parser parses the above sentence as inverted sentence and marks as SINV. The parse structure is as follows.

```
(SINV
    (S   ‘Ponting’s team looks a good one, but it carries no aura’)
    (VP  wrote)
    (NP  the former England bowler Mike Selvey)
)
```

CFG rule for SINV is as follows.

```
SINV →    S    VP   NP
```

The template that can map the above CFG rule is as follows.

```
SINV:sinv > * S:s1 VP:v1 NP:n1 *
          @ [v1:COMP===s1,v1:SUBJ===n1,sinv:↑===v1]
```

And the resulting annotation and LFG rule is as follows.

```
SINV → S: ↑ COMP =↓ ; VP: ↑ =↓ ,
                          ↑ CLAUSE_TYPE = DECLARATIVE;
                     NP: ↑ SUBJ =↓ ; .
```

4.2.2 Object (OBJ)

The regular position of the second argument of a verb is after the verb itself. In Penn Treebank based parsing, the parent clause of object is the verb phrase itself. So a noun phrase within the verb phrase is likely to be object of the verb. Examples 13 and 14 describe the regular occurrence of object. It has been observed that the object phrase is adjacent to verb. No other phrase can occur between verb and its object e.g. any other noun, adjective, clausal or other phrase except the particle words. Consider the following example.

Example 25:

```
He blew up his tires.
```

The above sentence has a particle ‘up’ with main verb and is moving the object forward. The C-Structure is described as;

```
(S
     (NP  He)
     (VP  blew~vbd
          (PRT up)
          (NP  his tire)
     )
)
```

The ‘blew~vbd’ refers that the original tree has a terminal ‘vbd’ which is representing the word ‘blew’. The CFG rule for VP is as follows.

```
VP   →    vbd  PRT  NP
```

The above CFG rule describes that in the production, ‘vbd’ and ‘NP’ are adjacent except with a particle in between. The template rule needed to annotate such phrase is the following;

```
VP:vp > * [VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] (PRT:a1)
          NP:n1 * @ [v1:OBJ===n1,vp:↑===v1] .
```

Note that it is not mandatory that every phrase is annotated in a template. The template annotates only the symbol referred in its annotation part. In above template, PRT (the particle) is only mandatory for alignment, not for annotation.

4.2.3 Secondary Object (OBJ2)

As described earlier, the third functional argument a verb can have is the secondary object. The secondary object or the object2 occurs strictly after the first object without any intermediate phrase. The following example shows the case of secondary object.

Example 26:

```
He showed them the way
```

Collins’ parser parses the sentence as;

```
(S
     (NP  He)
     (VP  showed~vbd
          (NP-A     them)
          (NP  the way)
     )
)
```

CFG rule for VP is as follows.

```
VP   →    vbd  NP-A NP
```

The second noun phrase in the verb phrase is the secondary object. The template needed to annotate the above rule is;

```
VP:vp > * [VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] (PRT:a1)
              [NP-A:n1|NPB:n1|NP:n1]
              [NBP:n2|NP:n2|NP-A:n2] *
        @ [v1:OBJ===n1,v1:OBJ2===n2,vp:↑===v1] .
```

We can have a particle between verb and first object, but we cannot have any phrase between object and secondary object.

4.2.4 Oblique (OBL)

We have observed that class oblique can be merged with the class adjunct. There are two reasons for this merge. The primary reason is inability to identify the oblique part i.e. we are unable to differentiate between the constituent structure of adjunct and oblique.

Example 27:

(27.a) `She gave the pen to ahmad.`

The C-Structure is as follows.

```
(S
  (NP-A
    (NPB she/PRP ) )
  (VP gave/VBD
    (NP-A
      (NPB the/DT pen/NN ) )
    (PP to/TO
```

```
        (NP-A
          (NPB Ahmed/NNP )))))
```

The CFG rules extracted from above C-Structure are as follows.

(27.1) S → NP-A VP

(27.2) NP-A → NPB

(27.3) NPB → PRP

(27.4) VP → VBD NP-A PP

(27.5) NP-A → NPB

(27.6) NPB → DT NN

(27.7) PP → IN NP-A

(27.8) NP-A → NPB

(27.9) NPB → NNP

We state sentence (27.b) and compare it with sentence (27.a).

(27.b) She saw a pen on table.

The C-Structure is as follows.

```
    (S
      (NP-A
        (NPB she/PRP ) )
      (VP saw/VBD
        (NP-A
```

```
    (NPB a/DT pen/NN ) )
  (PP on/IN
    (NP-A
      (NPB table/NN )))))
```

The C-Structure is as follows.

(27.10) S → NP-A VP

(27.11) NP-A → NPB

(27.12) NPB → PRP

(27.13) VP → VBD NP-A PP

(27.14) NP-A → NPB

(27.15) NPB → DT NN

(27.16) PP → IN NP-A

(27.17) NP-A → NPB

(27.18) NPB → NNP

In the sentence (27.a) of above example ‘to ahmed’ is parsed under PP which is a prepositional phrase. This phrase is more likely to be marked oblique (see Example 15). However, in the sentence (27.b) of Example 27, ‘on table’ is also parsed under PP. The PP in rule (27.4) is of oblique form whereas in rule (27.13) it is some other adjunct. Clearly, we can see no difference between the CFG rules (27.4) and (27.13) and hence we cannot differentiate the oblique and adjuncts. This problem leads us to merge the two verbal classes; oblique and adjuncts.

4.2.5 Closed Complementary Clause (COMP)

The complementary class can be sub-divided into two classes, one with subject and other without (or shared) subject. The class with subject (or closed clause) is represented here as COMP.

Example 28:

We quote Example 16.

```
Ahmad knows that Asif cheated.
```

The C-Structure is as follows;

```
(S
  (NP-A (NPB ahmad/RB ) )
  (VP knows/VBZ
    (SBAR-A     that/IN
      (S-A
        (NP-A
          (NPB  asif/IN ) )
        (VP cheated/VBN ) ) ) ) ) )
```

We have the CFG rule for above verb phrase as

```
VP   →    vbz  SBAR-A
```

The sentence clause under a verb phrase is marked as complimentary clause of the verb [25]. The template written for this annotation is as below.

```
VP:vp > * [VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] *
          [S:s1|SBAR-A:s1|SBAR:s1] *
        @ [v1:COMP$===s1,vp:↑===v1].
```

The symbols in Right Hand Side of the above template are arranged in two groups. First is the disjunction of verb POS symbols. The second set is the disjunction of symbols used to dominate the sentence clause. These subordinate clauses are marked as complementary clauses of the main verb represented by ‘v1’.

Example 29:

```
Ahmad came before I could leave.
```

The C-Structure is as follows.

```
(S
   (NP     Ahmad/NNP)
   (VP     came/VBD
       (SBAR-A  before/IN
           (S-A
              (NP I)
               (VP could
                   (VP leave))))))
```

The ‘SBAR-A’ in above C-Structure is similar to ‘SBAR-A’ in previous example. Note that the word ‘that’ and ‘because’ is tagged with same POS in both examples (Example 28 and Example 29). The role of prepositional element ‘before’ and ‘that’ is to show the subordinate element. For instance, consider the following template.

```
SBAR-A:sbar > * IN:i1 * [S-A:s1|S:s1] *
            @ [sbar:↑===s1,
               sbar:↑CONJ_FORM===i1:CONJ_FORM].
```

The template describes that the sentence clause is unified with its parent ‘SBAR’ without any subsidiary annotation i.e. ‘↑ = ↓’ and ‘IN’ only adds the ‘CONJ_FORM’ to its parent structure.

4.2.6 Open Complementary Clause (XCOMP)

XCOMP is the class where the subject of clause is functionally controlled outside this clause. Hence, this is a subordinate clause and in our observation the verb only occurs in its nonfinite form.

Example 30:

Using Example 16.

```
Asif refused to come.
```

The C-Structure is as follows;

```
(S
     (NP  Asif)
     (VP  refused~vbd
          (SG-A          to come)
     )
)
```

The CFG rule is:

```
VP   →    vbd  SG-A
```

The XCOMP shares object with the parent clause. This is because semantically, the subject of both verbs is the same [13] [25]. The template is as follows.

```
VP:vp > * VBD:v1 * SG-A:s1 *
      @ [vp:_===s1,vp:↑===v1]
      @ [ v1:↑ INF=NEG, s1:[
                        [↑=↓ & ↓XCOMP INF =c POS ]]
                        ||
                        [↑XADJUNCT=↓ & ↓INF =c NEG]]] .
```

As mentioned in Section 4.1, the constraint can only define which of the exclusive relations unify with rest of F-Structure in constraint solver. The SG-A either has '↑=↓' if there is 'XCOMP' already mentioned in 'SG-A' down the tree and attribute INF as POS or it is marked 'XADJUNCT' if at the current level of F-Structure building SG-A has attribute INF as NEG.

4.2.7 ADJUNCT

The ADJUNCT has five main sub-categorizations in our observation [13]. Though we have marked even more but they are rather specific and used in flatter rules.

4.2.7.1 ADJECTIVE

As mentioned above, the adjuncts do not play a vital role in a sentence and they are often the modifiers of a PRED. The adjectives as a modifier are marked within adjunct category. For instance adjective as a noun modifier is the ADJUNCT ADJ of a noun. For instance, the following template describes the relation.

```
NP:np > * [JJ:j1|JJR:j1] *
          [NN:n1|NNS:n1|NNP:n1|NNPS:n1]
      @ [np:↑ADJUNCT ADJ$===j1,np:↑$===n1] .
```

4.2.7.2 ADVERB

There are further two sub-types marked within adverbs.

1. The sentence level adverbs.

2. Any adverb other than the sentence level.

The sentence level adverbs are referred as following;

```
S:s > * ADVP:a1 * VP:v1 *
    @ [v1:ADJUNCT S_ADV$===a1,s:↑===v1] .
```

Any adverb which is parsed and marked at sentence level is ADJUNCT S_ADV.

In any other observation, we mark the adverbs as ADJUNCT ADV. Following templates are instances of the case 2.

```
ADJP:adjp > * RB:r1 * [JJ:j1|JJR:j1]
          @ [j1:ADJUNCT ADV$===r1,adjp:↑===j1] .

NP-A:np-a >  NPB:n1 ADVP:a1
          @ [n1:ADJUNCT ADV===a1,np-a:↑===n1] .

PP:pp > RB:r1 * PP:p1 * @ [p1:ADJUNCT ADV$===r1] .
```

The first instance of template describes the adverbial occurrence in adjective phrase. The observation describes that the head in adjective phrase is the adjective itself. Similarly, the second instance exemplifies for adverb phrase in a noun phrase and third template refers an adverbial phrase within a prepositional phrase.

4.2.7.3 Prepositions

Prepositional phrases often add the temporal and spatial reference in a sentence and again are categorized as ADJUNCT. The prepositional phrases are also of two types.

```
S:s > * PP:p1 * VP:v1 *
    @ [v1:ADJUNCT S_PREP$===p1,s:↑===v1]
    @ [v1:↑CLAUSE_TYPE=DECLARATIVE] .
```

```
S-A:s-a > * PP:p1 * VP:v1 *
        @ [v1:ADJUNCT S_PREP$===p1,s-a:↑===v1]
        @ [v1:↑CLAUSE_TYPE=DECLARATIVE] .

NP:np > * PP:p1 *
      @ [np:↑ ADJUNCT PREP$===p1] .

ADJP:adjp > * JJ:j1 * PP:p1
          @ [j1:ADJUNCT PREP$===p1,adjp:↑===j1] .

VP:vp > * VB:v1 * PP:p1 * (PP:p2 *)
      @ [v1:ADJUNCT PREP$===p1,v1:ADJUNCT PREP$===p2,
         vp:↑===v1] .
```

The first two instances refer to a sentence level preposition whereas remaining tree templates are example of the case where a preposition is not marked as sentence level preposition.

4.2.7.4 Relative Clause

Relative clause is the modifier of a noun phrase. This subordinate clause is a complete sentence within a noun phrase and is exemplified as following.

Example 31:

The woman, who died earlier this week, was from Cameron County on the edge of the Gulf of Mexico.

C-Structure;

```
(S
(NP  (NPB The woman)
     (SBAR     who died earlier this week)
)
```

```
        (VP  was from Cameron County on the edge of the
Gulf of Mexico)

    )
```

The CFG rule is as follows.

```
    NP   →    NPB  SBAR
```

This CFG rule is annotated by the following template.

```
    NP:np > * NPB:n1 * SBAR:s1 *
          @ [n1:ADJUNCT REL_CL===s1,np:↑===n1] .
```

The template marks the ‘NPB’ as the head of the phrase and SBAR as the modifier of this phrase. The type of modifier is ADJUNCT with sub-classification of relative clause (REL_CL).

4.2.7.5 Participle

The participle is a non-finite verb and shares the subject with parent clause. Usually it is also considered as a modifier unlike XCOMP. As alone they can be considered as predicative words, but in a presence of a noun this class acts as a modifier. For instance, the following example has an ADJUNCT PARTICIPLE in it.

Example 32:

```
Gazing at the painting she recalled the house
```

The C-Structure for this sentence can be as following.

```
    (S
        (NP
            (SG  Gazing at the painting
            (NP  she)
```

```
        )
        (VP   recalled the house)
    )
```

Containing the CFG rule:

```
NP → SG   NP
```

This CFG rule can be annotated with the following template.

```
NP:np > * SG:s1 * NP:n1
      @ [n1:ADJUNCT PARTICIPLE===s1,np-a:↑===n1] .
```

4.3 Relaxing Constraints

As mentioned above (see Section 3.3.4), the LFG built is domain specific and can only be used to reparse the sentence under-process. In developing templates and generating LFG rules, our intention is to avoid multiple parses in C-Structure and reduce the search space for constraint solver. However, the proposed methodology based on [7] (see section 2.3.4.1) describes the annotation within the scope of a single CFG rule. Therefore, it lacks the knowledge of constituent structure beyond the current rule in the hierarchy. For example, the CFG rule (33.1) in Example 33: describes that looking at only a single rule does not completely define the relation (SUBJ or OBJ). We need to add non-determinism in our functional description to provide the flexibility in deciding the relation at later stage. Later, with the help of constraint solver and the unification process, we decide which path is computable and can result an F-Structure.

We have introduced a binary operator '/' in the 'LFGAttibute' part of template syntax (described in Appendix B). The operator acts as a disjunction between its operands. To understand the need of this operator, we refer to the following example.

Example 33:

We have a sentence as;

```
What is your name?
```

The extract of C-Structure for the above sentence is as follows.

```
(SBARQ
     (WHNP          what)
     (SQ       is your name)
)
```

The CFG rule for SBARQ is as follows.

(33.1) SBARQ → WHNP SQ

It is too early to annotate the ‘WHNP’ (representative of the word ‘what’) by looking at the above CFG rule. The word ‘what’ can play a subject’s as well as an object’s role in a sentence [25]. Instead of making a wrong decision, we make two rules such that one annotates this ‘WHNP’ as subject and other as object. The template is written as follows.

```
SBARQ:sbar > * WHNP:w1 * SQ:s1 *
           @ [s1:SUBJ/OBJ===w1,sbar:↑===s1] .
```

This template annotates the CFG rule as follows.

```
SBARQ → WHNP: ↑ SUBJ =↓; SQ: ↑ =↓; .

SBARQ → WHNP: ↑ OBJ =↓; SQ: ↑ =↓; .
```

This annotation solves the problem and makes a non-deterministic path for the moment. The constraint solver solves the issue by making F-Structure from leaf nodes to root in a C-Structure. WHNP is selected depending upon the F-Structure we receive from SQ i.e. if there is already a subject in SQ, WHNP becomes OBJ otherwise SUBJ.

4.4 Structured Walk Through

The following discussion portrays the system flow using a dry run of corpus based examples.

Example 34:

`The report warns that inaction could push millions of people worldwide into unemployment.`

The C-Structure of the sentence is as follows.

```
(S
  (NP-A (NPB the/DT report/NN))
  (VP warns/VBZ
    (SBAR-A that/DT
      (S-A
        (NP-A (NPB inaction/NN))
        (VP could/MD
          (VP-A push/VB
            (NP-A (NPB millions/NNS)
              (PP of/IN
                (NP-A (NPB people/NNS)
                  (ADJP worldwide/JJ))))
            (PP into/IN
              (NP-A (NPB unemployment/NN)))))))))
```

This parse structure is the input of the annotation system described in Figure 15 *: Machine Translation System architecture* and Figure 16 *: Proposed architecture of Annotation System*. As a first step, we extract CFG from this input. The grammar is as follows.

(34.1) S → NP-A VP

(34.2) NP-A → NPB

(34.3) NPB → DT NN

(34.4) VP → VBZ SBAR-A

(34.5) SBAR-A → IN S-A

(34.6) S-A → NP-A VP

(34.7) NP-A → NPB

(34.8) NPB → NN

(34.9) VP → MD VP-A

(34.10) VP-A → VB NP-A PP

(34.11) NP-A → NPB PP

(34.12) NPB → NNS

(34.13) PP → IN NP-A

(34.14) NP-A → NPB ADJP

(34.15) NPB → NNS

(34.16) ADJP → JJ

(34.17) PP → IN NP-A

(34.18) NP-A → NPB

(34.19) NPB → NN

In order to get LFG for this CFG, we have to annotate it with feature and functional description. The next process is to select the appropriate templates which can annotate these CFG rules. In the rule selection, we ignore the CFG rules having only single right hand symbol (terminal or non-terminal). If there is only one term in the right hand side of a grammar rule, it can have only one relation with the left hand symbol (or parent symbol). This relation is '↑=↓' which implies that all the child's attributes are delivered to parent without any subsidiary change or semantic form. As a result, we exclude the CFG rules (34.2), (34.7), (34.8), (34.12), (34.15), (34.16), (34.18) and (34.19) listed above. We have to annotate rules (34.1), (34.3), (34.4), (34.5), (34.6), (34.9), (34.10), (34.11), (34.13), (34.14) and (34.17).

One grammar rule is checked against all templates to see if any of them can annotate the CFG rule. The algorithm of annotation is described in Section 3.3.2 . For instance, we annotate rule (34.17) using following template (see appendix C for complete list).

(34.20) PP:pp > * IN:i1 * [NPB:n1|NP-A:n1|ADJP-A:n1] *
@ [i1:OBJ===n1,pp:↑===i1] .

The template rule (34.20) shows that it can only be used for the CFG rules having left hand side as 'PP'. Furthermore, it should have an 'IN' symbol on right hand side followed by disjunction of 'NPB', 'NP-A' and 'ADJP-A' symbol. The rule selection is performed to make sure the correct alignment of symbols. The alignment of (34.17) and (34.20) is as follows.

PP → IN NP-A

PP:pp > * IN:i1 * [NPB:n1|NP-A:n1|ADJP-A:n1] *

We ignore the annotation part of a template in the Rule Selection. From the above alignment, we can also say that this template can even work if there is one or more

symbols between ‘IN’ and ‘NP-A’ in grammar rule (by definition of Kleene star [7]). In the next phase, we extract the solution by annotating CFG rule with corresponding aligned templates. From the CFG rule (34.17) and template(34.20) , we can annotate as follows.

```
PP   →    IN              NP-A

PP:pp > * IN:i1 * [NPB:n1|NP-A:n1|ADJP-A:n1] *
        @ [i1:OBJ===n1,pp:↑===i1] .
```

As described in Section 2.3.4.1 rule (34.6), (see appendix B also), the above annotation described that ‘IN’ (i1) has a relation ‘↑=↓’ with the parent ‘PP’ (pp) and ‘NP-A’ (n1) is subcategorized with relation ‘OBJ’ under ‘IN’. Using the alignment, we annotate CFG to result following LFG rule.

(34.21) `PP → IN: ↑ = ↓; NP-A: ↑ OBJ = ↓;`

Similarly, we annotate each CFG rule to give the following LFG for current sentence.

(34.22)
```
S → NP-A:↑ SUBJ = ↓ ; VP:↑ = ↓ ,
                        ↑ CLAUSE_TYPE=DECLARATIVE ; .
```

(34.23) `NP-A → NPB:↑ = ↓ ; .`

(34.24) `NPB→ DT:↑ SPEC DET = ↓ ; NN:↑ = ↓ ; .`

(34.25)
```
VP → VBZ:↑ = ↓ , ↑ INF = NEG ;
       SBAR-A:↑ COMP = ↓ ; .
```

(34.26)
```
SBAR-A → IN:↑CONJ_FORM=↓PHY_FORM;
           S-A:↑ = ↓; .
```

(34.27)
```
S-A → NP-A:↑ SUBJ = ↓ ; VP:↑ = ↓ ,
        ↑ CLAUSE_TYPE=DECLARATIVE ; .
```

(34.28) NP-A → NPB:↑ = ↓ ; .

(34.29) NPB → NN:↑ = ↓ ; .

(34.30) VP → MD:↑ = ↓ ; VP-A:↑ = ↓ ; .

(34.31) VP-A → VB:↑ = ↓ ; NP-A:↑ OBJ = ↓ ;
PP:↑ ADJUNCT PREP = ↓ ; .

(34.32) NP-A → NPB:↑ = ↓ ;
PP:↑ ADJUNCT PREP = ↓ ; .

(34.33) NPB → NNS:↑ = ↓ ; .

(34.34) PP → IN:↑ = ↓ ; NP-A:↑ OBJ = ↓ ; .

(34.35) NP-A → NPB:↑ = ↓ ;
ADJP:↑ ADJUNCT ADJ = ↓ ; .

(34.36) NPB → NNS:↑ = ↓ ; .

(34.37) ADJP → JJ:↑ = ↓ ; .

(34.38) PP → IN:↑ = ↓ ; NP-A:↑ OBJ = ↓ ; .

(34.39) NP-A → NPB:↑ = ↓ ; .

(34.40) NPB → NN:↑ = ↓ ; .

The rule (34.11) is annotated with following templates.

(34.41) NP-A:np > * NPB:n1 * PP:p1 *
@ [n1:ADJUNCT PREP$===p1,np:↑===n1] .

(34.42) NP-A:np > * PP:p1 *
@ [np:↑ ADJUNCT PREP$===p1] .

The rule (34.11) is annotated with two apparently different templates. However, the both templates are non-conflicting and provide the same information about annotation of ‘PP’. Thus, we get the annotation resulting as rule (34.32). The Lexical Functional Grammar listed above is still ambiguous. Rule (34.34) and (34.38) are equal and hence can be used alternatively, for instance in parsing the rule (34.32). To avoid this problem, we make the grammar un-ambiguous in ‘LFG generation’ step (Section 3.3.4) by adding unique numbers with each symbol as follows.

(34.43) S_28 → NP_A_45:↑ SUBJ = ↓ ; VP_100:↑ = ↓ ,
↑ CLAUSE_TYPE=DECLARATIVE ; .

(34.44) NP_A_45 → NPB_61:↑ = ↓ ; .

(34.45) NPB_61 → DT:↑ SPEC DET = ↓ ; NN:↑ = ↓ ; .

(34.46) VP_100 → VBZ:↑ = ↓ , ↑ INF = NEG ;
SBAR_A_129:↑ COMP = ↓ ; .

(34.47) SBAR_A_129 → IN:↑CONJ_FORM=↓PHY_FORM;
S_A_154:↑ = ↓; .

(34.48) S_A_154 → NP_A_173:↑ SUBJ = ↓ ; VP_223:↑ = ↓ ,
↑ CLAUSE_TYPE=DECLARATIVE ; .

(34.49) NP_A_173 → NPB_191:↑ = ↓ ; .

(34.50) NPB_191 → NN:↑ = ↓ ; .

(34.51) VP_223 → MD:↑ = ↓ ; VP_A_249:↑ = ↓ ; .

(34.52) VP_A_249 → VB:↑ = ↓ ; NP_A_278:↑ OBJ = ↓ ;
PP_436:↑ ADJUNCT PREP = ↓ ; .

(34.53) NP_A_278 → NPB_296:↑ = ↓ ;
PP_324:↑ ADJUNCT PREP = ↓ ; .

(34.54) NPB_296 → NNS:↑ = ↓ ; .

(34.55) PP_324 → IN:↑ = ↓ ; NP_A_349:↑ OBJ = ↓ ; .

(34.56) NP_A_349 → NPB_365:↑ = ↓ ;
ADJP_400:↑ ADJUNCT ADJ = ↓ ; .

(34.57) NPB_365 → NNS:↑ = ↓ ; .

(34.58) ADJP_400 → JJ:↑ = ↓ ; .

(34.59) PP_436 → IN:↑ = ↓ ; NP_A_469:↑ OBJ = ↓ ; .

(34.60) NP_A_469 → NPB_491:↑ = ↓ ; .

(34.61) NPB_491 → NN:↑ = ↓ ; .

All the non-terminal symbols in the grammar are annotated with a number that makes the parse structure unique using this grammar. For instance, rule (34.53) cannot use any other LFG rule but (34.55). The set of LFG rules (34.43) to (34.61) comprises the grammar we need to parse and make F-Structure of given sentence. This grammar is the input of 'Reparsing and F-Structure Building' module shown in Figure 15. This module is responsible for re-parsing and building F-Structure of the sentence.

Follows is the F-Structure generated by the Functional Mapper System[1] using the above LFG.

[1] The Machine Translation System available at www.crulp.org

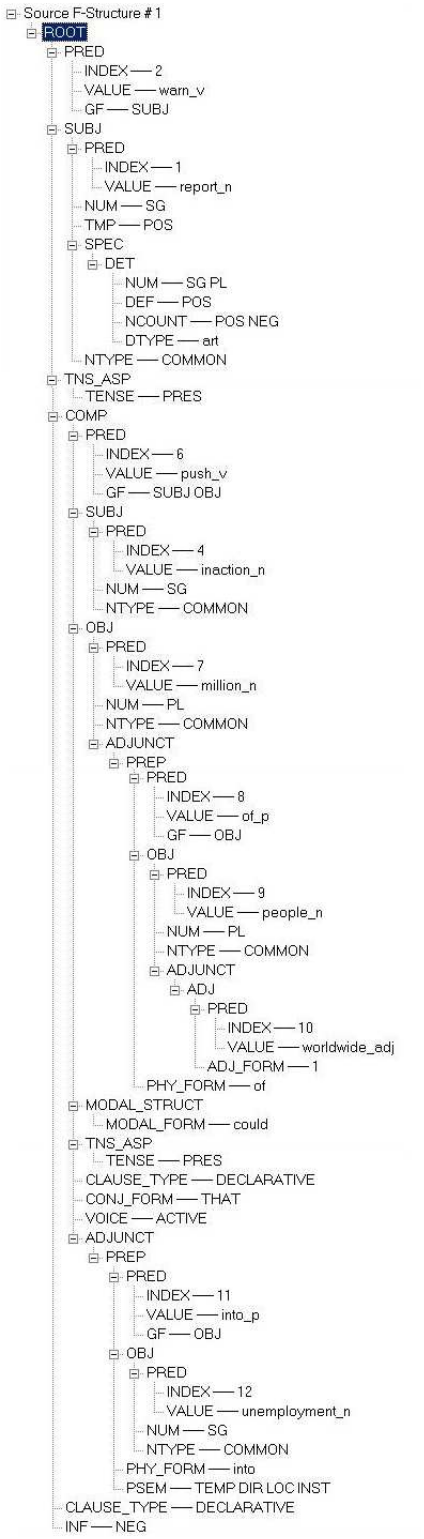


Figure 17 : A complete F-Structure using our generated LFG

The F-Structure is formed in a hierarchical manner instead of a attribute value matrix form. Each node contains either an attribute or an attribute value pair. In case if there is only one attribute, the subsidiary F-Structure is shown as the sub-tree of that attribute node. The root predicate of the F-Structure is the main verb identified in the sentence i.e. 'warn'. The 'SUBJ' grammatical function is the subject of predicate i.e. 'The report'. Tense we can report for this predicate is 'Present'. The clause `'that inaction could push millions of people worldwide into unemployment'` is identified as the close complementary clause of main predicate. This complementary clause has its own predicate i.e. 'push'. Note that 'that' is only adding an attribute to the complementary clause i.e. 'CONJ_FORM – THAT'.

Example 35:

`Roderick Daniels said police in Tenaha, Texas, took the money in October 2007 after they stopped him for doing 37 mph in a 35 mph zone.`

C-Structure of the sentence is as follows.

```
(S
  (NP-A
    (NPB   roderick/NN Daniels/NNS ) )
  (VP said/VBD
    (SBAR-A
      (S-A
        (NP-A
          (NPB  police/NNS ) )
        (PP     in/IN
```

```
  (NP-A
    (NPB     Tenaha/NNP )
    (NP (NPB Texas/NNP ) ) ) )
(VP      took/VBD
  (NP-A
    (NPB     the/DT money/NN ) )
  (PP    in/IN
    (NP-A (NPB    October/NNP 2007/CD ) ) )
  (SBAR  after/IN
    (S-A
      (NP-A
        (NPB they/PRP ) )
      (VP    stopped/VBD
        (NP-A     (NPB him/PRP ) )
        (PP  for/IN
          (SG-A
            (VP    doing/VBG
              (NP-A
                (NPB    37/CD mph/NN ) )
              (PP  in/IN
```

```
                    (NP-A
                      (NPB a/DT 35/CD mph/NN zone/NN
                       ))))))))))))))
```

The CFG rules extracted from the above C-Structure is as follows.

(35.1) S → NP-A VP

(35.2) NP-A → NPB

(35.3) NPB → NN NNS

(35.4) VP → VBD SBAR-A

(35.5) SBAR-A → S-A

(35.6) S-A → NP-A PP VP

(35.7) NP-A → NPB

(35.8) NPB → NNS

(35.9) PP → IN NP-A

(35.10) NP-A → NPB NP

(35.11) NPB → NNP

(35.12) NP → NPB

(35.13) NPB → NNP

(35.14) VP → VBD NP-A PP SBAR

(35.15) NP-A → NPB

(35.16) NPB → DT NN

(35.17) PP → IN NP-A

(35.18) NP-A → NPB

(35.19) NPB → NNP CD

(35.20) SBAR → IN S-A

(35.21) S-A → NP-A VP

(35.22) NP-A → NPB

(35.23) NPB → PRP

(35.24) VP → VBD NP-A PP

(35.25) NP-A → NPB

(35.26) NPB → PRP

(35.27) PP → IN SG-A

(35.28) SG-A → VP

(35.29) VP → VBG NP-A PP

(35.30) NP-A → NPB

(35.31) NPB → CD NN

(35.32) PP → IN NP-A

(35.33) NP-A → NPB

(35.34) NPB → DT CD NN NN

We want to align template(s) with the CFG rule (35.1). Assume that we want to check whether the following template can be aligned or not.

(35.35) S:s > * NP-A:n2 * NP:n1 * VP:v1 *
@ [n2:ADJUNCT MOD$===n1,s:↑ SUBJ===n2,
s:↑===v1]
@ [v1:↑CLAUSE_TYPE=DECLARATIVE] .

The template (35.35) can be aligned with rule (35.1) because both have ‘NP-A’ and ‘VP’ in sequence. However, there is a symbol ‘NP’ between both symbols showing that there must be another ‘NP’ in rule (35.1) between ‘NP-A’ and ‘VP’. Since, there is no other symbol on right hand side of rule (35.1) so we cannot align the template with rule (35.1). As a result, we ignore this template and look for some other template that can match the CFG rule. We have the following template to match our CFG rule.

(35.36) S:s > * NP-A:n1 * VP:v1 *
@ [v1:SUBJ===n1,s:↑===v1]
@ [v1:↑CLAUSE_TYPE=DECLARATIVE] .

The template (35.36) matches the CFG rule (35.1) exactly and can be used to annotate the CFG rule. We now find the templates matching rule (35.24).

(35.37) VP:vp > * [VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1]
(PRT:a1) [NP:n1|NPB:n1|NP-A:n1] *
@ [v1:OBJ===n1,vp:↑===v1]
@ [v1:↑INF = NEG] .

(35.38) VP:vp > * [VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] *
PP:p1 * (PP:p2 *)
@ [v1:ADJUNCT PREP$===p1,
v1:ADJUNCT PREP$===p2,vp:↑===v1]
@ [v1:↑INF=NEG] .

(35.39) VP:vp-a > * [VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] *
PP:p1 *
@ [v1:ADJUNCT PREP$===p1,vp-a:↑===v1]
@ [v1:↑INF = NEG] .

As we can see there are three possible templates that can match the rule (35.24). In template (35.37), there is an optional ‘PRT’ symbol that can be ignored while matching the template and grammar rule. The template (35.38) has another symbol ‘PP’ but this is optional too. This is to be noted that all of these templates are not conflicting and take part in annotating the CFG rule (35.24). These templates can also be used to annotate the CFG rule (35.29). Similarly, we find annotation for all CFG rules and result a LFG that can uniquely parse the sentence as follows (from rule (35.40) to (35.73)).

(35.40) S_26 → NP_A_44:↑ SUBJ = ↓ ; VP_106:↑ = ↓ ,
↑ CLAUSE_TYPE=DECLARATIVE ; .

(35.41) NP_A_44 → NPB_61:↑ = ↓ ; .

(35.42) NPB_61 → NN:↓$↑ ; NNS:↓$↑ ; .

(35.43) VP_106 → VBD:↑ = ↓ , ↑ INF = NEG ;
SBAR_A_134:↑ COMP = ↓ ; .

(35.44) SBAR_A_134 → S_A_148:↑ = ↓ ; .

(35.45) S_A_148 → NP_A_165:↑ SUBJ = ↓ ;
PP_209:↑ ADJUNCT S_PREP = ↓ ;
VP_327:↑ = ↓ ,
↑ CLAUSE_TYPE=DECLARATIVE ; .

(35.46) NP_A_165 → NPB_181:↑ = ↓ ; .

(35.47) NPB_181 → NNS:↑ = ↓ ; .

(35.48) PP_209 → IN:↑ = ↓ ; NP_A_234:↑ OBJ = ↓ ; .

(35.49) NP_A_234 → NPB_250:↑ = ↓ ;
NP_279:↑ ADJUNCT NOUN_MOD = ↓ ; .

(35.50) NPB_250 → NNP:↑ = ↓ ; .

(35.51) NP_279 → NPB_294:↑ = ↓ ; .

(35.52) NPB_294 → NNP:↑ = ↓ ; .

(35.53) VP_327 → VBD:↑ = ↓ , ↑ INF = NEG ;
NP_A_354:↑ OBJ = ↓ ;
PP_404:↑ ADJUNCT PREP = ↓ ;
SBAR_493:↑ COMP = ↓ ; .

(35.54) NP_A_354 → NPB_369:↑ = ↓ ; .

(35.55) NPB_369 → DT:↑ SPEC DET = ↓ ; NN:↑ = ↓ ; .

(35.56) PP_404 → IN:↑ = ↓ ; NP_A_430:↑ OBJ = ↓ ; .

(35.57) NP_A_430 → NPB_447:↑ = ↓ ; .

(35.58) NPB_447 → NNP:↑ = ↓ ; CD:↑ SPEC CARD = ↓ ; .

(35.59) SBAR_493 → IN:↑CONJ_FORM=↓PHY_FORM ;
S_A_521:↑ = ↓ ; .

(35.60) S_A_521 → NP_A_536:↑ SUBJ = ↓ ; VP_581:↑ = ↓ ,
↑ CLAUSE_TYPE=DECLARATIVE ; .

(35.61) NP_A_536 → NPB_550:↑ = ↓ ; .

(35.62) NPB_550 → PRP:↑ = ↓ ; .

(35.63) VP_581 → VBD:↑ = ↓ , ↑ INF = NEG ;
NP_A_609:↑ OBJ = ↓ ;
PP_648:↑ ADJUNCT PREP = ↓ ; .

(35.64) NP_A_609 → NPB_622:↑ = ↓ ; .

(35.65) NPB_622 → PRP:↑ = ↓ ; .

(35.66) PP_648 → IN:↑ = ↓ ; SG_A_673:↑ COMP = ↓ ; .

(35.67) SG_A_673 → VP_687:↑ = ↓ ; .

(35.68) VP_687 → VBG:↑ = ↓ , ↑ INF = NEG ;
NP_A_713:↑ OBJ = ↓ ;
PP_758:↑ ADJUNCT PREP = ↓ ; .

(35.69) NP_A_713 → NPB_726:↑ = ↓ ; .

(35.70) NPB_726 → CD:↑ SPEC CARD = ↓ ; NN:↑ = ↓ ; .

(35.71) PP_758 → IN:↑ = ↓ ; NP_A_781:↑ OBJ = ↓ ; .

(35.72) NP_A_781 → NPB_795:↑ = ↓ ; .

(35.73) NPB_795 → DT:↑ SPEC DET = ↓ ;
CD:↑ SPEC CARD = ↓ ; NN:↓$↑ ;
NN:↓$↑ ; .

5 RESULTS

5.1 Training

In the first iteration, the system is trained for randomly selected 100 sentences from Penn Treebank Corpus. The C-Structures of parsed sentences are observed and corresponding templates are extracted. Four iterations are made. In all iterations, a batch of 50 sentences is parsed; manually checked and corresponding templates are added wherever necessary. The following table describes the training phase.

Sentences per Iteration	100	50	50	50	50
Cumulative Templates	228	242	248	259	267
Addition	0	14	6	11	8

Table 1 : Iterations for development of Templates

As the above table shows that in first iteration of 100 sentences we have added 228 templates. The second iteration has added 14 templates, the 3rd has added 6, 4th has added 11 and the 5th has added 8. The reason of increasing graph of total templates in Figure 18 is the nature of natural language. The grammars obtained from a natural language continue to grow [3] if there is no compaction technique applied [19] [3]. The selection of our training sentences is made from the available news websites (including BBC and CNN). Figure 19 shows the change in number of Templates extracted in each iteration.

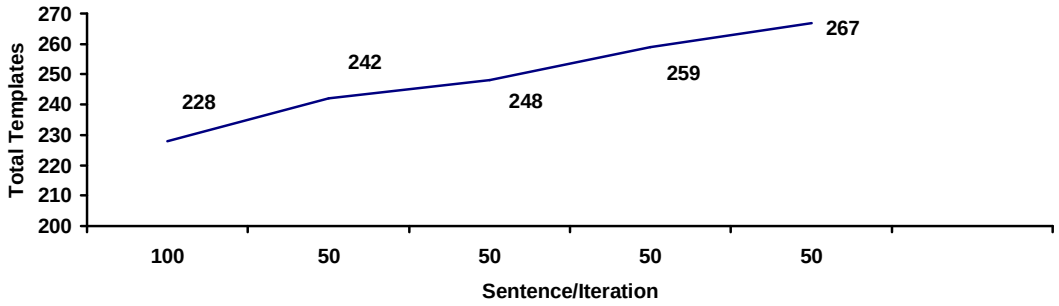


Figure 18 : Graph between Total Template and Sentences per training iteration

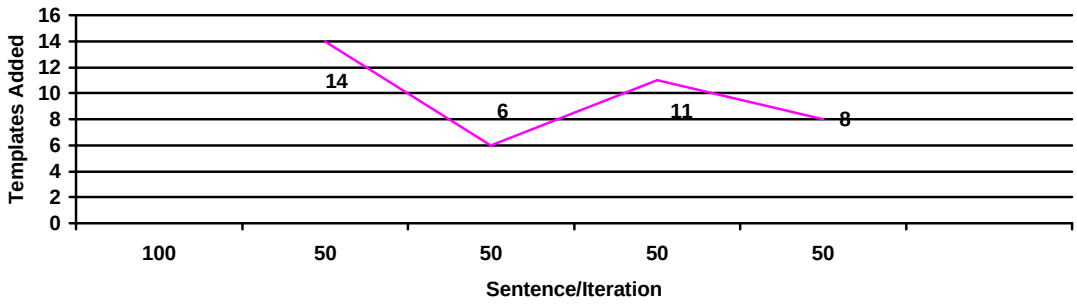


Figure 19 : Graph of Templates addition per iteration

5.2 Testing

We have tested 105 sentences for the annotation system. The selection of sentences is as follows.

BBC	45
CNN	45
Jang English News	15

The testing sentences selected from BBC and CNN covers the categories *world news, sports, and weblogs*. The number of sentences selected from each category is 15. Only the *world news* category is selected from Jang English News for testing sentences collection. The average sentence length is 22.4 words. Following sections discusses the quantitative and qualitative analysis of the testing.

5.2.1 Quantitative Analysis

We evaluate the system in terms of precision and recall as follows:

$$\text{precision} = \frac{\#\text{generated annotations also in reference}}{\#\text{generated annotations}}$$

$$\text{recall} = \frac{\#\text{generated annotations also in reference}}{\#\text{reference annotations}}$$

Precision	0.986
Recall	0.932

Here, the term annotation implies the LFG annotated rule. We have counted LFG rules in order to report results. The required countable form of LFG rule is shown in Example 34: (LFG rules from (34.22) to (34.40)) i.e. we ignore the 'uniqueness' we added in Section 3.3.4 in form of numbering. We also ignore the LFG rules having only one symbol on right hand side. So, in Example 33, we have '10' unique LFG rules annotated by our system. Also, an LFG rule is considered correct if and only if all the symbols are annotated correctly.

Total reference unique LFG rules are 397. Our system has generated 375 LFG rules. 5 of them are not correctly annotated. Hence, we have a total 370 of correct LFG rules out of 397 reference LFG rules. We have calculated PRECISION as a ratio of 370 correctly generated LFG rules with total 375 generated LFG rules and RECALL as a ratio of 370 correctly generated LFG rules with total 397 LFG rules in our reference. The system has generated these 375 LFG rule using 133 Templates (out of total 267 Templates). The percentage of Templates usage is as follows.

$$\text{percentage of templates used} = \frac{\#\text{Templates Used}}{\#\text{Total Templates}} \times 100 = \frac{133}{267} \times 100 = 49.81\%$$

The percentage of templates used shows the coverage of manually developed Templates. The results show that 50.19% of the Templates have not participated in the generation of required LFG rules. Hence, the coverage of Templates for our test sentences results to be 49.81%.

The test sentences are also tested on the Machine Translation System[1] that uses a manually crafted LFG. The system has resulted following stats shown in Table 2.

[1] See www.crulp.org

Results	Number Of sentences
CFG or F-Structure failed	48
No F-Structure	52
F-Structure	5

Table 2 : Results of Parsing system (in Machine Translation System)

Table 2 shows that 48 sentences have been failed during CFG parsing or F-Structure building. 52 sentences have been timed out and showed no results. Only 5 sentences have shown required F-Structure. On the other hand, the LFG generated by the Annotation System has successfully resulted F-Structures when used in Pipeline parsing model (section 2.3.1) for all 105 test sentences.

5.2.2 Qualitative Analysis

The following discussion shows the analysis of the issues and errors our system has made. In this section we exemplify the type of errors and reasons to count them. There are following three types of reasons observed.

1. Collins' parser has not performed as expected and hence misguided our system.
2. Our system has selected the wrong template thus the resulting annotation is wrong.
3. Our system cannot find any template matching the CFG rule.

In type 1 errors, we have checked to what level does the parser misguide. For instance, does it lead to unexpected phrase identification or wrong sub-categorization frame? If there is an insignificant error seen, for example in case of wrong POS tag or incorrect phrase identification that is categorized as ADJUNCT, the error is ignored. We have not included the sentence in our results for which parser guides our system to add or remove phrases participating in sub-categorization list of a predicate.

Type 2 errors are really serious and may affect the overall system performance. The problem is that the system cannot select a linguistically best solution among others. The

system selects a solution randomly, yet only 5 (out of 375) rules were annotated incorrectly. It decreases 'PRECISION' of the system (see Example 37 below).

Type 3 errors (or insufficient coverage) are an issue of some less severity. As described earlier that the grammars related to natural languages grow rapidly [3]. However, our observation can state that the coverage issues we found are mostly the flat rules from C-Structures i.e. the flatter grammars need more coverage and hence more training time than the non-flatter ones. This type of errors decreases the 'RECALL' of the system (see Example 38 and Example 39).

To add to all above, we find no annotation missing the vital sub-categorization argument i.e. the sub-categorization arguments of semantic forms are always marked correctly.

We quote the following example in order to show the type of errors.

Example 36:

`The United States, some European nations and Israel contend Iran's nuclear development is aimed at developing nuclear weapons.`

The sentence has subject clause `'The United States, some European nations and Israel'`. This implies that there are three nouns `'The United States'`, `'some European nations'` and `'Israel'`.

C-Structure of the sentence is as follows:

```
(S
   (NP-A (NPB   the/DT   United/NNP States/NNP))
   (NP
      (NP (NPB   some/DT European/jj nations/nns) )
      and/CC
```

```
    (NP (NPB   Israel/NNP) ))
  (VP   contend/VBP
    (SBAR-A
      (S-A
        (NP-A
          (NPB   Iran/NNP 's/POS)
          nuclear/JJ   development/NN)
        (VP   is/VBZ
          (VP-A   aimed/VBN
            (PP   at/IN
              (SG-A
                (VP    developing/VBG
                  (NP (NPB nuclear/JJ
                    weapons/NNS ))))))))))
```

The CFG rules for the given sentence are as follows.

(36.1) S → NP-A NP VP_271

(36.2) NP-A → NPB

(36.3) NPB → DT NNP NNPS

(36.4) NP → NP CC NP

(36.5) NP → NPB

(36.6) NPB → DT JJ NNS

(36.7) NP → NPB

(36.8) NPB → NNP

(36.9) VP → VBP SBAR-A

(36.10) SBAR-A → S-A

(36.11) S-A → NP-A VP

(36.12) NP-A → NPB

(36.13) NPB → NPB JJ NN

(36.14) NPB → NNP POS

(36.15) VP → VBZ VP-A

(36.16) VP-A → VBN PP

(36.17) PP → IN SG-A

(36.18) SG-A → VP

(36.19) VP → VBG NP-A

(36.20) NP-A → NPB

(36.21) NPB → JJ NNS

As CFG rule (36.1) shows there is a split in the subject phrase that marks the nouns ‘some European nations’ and ‘Israel’ separate to ‘The United States’.

Following is the LFG our system has generated for the above CFG.

(36.22) S_32 → NP_A_49:↑ SUBJ = ↓ ;
NP_121:↑ SUBJ ADJUNCT MOD = ↓ ;
VP_271:↑ = ↓ , ↑ CLAUSE_TYPE=DECLARATIVE;.

(36.23) NP_A_49 → NPB_65:↑ = ↓ ; .

(36.24) NPB_65 → dt:↑ SPEC DET = ↓ ; nnp:↓$↑ ;
nnps:↓$↑ ; .

(36.25) NP_121 → NP_137:↓$↑ ; cc:↑CONJ_FORM=↓CONJ_FORM ;
NP_220:↓$↑ ; .

(36.26) NP_137 → NPB_154:↑ = ↓ ; .

(36.27) NPB_154 → dt:SPEC DET = ↓ ; jj:↑ ADJUNCT ADJ = ↓;
nns:↑ = ↓ ; .

(36.28) NP_220 → NPB_236:↑ = ↓ ; .

(36.29) NPB_236 → nnp:↑ = ↓ ; .

(36.30) VP_271 → vbp:↑ = ↓ , ↑ INF = NEG ;
SBAR_A_300:↑ COMP = ↓ ; .

(36.31) SBAR_A_300 → S_A_312:↑ = ↓ ; .

(36.32) S_A_312 → NP_A_334:↑ SUBJ = ↓ ; VP_434:↑ = ↓ ,
↑ CLAUSE_TYPE=DECLARATIVE ; .

(36.33) NP_A_334 → NPB_355:↑ = ↓ ; .

(36.34) NPB_355 → NPB_367:↑ SPEC DET = ↓ ;
jj:↑ ADJUNCT ADJ = ↓ ; nn:↑ = ↓ ; .

(36.35) NPB_367 → nnp:↑ GENITIVE = ↓ , ↑ CASE=GEN ,
↑ DEF=POS , ↑ DTYPE=genitive ;
pos:↑ = ↓ ; .

(36.36) VP_434 → vbz:↑TNS_ASP=↓TNS_ASP ;
VP_A_459:↑ = ↓ , ↑ INF=NEG ; .

(36.37) VP_A_459 → vbn:↑ = ↓ , ↑ INF = NEG ;
PP_482:↑ ADJUNCT PREP = ↓ ; .

(36.38) PP_482 → in:↑ = ↓ ; SG_A_511:↑ COMP = ↓ ; .

(36.39) SG_A_511 → VP_530:↑ = ↓ ; .

(36.40) VP_530 → vbg:↑ = ↓ , ↑ INF = NEG ;
NP_A_565:↑ OBJ = ↓ ; .

(36.41) NP_A_565 → NPB_582:↑ = ↓ ; .

(36.42) NPB_582 → jj:↑ ADJUNCT ADJ = ↓ ; nns:↑ = ↓ ; .

As CFG rule (36.22) shows that nouns ‘some European nations’ and ‘Israel’ have become the noun modifiers of ‘The United States’.

This type of error is result of in-correct parse structure made by Collins’ parser. We mark it as the type 1 error. The error can be ignored because we have observed such errors do not participate in sub-categorization of the predicate e.g. modifiers have not been classified as sub-categorization of predicates. Also, we cannot guarantee a statistical parser to result a correct C-Structure always.

Example 37:

This brings me to the second big theme; who knows about the CSD and its works?

Following is the C-Structure of sentence.

```
(S
  (NP-A
    (NPB   this/DT ) )
  (VP      brings/VBZ
    (NP-A
      (NPB      me/PRP ) )
    (PP     to/TO
      (NP-A     (NPB the/DT second/JJ big/JJ theme/NN )
        (SBAR
          (WHNP  who/WP )
          (SG-A
            (VP      knows/VBZ
              (PP         about/IN
                (NP-A
                  (NP
                    (NPB  the/DT CSD/NNP ) )
                  and/CC
                  (NP~works~1~1
                    (NPB  its/PRPS works/NNS)))))))))))
```

The CFG rules are as follows.

(37.1) S → NP-A VP

(37.2) NP-A → NPB

(37.3) NPB → dt

(37.4) VP → vbz NP-A PP

(37.5) NP-A → NPB

(37.6) NPB → prp

(37.7) PP → to NP-A

(37.8) NP-A → NPB SBAR

(37.9) NPB → dt jj jj nn

(37.10) SBAR → WHNP SG-A

(37.11) WHNP → wp

(37.12) SG-A → VP

(37.13) VP → vbz PP

(37.14) PP → in NP

(37.15) NP-A → NP cc NP

(37.16) NP → NPB

(37.17) NPB → dt nnp

(37.18) NP → NPB

(37.19) NPB → prps nns

The CFG rule (37.8) shows that the clause ‘who knows about the CSD and its works’ is related with Noun phrase ‘the second big theme’. Our system generates the following LFG for above CFG.

(37.20) S_30 → NP_A_45:↑ SUBJ = ↓ ; VP_88:↑ = ↓ ,
↑ CLAUSE_TYPE=DECLARATIVE ; .

(37.21) NP_A_45 → NPB_59:↑ = ↓ ; .

(37.22) NPB_59 → dt:↑ = ↓ ; .

(37.23) VP_88 → vbz:↑ = ↓ , ↑ INF = NEG ;
NP_A_114:↑ OBJ = ↓ ;
PP_150:↑ ADJUNCT PREP = ↓ ; .

(37.24) NP_A_114 → NPB_126:↑ = ↓ ; .

(37.25) NPB_126 → prp:↑ = ↓ ; .

(37.26) PP_150 → to:↑ = ↓ , ↓INF =c NEG ;
NP_A_174:↑ OBJ = ↓ ; .

(37.27) NP_A_174 → NPB_189:↑ = ↓ ;
SBAR_246:↑ ADJUNCT REL_CL = ↓ ; .

(37.28) NPB_189 → dt:↑ SPEC DET = ↓ ; jj:↓$↑ ADJUNCT ADJ;
jj:↓$↑ ADJUNCT ADJ ; nn:↑ = ↓ ; .

(37.29) SBAR_246 → WHNP_260:↑ OBJ = ↓ ;
SG_A_287:↑ = ↓ ; .

(37.30) SBAR_246 → WHNP_260:↑ SUBJ = ↓ ;
SG_A_287:↑ = ↓ ; .

(37.31) WHNP_260 → wp:↑ = ↓ ; .

(37.32) SG_A_287 → VP_301:↑ = ↓ ; .

(37.33) VP_301 → vbz:↑ = ↓ , ↑ INF = NEG ;
PP_327:↑ ADJUNCT PREP = ↓ ; .

(37.34) PP_327 → in:↑ = ↓ ; NP_A_352:↑ OBJ = ↓ ; .

(37.35) NP_A_352 → NP_364:↓$↑ ;
cc:↑CONJ_FORM=↓CONJ_FORM ;
NP_423:↓$↑ ; .

(37.36) NP_364 → NPB_377:↑ = ↓ ; .

(37.37) NPB_377 → dt:↑ SPEC DET = ↓ ; nnp:↑ = ↓ ; .

(37.38) NP_423 → NPB_438:↑ = ↓ ; .

(37.39) NPB_438 → prps:↑ SPEC DET GEN_PRO = ↓ ;
nns:↑ = ↓ , ↑ SPEC DET DEF=POS ,
↑ SPEC DET DTYPE=gen_pro ; .

In our reference, this is not the correct parse structure and hence incorrect annotation. We do not expect parser to result parse as CFG rule (37.8) shows). As a result, the LFG rule (37.27) is not correct. ‘SBAR’ in rule (37.27) should have been classified in CFG rule (37.4). However, as the SBAR is somewhat related with the noun ‘theme’ so it can easily be confused in a parse structure. This is the reason it is classified as error type 1 and ignored in our error count.

Example 38:

A top aide to Senate Judiciary Chairman Patrick Leahy told fellow Democrats on Friday to get ready for President Obama's Supreme Court pick to come as early as next week, according to an e-mail obtained by CNN.

C-Structure of the sentence is as follows.

```
    (S
      (NP-A
        (NPB a/DT top/JJ aide/NN )
        (PP to/TO
          (NP-A
            (NPB Senate/NNP Judiciary/NNP Chairman/NNP
Patrick/NNP Leahy/NNP ) ) ) )
      (VP told/VBD
        (NP-A
          (NPB fellow/JJ Democrats/NNPS ) )
        (PP  on/IN
          (NP-A
            (NPB Friday/NNP ) ) )
        (SG-A
          (VP to/TO
            (VP-A get/VB
              (ADJP ready/JJ
                (PP for/IN
                  (NP-A
```

```
                    (NPB
                      (NPB President/NNP Obama/NNP 's/POS )
Supreme/NNP Court/NNP pick/NN )
                    (SG
                      (VP to/TO
                        (VP-A come/VB
                          (ADVP
                            (ADVP as/RB early/RB )
                            (PP as/IN
                              (NP-A
                                (NPB next/JJ
week/NN))))))))))))
          (PP according/VBG
            (PP-A to/TO
              (NP-A
                (NPB an/DT e-mail/NN )
                (VP obtained/VBN
                  (PP by/IN
                    (NP-A
                      (NPB CNN/NNP)))))))))
```

To avoid the lengthy details of example, we show the LFG generated by our system for the sentence. The LFG syntax already includes the CFG of parsed tree.

(38.1) S_26 → NP_A_41:↑ SUBJ = ↓ ; VP_224:↑ = ↓ ,
↑ CLAUSE_TYPE=DECLARATIVE ; .

(38.2) NP_A_41 → NPB_55:↑ = ↓ ;
PP_94:↑ ADJUNCT PREP = ↓ ; .

(38.3) NPB_55 → dt:↑ SPEC DET = ↓ ;
jj:↑ ADJUNCT ADJ = ↓ ; nn:↑ = ↓ ; .

(38.4) PP_94 → to:↑ = ↓ ; NP_A_118:↑ OBJ = ↓ ; .

(38.5) NP_A_118 → NPB_133:↑ = ↓ ; .

(38.6) NPB_133 → nnp:↓$↑ ; nnp:↓$↑ ; nnp:↓$↑ ;
nnp:↓$↑ ; nnp:↓$↑ ; .

(38.7) VP_224 → vbd:↑ = ↓ , ↑ INF = NEG ;
NP_A_255:↑ OBJ = ↓ ;
PP_318:↓$↑ ADJUNCT PREP ;
SG_A_391:↑ XADJUNCT=↓ , ↓INF =c NEG;
PP_819:↓$↑ ADJUNCT PREP ; .

(38.8) VP_224 → vbd:↑ = ↓ , ↑ INF = NEG ;
NP_A_255:↑ OBJ = ↓ ;
PP_318:↓$↑ ADJUNCT PREP ;
SG_A_391:↑ =↓ , ↓XCOMP INF =c POS ;
PP_819:↓$↑ ADJUNCT PREP ; .

(38.9) NP_A_255 → NPB_274:↑ = ↓ ; .

(38.10) NPB_274 → jj:↑ ADJUNCT ADJ = ↓ ; nnps:↑ = ↓ ; .

(38.11) PP_318 → in:↑ = ↓ ; NP_A_343:↑ OBJ = ↓ ; .

(38.12) NP_A_343 → NPB_359:↑ = ↓ ; .

(38.13) NPB_359 → nnp:↑ = ↓ ; .

(38.14) SG_A_391 → VP_402:↑ = ↓ ; .

(38.15) VP_402 → to:↑XCOMP INF=↓INF ;
VP_A_424:↑ XCOMP = ↓ ,
↑ XCOMP SUBJ PRED = 'pro' ,
↑ XCOMP SUBJ PRONTYPE = NULL ; .

(38.16) VP_A_424 → vb:↑ = ↓ ; ADJP_449:↑ PREDLINK = ↓ ; .

(38.17) ADJP_449 → jj:↑ = ↓ ;
PP_472:↑ ADJUNCT PREP = ↓ ; .

(38.18) PP_472 → in:↑ = ↓ ; NP_A_496:↑ OBJ = ↓ ; .

(38.19) NP_A_496 → NPB_510:↑ = ↓ ;
SG_610:↑ ADJUNCT PARTICIPLE = ↓ ; .

(38.20) NPB_510 → NPB_522:↑ SPEC DET = ↓ ; nnp:↓$↑ ;
nnp:↓$↑ ; nn:↓$↑ ; .

(38.21) NPB_522 → nnp:↑ GENITIVE ADJUNCT TITLE = ↓ ;
nnp:↑ GENITIVE = ↓ , ↑ CASE=GEN ,
↑ DEF=POS , ↑ DTYPE=genitive ;
pos:↑ = ↓ ; .

(38.22) SG_610 → VP_621:↑ = ↓ ; .

(38.23) VP_621 → to:↑XCOMP INF=↓INF ;
VP_A_644:↑ XCOMP = ↓ ,

↑ XCOMP SUBJ PRED = 'pro' ,
↑ XCOMP SUBJ PRONTYPE = NULL ; .

(38.24) VP_A_644 → vb:↑ = ↓ ;
ADVP_670:↑ ADJUNCT ADV = ↓ ; .

(38.25) ADVP_670 → ADVP_686:↑ = ↓ ; PP_718:↑ = ↓ ; .

(38.26) ADVP_686 → rb:↓$↑ ; rb:↓$↑ ; .

(38.27) PP_718 → in:↑ = ↓ ; NP_A_741:↑ OBJ = ↓ ; .

(38.28) NP_A_741 → NPB_755:↑ = ↓ ; .

(38.29) NPB_755 → jj:↑ ADJUNCT ADJ = ↓ ; nn:↑ = ↓ ; .

(38.30) PP_819 → vbg:↑ = ↓ ; PP_A_848:↑ COMP = ↓ ; .

(38.31) PP_A_848 → to:↑ = ↓ ; NP_A_873:↑ OBJ = ↓ ; .

(38.32) NP_A_873 → NPB_889:↑ = ↓ ;
VP_928:↑ ADJUNCT PARTICIPLE = ↓ ; .

(38.33) NPB_889 → dt:↑ SPEC DET = ↓ ; nn:↑ = ↓ ; .

(38.34) VP_928 → vbn:↑ = ↓ , ↑ INF = NEG ;
PP_954:↑ ADJUNCT PREP = ↓ ; .

(38.35) PP_954 → in:↑ = ↓ ; NP_A_976:↑ OBJ = ↓ ; .

(38.36) NP_A_976 → NPB_989:↑ = ↓ ; .

(38.37) NPB_989 → nnp:↑ = ↓ ; .

The LFG rule (38.19) makes the ‘to come as early as next week’ as the ‘ASJUNCT PARTICIPLE’ of ‘President Obama's Supreme Court pick’. We mark it as the type 2 error i.e. our system has incorrectly annotated the symbols. The

original solution to this problem by looking at rule (38.22) and (38.23) should look down the tree hierarchy and decide on the basis of constraints. Following is the reference solution for this wrong annotation.

(38.38) NP_A_496 → NPB_510:↑ = ↓ ;
SG_610:↑ ADJUNCT PARTICIPLE = ↓ ,
↓ INF =c NEG ; .

(38.39) NP_A_496 → NPB_510:↑ = ↓ ;
SG_610:↑ = ↓ ,
↓ XCOMP INF =c POS ; .

The LFG rule (38.38) and (38.39) describes that 'SG' is marked '↑ ADJUNCT PARTICIPLE = ↓' if the SG has finite verb as predicate. If there is some non-finite clause below the 'SG', it is annotated with relation '↑=↓'. By looking the CFG rule (38.22) and (38.23), the LFG rule (38.39) leads to a correct computable F-Structure in this sentence. This problem adds count to the wrong annotations and is the reason to decrease in 'PRECISION'.

Example 39:

For me, as president of UEFA, now this year there is even greater expectation because the teams are playing very well, but as president I look more at security than the game sometimes.

The C-Structure is as follows.

```
(S
  (S
    (PP for/IN
      (NP-A (NPB me/PRP ) ) )
```

```
  (PP as/IN
    (NP-A
      (NPB president/NN )
      (PP of/IN
        (NP-A (NPB UEFA/NNP ) ) ) ) )
  (ADVP now/RB )
  (NP (NPB this/DT year/NN ) )
  (NP-A (NPB there/EX ) )
  (VP is/VBZ
    (NP-A
      (NPB
        (ADJP even/RB greater/JJR ) expectation/NN ) )
    (SBAR because/IN
      (S-A
        (NP-A (NPB the/DT teams/NNS ) )
        (VP are/VBP
          (VP-A playing/VBG
            (ADVP very/RB well/RB)))))))
but/CC
(S
```

```
(PP as/IN
  (NP-A (NPB president/NN ) ) )
(NP-A
  (NPB I/PRP ) )
(VP look/VBP
  (ADJP
    (ADJP more/JJR
      (PP at/IN
        (NP-A (NPB security/NN ) ) ) )
    (PP than/IN
      (NP-A  (NPB the/DT game/NN ) ) ) )
  (ADVP sometimes/RB ) ) ) )
```

The corresponding LFG generated by our system is as follows.

(39.1) ADVP_1067 → rb:↑ = ↓ ; .

(39.2) NPB_1020 → dt:↑ SPEC DET = ↓ ; nn:↑ = ↓ ; .

(39.3) NP_A_1006 → NPB_1020:↑ = ↓ ; .

(39.4) PP_981 → in:↑ = ↓ ; NP_A_1006:↑ OBJ = ↓ ; .

(39.5) NPB_946 → nn:↑ = ↓ ; .

(39.6) NP_A_928 → NPB_946:↑ = ↓ ; .

(39.7) PP_901 → in:↑ = ↓ ; NP_A_928:↑ OBJ = ↓ ; .

(39.8) ADJP_879 → jjr:↓$↑ ADJUNCT ;
PP_901:↓$↑ ADJUNCT ; .

(39.9) ADJP_864 → ADJP_879:↑ = ↓ ;
PP_981:↑ ADJUNCT PREP = ↓ ; .

(39.10) VP_838 → vbp:↑ = ↓ , ↑ INF = NEG ;
ADJP_864:↑ PREDLINK = ↓ ;
ADVP_1067:↑ ADJUNCT ADV = ↓ ; .

(39.11) NPB_813 → prp:↑ = ↓ ; .

(39.12) NP_A_802 → NPB_813:↑ = ↓ ; .

(39.13) NPB_769 → nn:↑ = ↓ ; .

(39.14) NP_A_750 → NPB_769:↑ = ↓ ; .

(39.15) PP_722 → in:↑ = ↓ ; NP_A_750:↑ OBJ = ↓ ; .

(39.16) S_711 → PP_722:↑ ADJUNCT S_PREP = ↓ ;
NP_A_802:↑ SUBJ = ↓ ; VP_838:↑ = ↓ ,
↑ CLAUSE_TYPE=DECLARATIVE ; .

(39.17) ADVP_656 → rb:↓$↑ ; rb:↓$↑ ; .

(39.18) VP_A_627 → vbg:↑ = ↓ , ↑ INF = NEG ;
ADVP_656:↑ ADJUNCT ADV = ↓ ; .

(39.19) VP_599 → vbp:↑TNS_ASP=↓TNS_ASP ;
VP_A_627:↑ = ↓ , ↑ INF=NEG ; .

(39.20) NPB_562 → dt:↑ SPEC DET = ↓ ; nns:↑ = ↓ ; .

(39.21) NP_A_547 → NPB_562:↑ = ↓ ; .

(39.22) S_A_531 → NP_A_547:↑ SUBJ = ↓ ;
VP_599:↑ = ↓ ,
↑ CLAUSE_TYPE=DECLARATIVE ; .

(39.23) SBAR_505 → in:↑CONJ_FORM=↓PHY_FORM ;
S_A_531:↑ = ↓ ; .

(39.24) ADJP_440 → rb:↑ ADJUNCT ADV = ↓ ; jjr:↑ = ↓ ; .

(39.25) NPB_422 → ADJP_440:↑ ADJUNCT ADJ = ↓ ;
nn:↑ = ↓ ; .

(39.26) NP_A_401 → NPB_422:↑ = ↓ ; .

(39.27) VP_370 → vbz:↑ = ↓ , ↑ INF=NEG ;
NP_A_401:↑ OBJ = ↓ ;
SBAR_505:↑ COMP = ↓ ; .

(39.28) NPB_344 → ex:↑ = ↓ ; .

(39.29) NP_A_329 → NPB_344:↑ = ↓ ; .

(39.30) NPB_289 → dt:↑ SPEC DET = ↓ ; nn:↑ = ↓ ; .

(39.31) NP_275 → NPB_289:↑ = ↓ ; .

(39.32) ADVP_251 → rb:↑ = ↓ ; .

(39.33) NPB_216 → nnp:↑ = ↓ ; .

(39.34) NP_A_202 → NPB_216:↑ = ↓ ; .

(39.35) PP_179 → in:↑ = ↓ ; NP_A_202:↑ OBJ = ↓ ; .

(39.36) NPB_151 → nn:↑ = ↓ ; .

(39.37) NP_A_132 → NPB_151:↑ = ↓ ;
PP_179:↑ ADJUNCT PREP = ↓ ; .

(39.38) PP_104 → in:↑ = ↓ ; NP_A_132:↑ OBJ = ↓ ; .

(39.39) NPB_78 → prp:↑ = ↓ ; .

(39.40) NP_A_66 → NPB_78:↑ = ↓ ; .

(39.41) PP_44 → in:↑ = ↓ ; NP_A_66:↑ OBJ = ↓ ; .

(39.42) S_32 → PP_44:↑ ADJUNCT = ↓ ;
PP_104:↑ ADJUNCT S_PREP = ↓ ;
ADVP_251:↑ ADJUNCT S_ADV = ↓ ;
NP_275:↑ SUBJ ADJUNCT MOD = ↓ ;
NP_A_329:↑ SUBJ = ↓ ; VP_370:↑ = ↓ ,
↑ CLAUSE_TYPE=DECLARATIVE ; .

(39.43) S_22 → S_32:↓$↑ ; cc:↑CONJ_FORM=↓CONJ_FORM ;
S_711:↓$↑ ; .

In the LFG rule (39.8) ‘JJR’ and ‘PP’ are annotated ‘ADJUNCT’. This is because system does not have sufficient coverage to annotate them accordingly. This error is classified as type 3 error. It is added to the coverage count and reduces system’s coverage. Similarly, system has been unable to annotate ‘PP’ in LFG rule (39.42). This is because we have not encountered any such example in the training phase. Although, it is annotated ‘ADJUNCT’ which is partially correct, yet the LFG rule (39.42) increases count of ‘insufficient coverage’ as 1. As a result, this sentence has incremented the total count of un-covered LFG rules by 2.

Example 40:

The poll also indicates that 42 percent of people questioned think the country's in a serious recession, up 10 points from last October.

C-Structure of the sentence:

```
(S
  (NP-A (NPB the/DT poll/NN ) )
  (ADVP also/RB )
  (VP indicates/VBZ
    (SBAR-A that/IN
      (S-A
        (NP-A (NPB 42/CD percent/NN )
          (PP of/IN
            (NP-A (NPB people/NNS )
              (VP questioned/VBN ) ) ) )
        (VP think/VBP
          (SBAR-A
            (S-A
              (NP-A (NPB the/DT country/NN ) )
              (VP 's/VBZ
                (PP in/IN
                  (NP-A
                   (NPB a/DT serious/JJ recession/NN)))
                (ADVP up/RB
```

```
                    (NP (NPB 10/CD points/NNS ) )
                    (PP from/IN
                     (NP-A
                      (NPB last/JJ October/NNP))))))))))))
```

The corresponding LFG for above C-Structure is as follows.

(40.1) S_36 → NP_A_51:↑ SUBJ = ↓ ;
ADVP_103:↑ ADJUNCT S_ADV = ↓ ;
VP_133:↑ = ↓ ,
↑ CLAUSE_TYPE=DECLARATIVE ; .

(40.2) NP_A_51 → NPB_65:↑ = ↓ ; .

(40.3) NPB_65 → dt:↑ SPEC DET = ↓ ; nn:↑ = ↓ ; .

(40.4) ADVP_103 → rb:↑ = ↓ ; .

(40.5) VP_133 → vbz:↑ = ↓ , ↑ INF = NEG ;
SBAR_A_166:↑ COMP = ↓ ; .

(40.6) SBAR_A_166 → in:↑CONJ_FORM=↓PHY_FORM ;
S_A_191:↑ = ↓ ; .

(40.7) S_A_191 → NP_A_209:↑ SUBJ = ↓ ;
VP_374:↑ = ↓ ,
↑ CLAUSE_TYPE=DECLARATIVE ; .

(40.8) NP_A_209 → NPB_226:↑ = ↓ ;
PP_260:↑ ADJUNCT PREP = ↓ ; .

(40.9) NPB_226 → cd:↑ SPEC CARD = ↓ ; nn:↑ = ↓ ; .

(40.10) PP_260 → in:↑ = ↓ ; NP_A_285:↑ OBJ = ↓ ; .

(40.11) NP_A_285 → NPB_301:↑ = ↓ ;
VP_335:↑ ADJUNCT PARTICIPLE = ↓ ; .

(40.12) NPB_301 → nns:↑ = ↓ ; .

(40.13) VP_335 → vbn:↑ = ↓ ; .

(40.14) VP_374 → vbp:↑ = ↓ , ↑ INF = NEG ;
SBAR_A_401:↑ COMP = ↓ ; .

(40.15) SBAR_A_401 → S_A_413:↑ = ↓ ; .

(40.16) S_A_413 → NP_A_431:↑ SUBJ = ↓ ;
VP_485:↑ = ↓ ,
↑ CLAUSE_TYPE=DECLARATIVE ; .

(40.17) NP_A_431 → NPB_448:↑ = ↓ ; .

(40.18) NPB_448 → dt:↑ SPEC DET = ↓ ; nn:↑ = ↓ ; .

(40.19) VP_485 → vbz:↑ = ↓ , ↑ INF = NEG ;
PP_505:↑ ADJUNCT PREP = ↓ ;
ADVP_606:↑ ADJUNCT ADV = ↓ ; .

(40.20) PP_505 → in:↑ = ↓ ; NP_A_533:↑ OBJ = ↓ ; .

(40.21) NP_A_533 → NPB_552:↑ = ↓ ; .

(40.22) NPB_552 → dt:↑ SPEC DET = ↓ ;
jj:↑ ADJUNCT ADJ = ↓ ; nn:↑ = ↓ ; .

(40.23) ADVP_606 → rb:↑ = ↓ ; NP_629:↑ ADJUNCT = ↓ ;
PP_683:↑ ADJUNCT PREP = ↓ ; .

(40.24) NP_629 → NPB_645:↑ = ↓ ; .

(40.25) NPB_645 → cd:↑ SPEC CARD = ↓ ; nns:↑ = ↓ ; .

(40.26) PP_683 → in:↑ = ↓ ; NP_A_711:↑ OBJ = ↓ ; .

(40.27) NP_A_711 → NPB_728:↑ = ↓ ; .

(40.28) NPB_728 → jj:↑ ADJUNCT ADJ = ↓ ; nnp:↑ = ↓ ; .

The ‘NP’ in LFG rule (40.23) is marked as ‘ADJUNCT’. Our system has been unable to find a proper template that can annotate the ‘NP’ before a ‘PP’ in an Adverbial phrase. Hence, it has been annotated with system default behavior. This problem is also the example of insufficient coverage and reason to decrease ‘RECALL’.

The section 5.2.1 also shows the coverage of the Templates. There are three types of Templates normally not selected by Annotation System.

1 The template is too specific to occur frequently. For instance, if there is no wild card used in a Template, it become specific. The more symbols occur in a template, the more specific and rare to occur it becomes. Consider the following Template which doesn’t have a wild card used and hence has became more context specific than others.

```
NPB:npb > QP:q1 [NN:n1|NNS:n1|NNP:n1|NNPS:n1] POS:p1
        @ [n1:SPEC QUANT===q1,
           n1: SPEC DET GEN_PRO===p2,
           npb:^===p1,npb:^GENITIVE===n1]
        @ [n1:^ CASE=GEN,n1:^ DEF=POS,
           n1:^ DTYPE=genitive] .
```

This Template can be used only if a Quantifier Phrase is followed by a noun and a possessive marker. In our test sentences, this specific and precise case does not occur and hence the template is not used.

2 The template contains symbols which are rare to occur. For instance, the FRAG symbol shows the fragmentation in a C-Structure. This fragmentation is the result

of inadequate parse from Collins' parser. The fragmentation is not like to occur frequently and hence the Templates containing such symbols are not generally used.

```
SBARQ:sbarq > * WRB:w1 * FRAG:f1 *
            @ [f1:ADJUNCT S_ADV$===w1,sbarq:^===f1]
            @ [f1:^CLAUSE_TYPE=INTERROGATIVE] .
```

The template has a symbol FRAG which has not occurred in our tests and hence the above template is not used.

3 A template has such LHS that has not been occurred in CFG of test sentences. For instance, a template that is specific with interrogative sentences is not used for a declarative sentence.

```
SQ:sq > * ADVP:a1 * VP:v1 *
      @ [v1:ADJUNCT S_ADV$===a1,sq:^===v1]
      @ [v1:^CLAUSE_TYPE=INTERROGATIVE] .

SQ:sq > * MD:m1 * [NP:n1|NP-A:n1] VP:v2 *
      @ [v2:SUBJ===n1,sq:^===m1,sq:^===v2]
      @ [v2:^CLAUSE_TYPE=INTERROGATIVE,
         m1:^HelpVP TNS_ASP=!TNS_ASP] .

SQ:sq > * MD:m1 * VP:v1 *
      @ [sq:^===m1,sq:^===v1]
      @ [v1:^CLAUSE_TYPE=INTERROGATIVE] .

SQ:sq > * SBAR:s1 * VP:v1 *
       @ [v1:COMP$===s1,sq:^===v1]
       @ [v1:^CLAUSE_TYPE=INTERROGATIVE] .

SQ:sq > [VBP:v1|VBZ:v1|VBD:v1] * [NP:n1|NP-A:n1] *
        VP:v2 *
       @ [v2:SUBJ===n1,
```

```
    v2:HelpVP TNS_ASP===v1:TNS_ASP,
    sq:^===v2]
@ [v2:^CLAUSE_TYPE=INTERROGATIVE,
    v2:^HelpVP TNS_ASP=!TNS_ASP,
    v1:^TNS_ASP=!TNS_ASP] .
```

None of the test sentences contains an interrogative sentence so none of the above templates is used.

The observation shows that the more generic template is more likely to occur unless it contains a less frequent symbol. The less symbols on right hand side of a template lead to a more generic template. Following is an example of the most generic template.

```
S-A:s-a > * PP:p1 * @ [s-a:ADJUNCT S_PREP===p1] .
```

The template is applicable if there is a prepositional phrase (PP) under sentence symbol (S-A). The occurrence of PP is independent of its predecessor or successor symbols.

6 CONCLUSION

We have proposed and developed a system to automatically generate LFG for English language. The aim of the system is to enable the Machine Translation system to re-parse and generate the F-Structures for English sentences. The results are encouraging and induce us to use this system to build LFG. Possible improvements in tagging (the POS tags) and parsing can even improve the accuracy and coverage of the system. We have also observed that development of templates have been quite easy than the development of Lexical Functional Grammar for English. The performance of the generating English F-Structure has also noticeably improved in terms of time than the use of manually crafted Lexical Functional Grammar based F-Structure building.

7 FUTURE WORK

The system can be enhanced by adding more iterations in the training phase and by adding more template rules. This can probably improve recall of the system. We have observed that development of these grammars is relatively easy and consumes less time than crafting large coverage, rich unification based grammar resources. The parallel technique presented in Section 2.3.4.2 can also be used and tested, which can save time by avoiding the uncertainty (Section 4.3).

Another possible future work can be the use of the system to build a large annotated corpus to be used in the integrated model presented in Section 2.3.2 . However, as we have used already trained statistical parser [22] [23], this addition will demand the Collins' parser to be retrained on the annotated corpus or probably will require a new parser to be built and trained on our linguistic analysis.

A potential work is to identify the voice of sentence. We could not address this problem because of the recursive nature of Collins' parser. However, preprocessing can add some heuristic that can be used for this purpose.

Section 5.2.2 suggests that the technique like grammar compaction [19] [3] can also significantly change the results. The more recursive grammars can possibly perform even better as most of the coverage issues are the reason of flatter rules.

8 REFERENCE

[1] Cahill A., McCarthy M., van Genabith J. and Way A. (2002a), "Parsing with PCFGs and Automatic F-Structure Annotation", Proceedings of the *Seventh International Conference on LFG*, CSLI Publications, Stanford, CA., pp.76-95

[2] Kinyon A., Prolo Carlos A. (2002), "A classification of grammar development strategies", in the Proceeding of *COLING-02 on Grammar engineering and evaluation*, pp.1-7.

[3] Krotov, A., M. Hepple, R. Gaizauskas, and Y. Wilks. (1998), "Compactinkg the Penn Treebank Grammar", In Proceedings of *COLING/ACL'98*, pp 699-703.

[4] Charniak, E. (1996), "Tree-bank Grammars". In *AAAI-96. Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp:1031-1036. MIT Press.

[5] van Genabith, J., Way A., and Sadler L. (1999c), "Semi-Automatic Generation of F-Structures from Tree Banks". In M. Butt and T. King (Eds.), *Proceedings of the LFG99 Conference*, Manchester University, 19-21 July, CSLI Online Publications, Stanford, CA. http://www-csli.stanford.edu/publications/.

[6] van Genabith, J., Sadler L., and Way A. (1999a), "Data-Driven Compilation of LFG Semantic Forms". In *EACL'99 Workshop on Linguistically Interpreted Corpora (LINC-99)*, pp: 69-76, Bergen, Norway, June 12th.

[7] Sadler L., Genabith J. and Way A. (2000), "Automatic F-Structure Annotation from the AP Treebank". *Fifth International Conference on Lexical-Functional Grammar*, The University of California at Berkeley, CSLI Publications, Stanford, CA

[8] Cahill A., McCarthy M., van Genabith J. and Way A. (2002c), "Evaluating Automatic F-Structure Annotation for the Penn II Treebank", in Proceedings of the *Treebanks and Linguistic Theories (TLT'02) Workshop*, Sozopol. Bulgaria.

[9] Hutchins, W. J. and Somers, H. L. (1992), *An introduction to machine translation*, Academic Press, London.

[10] Kaplan R., Bresnan J. (1982), Lexical Functional Grammar: a formal system for grammatical representation. In Bresnan, J. editor 1982, *The Mental representation of Grammatical Relations*. MIT Press, Cambridge Mass. 173-281

[11] Chrupala G. (2008), *Towards a machine-learning architecture for Lexical Functional Grammar Parsing*. PhD Dissertation, Dublin City University.

[12] Kaplan R. (1989), The formal architecture of Lexical-Functional Grammar. *Journal of Information Science and Engineering, vol. 5*, pp: 305--322. Reprinted in Dalrymple,

Kaplan, Maxwell, and Zaenen (eds), *Formal Issues in Lexical-Functional Grammar*, pp:7-27. Stanford: Center for the Study of Language and Information 1995.

[13] Butt M., King T.H., Nifio M.E., and F. Segond. (1999), *A Grammar Writer's Cookbook*. CSLI Publications, Stanford, CA

[14] Kaplan R. M., Netter K., Wedekind J. & Zaenen, A. (1989), "Translation by structural correspondences", in '*Proceedings of the 4th Conference of the European Chapter of the Association for Computational Linguistics, UMIST, Manchester, pp:10-12 April 1989'*, Association for Computational Linguistics, pp: 272-281, New Brunswick, NJ.

[15] Jurafsky D. and Martin J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*, Prentice Hall Series in Artificial Intelligence, Prentice Hall.

[16] Bresnan, J. 2001. *Lexical Functional Syntax*. Blackwells Publishers, Oxford.

[17] Kroeger, P. 1995. *Phrase Structure and Grammatical Relations in Tagalog*. Stanford: CSLI.

[18] Kroeger, Paul R. (2005), *Analyzing grammar: an introduction*. New York: Cambridge University Press

[19] van Genabith, J., Sadler L., and Way A. (1999b), "Structure Preserving CF-PSG Compaction, LFG and Treebanks". In Proceedings *ATALA Workshop - Treebanks, Journ´ees ATALA, Corpus annot´es pour la syntaxe*, pp: 107–114, Universite Paris 7, France, 18-19 Juin 1999.

[20] Frank A. (2000), "Automatic F-Structure Annotation of Treebank Trees". In: M. Butt and T.H. King editors, *Proceedings of the LFG00 Conference*, University of California at Berkeley, CSLI Online Publications, Stanford

[21] Frank A., Sadler L., van Genabith J. and Way A. (2002), "From Treebank Resources to LFG F-Structures". In (ed.)Anne Abeille, *Treebanks: Building and Using Syntactically Annotated Corpora*, Kluwer Academic Publishers

[22] Collins M. J., (1999), *Head-Driven Statistical Models for Natural Language Parsing*. PhD Dissertation, University of Pennsylvania

[23] Collins, M. J. (1996). "A new statistical parser based on bigram lexical dependencies". In Proceedings of *the 34th Annual Meeting of the ACL* Ma. 1996.

[24] Khalid, U., Karamat, N., Iqbal, S. and Hussain, S. (2009) "Semi-Automatic Lexical Functional Grammar Development", Proceedings of the *Conference on Language and*

Technology 2009 (CLT09), FAST NU, Lahore, Pakistan, 22-24 Jan 2009 (URL: http://www.crulp.org/clt09/index.htm)

[25] Quirk, R., Svartvik, J., Leech, G. (1985), *A Comprehensive Grammar of the English Language.* Addison-Wesley Publications, .

[26] Karamat, N., (2006), *Verb Transfer for English to Urdu Machine Translation (Using Lexical Functional Grammar (LFG))*, National University of Computer and Emerging Sciences, Lahore, Pakistan, 2006.

[27] Tsuruoka Y. and Tsujii J. (2005), "Bidirectional Inference with the Easiest-First Strategy for Tagging Sequence Data", Proceedings of *HLT/EMNLP*, pp. 467-474.

[28] Cahill A., McCarthy M., van Genabith J. and Way A. (2002b), "Automatic Annotation of the Penn-Treebank with LFG F-Structure Information", *LREC.*

[29] Bod R. and Kaplan R., (1998), "A probabilistic corpus-driven model for lexical-functional grammar". In *Proceedings of Coling/ACL'98*, pp: 145–151.

[30] Kay M. (1999), "Chart Translation". In *Proceedings of the Machine Translation Summit VII. "MT in the great Translation Era"*, pp: 9–14.

[31] Cahill, A., M. Burke, R. O'Donovan, J. van Genabith, and A. Way. (2004a), "Long-Distance Dependency Resolution in Automatically AcquiredWide-Coverage PCFG-Based LFG Approximations". In *Proceedings of 42nd Conference of the Association for Computational Linguistics*, pages 319–326, Barcelona, Spain. printed.

[32] Burke, M., Cahill, A., O'Donovan, R., van Genabith, J., and Way, A., (2004), "Treebank-Based Acquisition of Wide-Coverage, Probabilistic LFG Resources: Project Overview, Results and Evaluation", In *Proceedings of The First International Joint Conference on Natural Language Processing (IJCNLP-04), Workshop "Beyond shallow analyses - Formalisms and statistical modeling for deep analyses"*; March 22-24, 2004 Sanya City, Hainan Island, China.

[33] Burke, M., Cahill, A., O'Donovan, Ruth., van Genabith, J., and Way, A., (2004), "The Evaluation of an Automatic Annotation Algorithm against the PARC 700 Dependency Bank". In *Proceedings of the Ninth International Conference on LFG*, Christchurch, New Zealand, pages 101-121.

[34] Bies, A., Ferguson, M., Katz, K., and MacIntyre, R. (1995). *Bracketing guidelines for Treebank II style Penn Treebank project.* Technical report, University of Pennsylvania.

[35] Dalrymple, M. (1999), *Semantics and syntax in Lexical Functional Grammar: The Resource Logic Approach*. Cambridge Mass, MIT Press.

APPENDIX A

Penn Treebank POS Tag-set

Tag	Description
$	dollar
``	opening quotation mark
"	closing quotation mark
(	opening parenthesis
)	closing parenthesis
,	comma
--	dash
.	sentence terminator
:	colon or ellipsis
CC	conjunction, coordinating
CD	numeral, cardinal
DT	determiner
EX	existential there
FW	foreign word
IN	preposition or conjunction, subordinating
JJ	adjective or numeral, ordinal
JJR	adjective, comparative
JJS	adjective, superlative
LS	list item marker
MD	modal auxiliary
NN	noun, common, singular or mass
NNP	noun, proper, singular
NNPS	noun, proper, plural
NNS	noun, common, plural
PDT	pre-determiner
POS	genitive marker
PRP	pronoun, personal
PRP$	pronoun, possessive
RB	adverb
RBR	adverb, comparative
RBS	adverb, superlative
RP	particle
SYM	symbol
TO	to as preposition or infinitive marker
UH	interjection
VB	verb, base form
VBD	verb, past tense
VBG	verb, present participle or gerund

VBN	verb, past participle
VBP	verb, present tense, not 3rd person singular
VBZ	verb, present tense, 3rd person singular
WDT	WH-determiner
WP	WH-pronoun
WP$	WH-pronoun, possessive
WRB	Wh-adverb

APPENDIX B

Syntax of Annotation Rules

```
Lhs  →   Rhs ‘@ [’ Anno1 ‘]’ ( ‘@ [’ Anno2 ‘]’) .

Lhs  → Symbols ‘:’ f_var

     Symbols → CFG_NT_Symbol | Symbols ‘,’ CFG_NT_Symbol

Rhs  → ‘*’ | ‘(’ Rhs ‘)’ | RhsStatement

Rhs  → Rhs Rhs

Rhs  → ‘[’ RhsStatement ‘|’ RhsStatementsRec ‘]’

     RhsStatementsRec    → RhsStatement

     RhsStatementsRec    → RhsStatement '|' RhsStatementsRec

     RhsStatement        → CFG_NT_Symbol ‘:’ f_var

Anno1→ LFGAnnotation ( ‘,’ Anno1 )

     LFGAnnotation → f_var ‘:’ LFGRelation ‘===’ f_var (‘:’
                                              LFGRelation )

Anno2→ f_var ‘:’ LFGAttribute ( ‘,’ Anno2 )
```

APPENDIX C

Templates

Following are the Templates used by Annotation System.

ADJP:adjp	>	* [JJ:a1|ADJP:a1] (CC:c1) [JJ:a2|ADJP:a2] * @ [a2:$===a1,a2:CONJ_FORM===c1,adjp:^===a2] .
ADJP:adjp	>	* [RB:r1|RBR:r1] * [VBG:v1|VBN:v1|JJ:v1] @ [v1:ADJUNCT ADV$===r1,adjp:^===v1] .
ADJP:adjp	>	* [RBR:r1|RB:r1|RBS:r1] * @ [adjp:^ADJUNCT ADV$===r1] .
ADJP:adjp	>	* [VBG:v1|VBN:v1|JJ:v1] PP:p1 @ [v1:ADJUNCT PREP$===p1,adjp:^===v1] .
ADJP:adjp	>	* ADJP:a1 * PP:p1 * @ [a1:ADJUNCT PREP$===p1,adjp:^===a1] .
ADJP:adjp	>	* CC:c1 * JJ:j2 @ [j2:CONJ_FORM===c1,adjp:^===j2] .
ADJP:adjp	>	* CD:c1 * [NN:n1|NNS:n1|NNP:n1|NNPS:n1] * PRN:p1 * @ [n1:ADJUNCT MOD$===p1,n1:SPEC CARD$===c1,adjp:^===n1] .
ADJP:adjp	>	* CD:c1 * [NN:n1|NNS:n1|NNP:n1|NNPS:n1] @ [n1:SPEC CARD$===c1,adjp:^===n1] .
ADJP:adjp	>	* JJ:j1 * [SG:s1|SG-A:s1] * @ [j1:XCOMP===s1,adjp:^===j1] @ [j1:^XCOMP SUBJ PRED = 'pro',j1:^XCOMP SUBJ PRONTYPE = NULL] .
ADJP:adjp	>	* JJ:j1 * PP:p1 @ [j1:ADJUNCT PREP$===p1,adjp:^===j1] .
ADJP:adjp	>	* JJ:j1 * SBAR:s1 * @ [j1:COMP===s1,adjp:^===j1] .
ADJP:adjp	>	* JJS:j1 * VBN:v1 @ [v1:$===j1,adjp:^===v1] .
ADJP:adjp	>	* NP:n1 * [JJ:j1|JJR:j1] @ [j1:ADJUNCT MOD$===n1,adjp:^===j1] .
ADJP:adjp	>	* RB:r1 * [JJ:j1|JJR:j1] @ [j1:ADJUNCT ADV$===r1,adjp:^===j1] .
ADJP:adjp	>	* VBN:j1 (CC:c1) VBN:j2 @ [j1:CONJ_FORM===c1,j1:$===j2,adjp:^===j1] .

```
ADJP:adjp   >   * VBN:v1 CC:c1 JJ:j1 @
                [j1:$===v1,j1:CONJ_FORM===c1,adjp:^===j1]
                .
ADJP:adjp   >   ADJP:a1 SBAR:s1 @
                [a1:COMP===s1,adjp:^===a1] .
ADJP:adjp   >   ADVP:a1 SBAR:s1 @
                [a1:COMP$===s1,adjp:^===a1] .
ADJP:adjp   >   DT:d1 QP:q1 @ [q1:SPEC
                DET$===d1,adjp:^===q1] .
ADJP:adjp   >   JJ:j1 NN:n1 JJ:j2 @
                [j2:$===j1,j2:$===n1,adjp:^===j2] .
ADJP:adjp   >   JJR:j1 CC:c1 JJR:j2 @
                [j2:CONJ_FORM===c1,j2:$===j1,adjp:^===j2]
                .
ADJP:adjp   >   RB:r1 JJ:j1 RB:r2 @ [j1:ADJUNCT
                ADV$===r1,j1:ADJUNCT
                ADV$===r2,adjp:^===j1] .
ADJP:adjp   >   RBS:r1 JJ:j1 @ [j1:ADJUNCT
                ADV$===r1,adjp:^===j1] .
ADVP:advp   >   * [RB:r1|ADVP:r1] * PP:p1 * @
                [r1:ADJUNCT PREP$===p1,advp:^===r1] .
ADVP:advp   >   * ADVP:r1 (CC:c1) PP:p1 * @
                [r1:CONJ_FORM===c1,r1:$===p1,advp:^===r1]
                .
ADVP:advp   >   * ADVP:r1 * SBAR:s1 * @ [r1:ADJUNCT
                SBAR===s1,advp:^===r1] .
ADVP:advp   >   * CC:c1 * RB:r1 @
                [r1:CONJ_FORM===c1,advp:^===r1] .
ADVP:advp   >   * NP:n1 * [RBR:r1|RB:r1] @
                [r1:SPEC===n1,advp:^===r1] .
ADVP:advp   >   * NPB:n1 IN:i1 PP:p1 * @
                [i1:OBJ===p1,n1:ADJUNCT
                PREP$===i1,advp:^===n1] .
ADVP:advp   >   * RB:r1 * RB:r2 * @
                [r2:$===r1,advp:^===r2] .
ADVP:advp   >   * RB:r1 * VBN:v1 @ [v1:ADJUNCT
                ADV$===r1,advp:^===r1] .
ADVP:advp   >   * RB:r1 CC:c1 * RB:r2 * @
                [r1:CONJ_FORM===c1,r1:$===r2,advp:^===r1]
                .
ADVP:advp   >   RB:r1 NP:n1 @ [n1:ADJUNCT
                ADV$===r1,advp:^===n1] .
CONJP:conjp >   * RB:r1 RB:r2 IN:i1 * @
                [conjp:^===r1,r1:ADJUNCT
                ADV$===r2,r1:ADJUNCT $===i1] .
NAC:nac     >   * NNP:n1 * PP:p1 * @ [n1:ADJUNCT
```

		PREP$===p1,nac:^===n1] .
NP,NP-A,NPB:np	>	* PP:p1 * @ [np:^ ADJUNCT PREP$===p1] .
NP,NP-A:np	>	* [NPB:n1|NP:n1] (CC:c1)(*) [NP:n2|NPB:n2] @ [n2:$===n1,n2:CONJ_FORM===c1,np:^===n2] .
NP,NP-A:np	>	* [NPB:n1|NP:n1] * [SBAR:s1|SBAR-g:s1] * @ [n1:ADJUNCT REL_CL===s1,np:^===n1] .
NP,NP-A:np	>	* NPB:n1 * PP:p1 * @ [n1:ADJUNCT PREP$===p1,np:^===n1] .
NP,NP-A:np	>	* PP:p1 * PP:p2 * @ [np:^ ADJUNCT PREP$===p1,np:^ ADJUNCT PREP$===p2] .
NP:np	>	* [JJ:j1|JJR:j1] * [NN:n1|NNS:n1|NNP:n1|NNPS:n1] * @ [n1:ADJUNCT ADJ$===j1,np:^===n1] .
NP:np	>	* [JJ:j1|JJR:j1] * [NN:n1|NNS:n1|NNP:n1|NNPS:n1] @ [n1:ADJUNCT ADJ$===j1,np:^===n1] .
NP:np	>	* [NN:n1|NNS:n1|NNP:n1|NNPS:n1|NAC:n1] * [NN:n2|NNS:n2|NNP:n2|NNPS:n2] * @ [n2:$===n1,np:^===n2] .
NP:np	>	* [NP:n1|NPB:n1] * NP:n2 @ [n2:$===n1,np:^===n2] .
NP:np	>	* [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1] * NN:n2 @ [n2:ADJUNCT ADJ$===v1,np:^===n2] .
NP:np	>	* CD:c1 * [NN:n1|NNS:n1|NNP:n1|NNPS:n1] @ [n1:SPEC CARD$===c1,np:^===n1] .
NP:np	>	* DT:d1 * [NN:n1|NNS:n1|NNP:n1|NNPS:n1] @ [n1:SPEC DET$===d1,np:^===n1] .
NP:np	>	* NNP:n1 * [NN:n2|NNS:n2|NNP:n2|NNPS:n2] @ [n2:$===n1,np:^===n2] .
NP:np	>	NP:n1 * NP:n2 @ [n2:$===n1,np:^===n2] .
NP:np	>	NP:n1 CONJP:c1 @ [n1:CONJ_FORM===c1,np:^===n2] .
NP-A,NP,NPB:np-a	>	* [NP:n1|NPB:n1|NN:n1|NNS:n1|NNP:n1|NNPS:n1] * [SG:s1|VP:s1|VBZ:s1] * @ [n1:ADJUNCT PARTICIPLE===s1,np-a:^===n1] .
NP-A,NP:np-a	>	* NPB:n1 CONJP:c1 NPB:n2 @ [n2:$===n1,n2:CONJ_FORM===c1,np-a:^===n2] .
NP-A:np-a	>	NPB:n1 ADVP:a1 @ [n1:ADJUNCT ADV===a1,np-a:^===n1] .
NP-A:np-a	>	* [NP:n1|NPB:n1] * ADJP:a1 * @ [n1:ADJUNCT ADJ$===a1,np-a:^===n1] .

NP-A:np-a	>	* [NPB:n1|NP:n1] * NP:n2 * @ [n2:$===n1,np-a:^===n2] .
NP-A:np-a	>	* NP:n1 (CC:c1) [NPB:n2|NP:n2] @ [n2:$===n1,n2:CONJ_FORM===c1,np-a:^===n2] .
NP-A:np-a	>	* NP:n1 * UCP:u1 * @ [n1:ADJUNCT UCP$===u1,np-a:^===n1] .
NP-A:np-a	>	* NPB:n1 * [SBAR:s1|SBAR-g:s1] * @ [n1:ADJUNCT REL_CL===s1,np-a:^===n1] .
NP-A:np-a	>	* NPB:n1 * PP:p1 * @ [n1:ADJUNCT PREP$===p1,np-a:^===n1] .
NP-A:np-a	>	* NPB:n1 * PRN:p1 * @ [n1:ADJUNCT PRN$===p1,np-a:^===n1] .
NP-A:np-a	>	* NPB:n1 * RRC:r1 @ [n1:ADJUNCT REL_CL===r1,np-a:^===n1] .
NP-A:np-a	>	NPB:n1 * ADVP:a1 @ [n1:ADJUNCT ADV$===a1,np-a:^===n1] .
NP-A:np-a	>	NPB:n1 NP:n2 @ [n1:ADJUNCT NOUN_MOD===n2,np-a:^===n1] .
NPB,NP-A:npb	>	* [ADVP:a1|RB:a1] * [NP:n1|NPB:n1] @ [n1:ADJUNCT ADV$===a1,npb:^===n1] .
NPB,NP-A:npb	>	* PRPS:p1 * [NN:n1|NNS:n1|NNP:n1|NNPS:n1|NP:n1] * @ [n1:$===n2,n1:SPEC DET GEN_PRO===p1,npb:^===n1] @ [n1:^SPEC DET DEF=POS,n1:^SPEC DET DTYPE=gen_pro] .
NPB:npb	>	* (DT:d1 *) (JJ:j1 *) ([NN:n2|NNS:n2|NNP:n2|NNPS:n2] *) [NN:n1|NNS:n1|NNP:n1|NNPS:n1] POS:p1 * @ [n1:ADJUNCT ADJ$===j1,n1:SPEC DET ===d1,n1:ADJUNCT TITLE===n2,n1: SPEC DET GEN_PRO===p2,npb:^===p1,npb:^ GENITIVE===n1] @ [n1:^ CASE=GEN,n1:^ DEF=POS,n1:^ DTYPE=genitive] .
NPB:npb	>	* (DT:d1 *) [NN:n1|NNS:n1|NNP:n1|NNPS:n1] POS:p1 * @ [n1: SPEC DET===d1,npb:^===p1,npb:^GENITIVE===n1] @ [n1:^ CASE=GEN,n1:^ DEF=POS,n1:^ DTYPE=genitive] .
NPB:npb	>	* (PRPS:p2 *) [NN:n1|NNS:n1|NNP:n1|NNPS:n1] POS:p1 * @ [n1: SPEC DET GEN_PRO===p2,npb:^===p1,npb:^GENITIVE===n1] @ [n1:^ CASE=GEN,n1:^ DEF=POS,n1:^ DTYPE=genitive] .

```
NPB:npb     >   * (PRPS:p2 *)
                [NN:n2|NNS:n2|NNP:n2|NNPS:n2|NP:n2] *
                [NN:n1|NNS:n1|NNP:n1|NNPS:n1] POS:p1 * @
                [n1:ADJUNCT TITLE===n2,n1: SPEC DET
                GEN_PRO===p2,npb:^===p1,npb:^
                GENITIVE===n1] @ [n1:^ CASE=GEN,n1:^
                DEF=POS,n1:^ DTYPE=genitive] .
NPB:npb     >   * [ADVP:a1|RB:a1] [QP:q1|CD:q1] @
                [q1:ADJUNCT ADV$===a1,npb:^===q1] .
NPB:npb     >   * [JJR:j1|JJS:j1] *
                [NN:n1|NNS:n1|NNP:n1|NNPS:n1] @
                [n1:ADJUNCT ADJ$===j1,npb:^===n1] .
NPB:npb     >   * [NN:n1|NNS:n1|NNP:n1|NNPS:n1] *
                [NN:n2|NNS:n2|NNP:n2|NNPS:n2] * @
                [n2:$===n1,npb:^===n2] .
NPB:npb     >   * [NN:n1|NNS:n1|NNP:n1|NNPS:n1] CC:c1 *
                [NN:n2|NNS:n2|NNP:n2|NNPS:n2] @
                [n2:CONJ_FORM===c1,n2:$===n1,npb:^===n2]
                .
NPB:npb     >   * [NN:n1|NNS:n1|NNP:n1|NNPS:n1] CC:c1
                [NN:n2|NNS:n2|NNP:n2|NNPS:n2] @
                [n2:$===n1,n2:CONJ_FORM===c1,npb:^===n2]
                .
NPB:npb     >   * [NN:n1|NNS:n1|NNP:n1|NNPS:n1] RB:r1 @
                [n1:ADJUNCT ADV$===r1,npb:^===n1] .
NPB:npb     >   * [NPB:n1|NP:n1] *
                [NN:n2|NNS:n2|NNP:n2|NNPS:n2] @ [n2:SPEC
                DET===n1,npb:^===n2] .
NPB:npb     >   * ADJP:a1 *
                [NN:n1|NNS:n1|NNP:n1|NNPS:n1] @
                [n1:ADJUNCT ADJ$===a1,npb:^===n1] .
NPB:npb     >   * CD:c1 * [NN:n1|NNS:n1|NNP:n1|NNPS:n1]
                @ [n1:SPEC CARD$===c1,npb:^===n1] .
NPB:npb     >   * DT:d1 * [NN:n1|NNS:n1|NNP:n1|NNPS:n1]
                @ [n1:SPEC DET$===d1,npb:^===n1] .
NPB:npb     >   * DT:d1 * VBG:v1 @ [v1:SPEC
                DET$===d1,npb:^===v1] .
NPB:npb     >   * DT:d1 @ [npb:^===d1] .
NPB:npb     >   * DT:j1 * CD:c1 @ [c1:ADJUNCT
                ADJ$===j1,npb:^===c1] .
NPB:npb     >   * IN:i1 * [NN:n1|NNS:n1|NNP:n1|NNPS:n1]
                @ [npb:^===n1,n1:PHY_FORM===i1:PHY_FORM]
                .
NPB:npb     >   * IN:i1 DT:d1 JJ:j1 @ [j1:SPEC
                DET$===d1,npb:^===i1,i1:OBJ===j1] .
NPB:npb     >   * IN:i1 DT:d1 JJ:j1 @ [j1:SPEC
```

```
                DET$===d1,npb:^===i1,i1:OBJ===j1] .
NPB:npb     >    * JJ:j1 (CC:c1) JJ:j2 *
                [NN:n1|NNS:n1|NNP:n1|NNPS:n1] @
                [n1:ADJUNCT ADJ CONJFORM===c1,n1:ADJUNCT
                ADJ$===j1,n1:ADJUNCT
                ADJ$===j2,npb:^===n1] .
NPB:npb     >    * JJ:j1 * [NN:n1|NNS:n1|NNP:n1|NNPS:n1]
                @ [n1:ADJUNCT ADJ$===j1,npb:^===n1] .
NPB:npb     >    * JJ:j1 * [NN:n1|NNS:n1|NNP:n1|NNPS:n1]
                POS:p1 @ [n1:ADJUNCT
                ADJ$===j1,npb:^===p1,npb:^GENITIVE===n1]
                @ [n1:^ CASE=GEN,n1:^ DEF=POS,n1:^
                DTYPE=genitive] .
NPB:npb     >    * JJ:j1 * CD:c1 @ [c1:ADJUNCT
                ADJ$===j1,npb:^===c1] .
NPB:npb     >    * JJ:j1 [NN:n1|NNS:n1|NNP:n1|NNPS:n1] *
                @ [n1:ADJUNCT ADJ$===j1,npb:^===n1] .
NPB:npb     >    * NAC:n1 * [NN:n2|NNS:n2|NNP:n2|NNPS:n2]
                @ [n2:ADJUNCT NAC$===n1,npb:^===n2] .
NPB:npb     >    * NPB:n1 (CC:c1) NPB:n2 @
                [n2:$===n1,n2:CONJ_FORM===c1,npb:^===n2]
                .
NPB:npb     >    * NPB:n1 * [NN:n2|NNS:n2|NNP:n2|NNPS:n2]
                @ [n2:SPEC DET===n1,npb:^===n2] .
NPB:npb     >    * PDT:p1 *
                [NN:n1|NNS:n1|NNP:n1|NNPS:n1|NAC:n1] @
                [n1:SPEC PRE-DET===p1,npb:^===n1] .
NPB:npb     >    * QP:q1 * [NN:n1|NNS:n1|NNP:n1|NNPS:n1]
                @ [n1:SPEC QUANT$===q1,npb:^===n1] .
NPB:npb     >    * RB:r1 * @ [npb:^ADJUNCT ADV$===r1] .
NPB:npb     >    * RB:r1 * [JJ:j1|JJS:j1] @ [j1:ADJUNCT
                ADV$===r1,npb:^===j1] .
NPB:npb     >    * RB:r1 * [NN:n1|NNS:n1|NNP:n1|NNPS:n1]
                @ [n1:ADJUNCT ADV$===r1,npb:^===n1] .
NPB:npb     >    * VBG:v1 * [NN:n1|NNS:n1|NNP:n1|NNPS:n1]
                @ [n1:ADJUNCT PARTICIPLE$===v1,np:^===n1]
                .
NPB:npb     >    * VBG:v1 * [NN:n1|NNS:n1|NNP:n1|NNPS:n1]
                @ [npb:^ ADJUNCT PARTICIPLE===v1] .
NPB:npb     >    * VBN:v1 * [NN:n1|NNS:n1|NNP:n1|NNPS:n1]
                @ [n1:ADJUNCT
                PARTICIPLE$===v1,npb:^===n1] .
NPB:npb     >     CD:c1 NNP:n1 CD:c2 @ [c2:$===c1,
                c2:$===n1, npb:^===c2] @ [c2:^DATE=Yes] .
NPB:npb     >     CD:c2 * CD:c1 @ [c1:$===c2,npb:^===c1] .
NPB:npb     >     DLR:d1 CD:c1 @
```

		[c1:CURRENCY===d1,npb:^===c1] @ [c1:^DOLLER = TRUE] .
NPB:npb	>	DT:d1 @ [npb:^===d1] .
NPB:npb	>	DT:d1 JJ:j1 @ [j1:SPEC DET===d1,npb:^===j1] .
NPB:npb	>	DT:d1 JJS:j1 @ [j1:SPEC DET===d1,npb:^===j1] .
NPB:npb	>	NN:n1 CD:c1 @ [n1:SPEC QUANT===c1,npb:^===n1] .
NPB:npb	>	NNP:n1 CD:c1 @ [n1:SPEC CARD$===c1,npb:^===n1] .
NPB:npb	>	QP:q1 [NN:n1|NNS:n1|NNP:n1|NNPS:n1] POS:p1 @ [n1:SPEC QUANT===q1,n1: SPEC DET GEN_PRO===p2,npb:^===p1,npb:^GENITIVE===n1] @ [n1:^ CASE=GEN,n1:^ DEF=POS,n1:^ DTYPE=genitive] .
NPB:npb	>	RBR:r1 JJ:j1 [NN:n1|NNS:n1|NNP:n1|NNPS:n1] @ [j1:ADJUNCT ADV$===r1,n1:ADJUNCT ADJ===j1,npb:^===n1] .
NP-PRD:np-prd	>	* NP:n1 * PP:p1 @ [n1:ADJUNCT PREP$===p1,np-prd:^===n1] .
NP-TMP:np-tmp	>	* NNP:n1 * CD:c1 * @ [n1:SPEC CARD$===c1,np-tmp:^===n1] .
PP,PP-A:pp	>	(RB:r1) * [IN:i1|TO:i1] * [SG:s1|SG-A:s1|S-A:s1] * @ [s1:ADJUNCT ADV$===r1,s1:CONJ_FORM===i1:PFORM,pp:^===s1] .
PP:pp	>	* [IN:i1|TO:i1] * [NPB:n1|NP-A:n1|ADJP-A:n1] * @ [i1:OBJ===n1,pp:^===i1] .
PP:pp	>	* ADVP:a1 IN:i1 * @ [i1:ADJUNCT ADV$===a1,pp:^===i1] .
PP:pp	>	* IN:i1 * [PP:p1|PP-A:p1] * @ [i1:ADJUNCT PREP===p1,pp:^===i1] .
PP:pp	>	* IN:i1 * SBAR-A:s1 @ [s1:CONJ_FORM===i1:PFORM,pp:^===s1] .
PP:pp	>	* IN:i1 IN:i2 NP-A:n1 * @ [i2:OBJ===n1,i1:ADJUNCT PREP===i2,pp:^===i1] .
PP:pp	>	* JJ:j1 * [IN:i1|TO:i1] * @ [i1:ADJUNCT ADJ$===j1,pp:^===i1]
PP:pp	>	* PP:p1 * @ [pp:^ $===p1] .
PP:pp	>	* PP:p1 * @ [pp:^===p1] .
PP:pp	>	* PP:p1 CC:c1 PP:p2 @ [p2:$===p1,p2:CONJ_FORM===c1,pp:^===p2] .
PP:pp	>	* PUNC:p1 * @ [pp:ADJUNCT$===p1] .

```
PP:pp       >   RB:r1 * PP:p1 * @ [p1:ADJUNCT ADV$===r1]
                .
PP:pp       >   vbg:v1  PP-A:p1 @
                [v1:OBJ===p1,pp:^===v1] .
PP:pp       >   VBG:v1 * NP-A:n1 @
                [v1:OBJ===n1,pp:^===v1] .
PP:pp       >   VBG:v1 * PP-A:p1 * @
                [v1:COMP$===p1,pp:^===v1] .
PP-A:pp-a   >   [IN:i1|TO:i1] * [NP-
                A:n1|NP:n1|NPB:n1|PP-A:n1] * @
                [i1:OBJ===n1,pp-a:^===i1] .
PP-CLR:pp-  >   IN:i1 NP:n1 * @ [i1:OBJ===n1,pp-
clr             clr:^===i1] .
PRN:prn     >   * [LRB:l1|RRB:l1] * @
                [prn:^ADJUNCT$===l1] .
PRN:prn     >   * NP:n1 * @ [prn:^===n1] .
QP:qp       >   * [IN:i1|RB:i1] * CD:c1 @ [c1:ADJUNCT
                ADV$===i1,qp:^===c1] .
QP:qp       >   * [JJS:j1|JJR:j1] * CD:c1 @ [c1:ADJUNCT
                ADJ===j1,qp:^===c1] .
QP:qp       >   * CD:c1 (CC:c3) CD:c2 @
                [c2:$===c1,c2:CONJ_FORM===c3,qp:^===c2] .
QP:qp       >   * CD:c1 @ [qp:^===c1] .
QP:qp       >   * DLR:d1 * CD:c1 * @ [qp:^
                CURRENCY===d1,qp:^$===c1] @ [c1:^ DOLLAR
                = TRUE] .
QP:qp       >   * DLR:d1 * CD:c1 @ [c1:^
                CURRENCY===d1,qp:^===c1] @ [c1:^ DOLLAR =
                TRUE].
QP:qp       >   * IN:i1 PDT:p1 @ [p1:COMPARITIVE
                COMP_FORM===i1,qp:^===p1] .
QP:qp       >   * IN:i1 TO:t1 DLR:d1 * CD:c1 * @
                [qp:_===i1,qp:_===t1,qp:^
                CURRENCY===d1,qp:^$===c1] @ [c1:^ DOLLER
                = TRUE] .
RRC:rrc     >   * ADVP:a1 * PP:p1 @ [p1:ADJUNCT
                ADV$===a1,rrc:^===p1] .
S,S-A:s     >   * [NP:n1|NP-A:n1] * NP-A:n2 * VP:v1 * @
                [n2:ADJUNCT MOD$===n1,s:^
                SUBJ===n2,s:^===v1] @
                [v1:^CLAUSE_TYPE=DECLARATIVE] .
S,S-A:s     >   * [SBAR:s1|SBAR-A:s1] * VP:v1 * @
                [v1:COMP$===s1,s:^===v1] @
                [v1:^CLAUSE_TYPE=DECLARATIVE] .
S:s         >   * [ADVP:a1|INTJ:a1] * VP:v1 * @
                [v1:ADJUNCT S_ADV$===a1,s:^===v1] @
```

```
                [v1:^CLAUSE_TYPE=DECLARATIVE] .
S:s         >    * [LRB:l1|RRB:l1] * @ [s:^ADJUNCT$===l1]
                .
S:s         >    * [NPB:n1|NP:n1] [NPB:n2|NP-A:n2] *
                VP:v1 * @
                [v1:SUBJ===n1,n1:SPEC$===n2,s:^===v1] @
                [v1:^CLAUSE_TYPE=DECLARATIVE] .
S:s         >    * [S-A:s1|S:s1] * VP:v1 * @
                [v1:COMP$===s1,s:^===v1] @
                [v1:^CLAUSE_TYPE=DECLARATIVE] .
S:s         >    * CC:c1 * VP:v1 * @
                [v1:CONJ_FORM===c1,s:^===v1] @
                [v1:^CLAUSE_TYPE=DECLARATIVE] .
S:s         >    * CC:c1 S:s1 S:s2 * @ [s:CONJ_FORM===c1]
                .
S:s         >    * IN:i1 NP-A:n1 VP:v1 * @
                [v1:SUBJ===n1,v1:CONJ_FORM===i1,s:^===v1]
                .
S:s         >    * NP-A:n1 * VP:v1 * @
                [v1:SUBJ===n1,s:^===v1] @
                [v1:^CLAUSE_TYPE=DECLARATIVE] .
S:s         >    * NP-A:n2 * NP:n1 * VP:v1 * @
                [n2:ADJUNCT MOD$===n1,s:^
                SUBJ===n2,s:^===v1] @
                [v1:^CLAUSE_TYPE=DECLARATIVE] .
S:s         >    * NPB:n1 * VP:v1 * @
                [v1:SUBJ===n1,s:^===v1] @
                [v1:^CLAUSE_TYPE=DECLARATIVE] .
S:s         >    * PP:p1 * VP:v1 * @ [v1:ADJUNCT
                S_PREP$===p1,s:^===v1] @
                [v1:^CLAUSE_TYPE=DECLARATIVE] .
S:s         >    * RRB:r1 * @ [s:^ ADJUNCT ADV$===r1] .
S:s         >    * S:s1 (CC:c1) S:s2 * @
                [s2:CONJ_FORM===c1,s2:$===s1,s:^===s2] .
S:s         >    * SG:s1 * VP:v1 * @
                [v1:XADJUNCT===s1,s:^===v1] @
                [v1:^CLAUSE_TYPE=DECLARATIVE,v1:^XADJUNCT
                SUBJ PRED = 'pro',v1:^XADJUNCT SUBJ
                PRONTYPE = NULL] .
S:s         >    NP:n1 SG:s1 @ [n1:ADJUNCT
                PARTICIPAL===s1,s:^===n1] .
S:s         >    NP-A:n1 ADJP:a1 @ [n1:ADJUNCT
                ADJ===a1,s:^===n1] .
S-A:s-a     >    * [NPB:n1|NP-A:n1] * VP:v1 * @
                [v1:SUBJ===n1,s-a:^===v1] @
                [v1:^CLAUSE_TYPE=DECLARATIVE] .
```

```
S-A:s-a     >   * ADVP:a1 * VP:v1 * @ [v1:ADJUNCT
                S_ADV$===a1,s-a:^===v1] @
                [v1:^CLAUSE_TYPE=DECLARATIVE] .
S-A:s-a     >   * CC:c1 * @ [s-a:^CONJ_FORM===c1] .
S-A:s-a     >   * MD:m1 * VP:v1 * @ [s-a:^===m1,s-
                a:^===v1] @ [v1:^CLAUSE_TYPE=DECLARATIVE]
                .
S-A:s-a     >   * NBP:n1 NP-A:n2 @ [n1:ADJUNCT
                MOD$===n2,s-a:^===n1] .
S-A:s-a     >   * NP_A:n1 (ADJP:a1) SBAR:s1 * @
                [s1:SUBJ===n1,s1:ADJUNCT S__ADJ$===a1,s-
                a:^===s1] .
S-A:s-a     >   * NP-A:n1 ADVP:a1 @ [n1:ADJUNCT
                S_ADV$===a1,s-a:^===n1] .
S-A:s-a     >   * PP:p1 * @ [s-a:ADJUNCT S_PREP===p1] .
S-A:s-a     >   * PP:p1 * VP:v1 * @ [v1:ADJUNCT
                S_PREP$===p1,s-a:^===v1] @
                [v1:^CLAUSE_TYPE=DECLARATIVE] .
S-A:s-a     >   * S:s1 * S:s2 * @ [s2:$===s1,s-a:^===s2]
                .
S-A:s-a     >   * S-A:s1 * VP:v1 * @ [v1:COMP$===s1,s-
                a:^===v1] .
S-A:s-a     >   [NBP:n1|NP-A:n1] NP-A:n2 @ [n2:$===n1,s-
                a:^===n2] .
S-A:s-a     >   S:s1 * S:s2 @
                [s2:$===s1,s2:CONJ_FORM===c1,s-a:^===s2]
                .
SBAR,S-     >   * ADVP:a1 * @ [sbar:^ADJUNCT
A:sbar          S_ADV$===a1] .
SBAR,SBARQ: >   * WHNP:w1 * [SG-A:s1|SQ:s1|S-A:s1] * @
sbar            [s1:SUBJ/OBJ===w1,sbar:^===s1] @ [w1:^
                WHQ = POS] .
SBAR:sbar   >   * IN:i1 * NN:n1 [S-A:s1|SG-A:s1] * @
                [i1:OBJ===n1,i1:COMP$===s1,sbar:^===i1] .
SBAR:sbar   >   * IN:i1 * SINV:s1 * @
                [s1:CONJ_FORM===i1,sbar:^===s1] .
SBAR:sbar   >   * IN:i1 [S-A:s1|SG-A:s1] * @
                [s1:CONJ_FORM===i1,sbar:^===s1] .
SBAR:sbar   >   * RB:r1 * IN:i1 * @ [i1:ADJUNCT
                ADV$===r1,sbar:^===i1] .
SBAR:sbar   >   * SBAR:s1 (CC:c1) SBAR:s2 * @
                [s2:$===s1,s2:CONJ_FORM===c1,sbar:^===s2]
                .
SBAR:sbar   >   * WHADVP:w1 * [S-A:s1|SG-A:s1] * @
                [s1:ADJUNCT S_ADV$===w1,sbar:^===s1] @
                [w1:^ WHQ = POS] .
```

```
SBAR-A:sbar >  SBAR:s1 CC:c1 SBAR:s2 @
               [s2:$===s1,s2:CONJ_FORM===c1,sbar:^===s2]
               .
SBAR-          >  * IN:i1 * S-A:s1 * @
A:sbar-a       [s1:CONJ_FORM===i1:PFORM,sbar-a:^===s1] .
SBAR-          >  * WHADVP:w1 * S-A:s1 * @ [s1:ADJUNCT
A:sbar-a       S_ADV$===w1,sbar-a:^===s1] .
SBAR-          >  * WHNP:w1 * [SG-A:s1|S-A:s1] * @
A:sbar-a       [s1:SUBJ/OBJ===w1,sbar:^===s1] .
SBAR-A-        >  * WHNP:w1 * S-A-g:s1 * @
g:sbar         [s1:SUBJ/OBJ===w1,sbar:^===s1] .
SBAR-g:sbar >   * WHNP:w1 * S-A-g:s1 * @
               [s1:SUBJ/OBJ===w1,sbar:^===s1] .
SBAR-          >   * WHNP:w1 * S-A-G:s1 * @
G:sbar-g       [s1:SUBJ/OBJ===w1,sbar-g:^===s1] .
SBAR-          >   * WHNP:w1 * S-A-G:s1 * @
G:sbar-g       [s1:SUBJ/OBJ===w1,sbar-g:^===s1] @ [w1:^
               WHQ = POS] .
SBAR-          >   WHNP:w1  S-A-g:s1 @
G:sbar-g       [s1:SUBJ/OBJ===w1,sbar-g:^===s1] @ [w1:^
               WHQ = POS] .
SBARQ:sbarq >  * [SBAR-ADV:r1|RB:r1|INTJ:r1|ADVP:r1] *
               SQ:s1 * @ [s1:ADJUNCT
               S_ADV$===r1,sbarq:^===s1] @
               [s1:^CLAUSE_TYPE=INTERROGATIVE] .
SBARQ:sbarq >  * CC:c1 * SBARQ:s1 * @
               [s1:CONJ_FORM===c1,sbarq:^===s1] .
SBARQ:sbarq >  * PRN:p1 * SQ:s1 * @ [s1:ADJUNCT
               PRN$===p1,sbarq:^===s1] @
               [s1:^CLAUSE_TYPE=INTERROGATIVE] .
SBARQ:sbarq >  * SBARQ:s1 * SBARQ:s2 * @
               [s2:$===s1,sbarq:^===s2] .
SBARQ:sbarq >  * WHADVP:w1 * SQ:s1 * @ [s1:ADJUNCT
               S_ADV$===w1,sbarq:^===s1] @ [w1:^ WHQ =
               POS,s1:^CLAUSE_TYPE=INTERROGATIVE] .
SBARQ:sbarq >  * WHNP:w1 * [SQ:s1|S:s1] * @
               [s1:SUBJ===w1,sbarq:^===s1] @
               [s1:^CLAUSE_TYPE=INTERROGATIVE,w1:^ WHQ =
               POS] .
SBARQ:sbarq >  * WHNP:w1 * SQ:s1 * @
               [s1:OBJ===w1,sbarq:^===s1] @
               [s1:^CLAUSE_TYPE=INTERROGATIVE] @ [w1:^
               WHQ = POS] .
SBARQ:sbarq >  * WHPP:w1 * SQ:s1 * @ [s1:ADJUNCT
               S_PREP$===p1,sbarq:^===s1] @
               [s1:^CLAUSE_TYPE=INTERROGATIVE] .
```

SBARQ:sbarq	>	* WRB:w1 * FRAG:f1 * @ [f1:ADJUNCT S_ADV$===w1,sbarq:^===f1] @ [v1:^CLAUSE_TYPE=INTERROGATIVE] .
SBARQ-TPC,SBARQ:sbarq-tpc	>	* WHNP:w1 * [SQ:s1|S:s1] * @ [s1:SUBJ===w1,sbarq-tpc:^===s1] @ [w1:^ WHQ = POS] .
SG:sg	>	(CC:c1) * ADVP:a1 * VP:v1 * @ [v1:CONJ_FORM===c1,v1:ADJUNCT S_ADV$===a1,sg:^===v1] .
SINV:sinv	>	(PP:p1) VBZ:v2 NP:n1 VP:v1 @ [v1:ADJUNCT S_PREP$===p1,v1:SUBJ===n1,v1:TNS_ASP===v2:TNS_ASP,sinv:^===v1] .
SINV:sinv	>	(S:s1|*) VP:v1 NP:n1 * @ [v1:COMP===s1,v1:SUBJ===n1,sinv:^===v1] @ [v1:^CLAUSE_TYPE=DECLARATIVE] .
SINV:sinv	>	* (S:s1) VP:v1 NP:n1 * @ [v1:COMP===s1,v1:SUBJ===n1,sinv:^===v1] @ [v1:^CLAUSE_TYPE=DECLARATIVE] .
SINV:sinv	>	* ADVP:a1 * VP:v1 * @ [v1:ADJUNCT S_ADV$===a1,sinv:^===v1] @ [v1:^CLAUSE_TYPE=DECLARATIVE] .
SINV:sinv	>	* md:m1 * VP:v1 * @ [sinv:^===m1,sinv:^===v1] @ [v1:^CLAUSE_TYPE=DECLARATIVE] .
SINV:sinv	>	* PP:p1 * VP:v1 * @ [v1:ADJUNCT S_PREP$===p1,sinv:^===v1] @ [v1:^CLAUSE_TYPE=DECLARATIVE] .
SINV:sinv	>	* VBD:v2 [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] NP:n1 * @ [v1:SUBJ===n1,v1:TNS_ASP===v2:TNS_ASP,sinv:^===v1] @ [v1:^CLAUSE_TYPE=DECLARATIVE] .
SINV:sinv	>	* VBD:v2 NP:n1 VP:v1 * @ [v1:SUBJ===n1,v1:TNS_ASP===v2:TNS_ASP,sinv:^===v1] @ [v1:^CLAUSE_TYPE=DECLARATIVE] .
SINV:sinv	>	S:s1 VP:v1 NP:n1 * @ [v1:COMP===s1,v1:SUBJ===n1,sinv:^===v1] @ [v1:^CLAUSE_TYPE=DECLARATIVE] .
SQ,SINV:sq	>	* [ADVP:a1|RB:a1] * @ [sq:^ ADJUNCT S_ADV$===a1] .
SQ:sq	>	* ADVP:a1 * VP:v1 * @ [v1:ADJUNCT S_ADV$===a1,sq:^===v1] @ [v1:^CLAUSE_TYPE=INTERROGATIVE] .
SQ:sq	>	* MD:m1 * [NP:n1|NP-A:n1] VP:v2 * @

		[v2:SUBJ===n1,sq:^===m1,sq:^===v2] @ [v2:^CLAUSE_TYPE=INTERROGATIVE,m1:^HelpVP TNS_ASP=!TNS_ASP] .
SQ:sq	>	* MD:m1 * VP:v1 * @ [sq:^===m1,sq:^===v1] @ [v1:^CLAUSE_TYPE=INTERROGATIVE] .
SQ:sq	>	* SBAR:s1 * VP:v1 * @ [v1:COMP$===s1,sq:^===v1] @ [v1:^CLAUSE_TYPE=INTERROGATIVE] .
SQ:sq	>	[VBP:v1|VBZ:v1|VBD:v1] * [NP:n1|NP-A:n1] * VP:v2 * @ [v2:SUBJ===n1,v2:HelpVP TNS_ASP===v1:TNS_ASP,sq:^===v2] @ [v2:^CLAUSE_TYPE=INTERROGATIVE,v2:^HelpVP TNS_ASP=!TNS_ASP,v1:^TNS_ASP=!TNS_ASP] .
SQ:sq	>	[VBZ:v1|VBD:v1] [NP:n1|NP-A:n1] ([NP:n2|NP-A:n2]) @ [v1:SUBJ===n1,v1:OBJ===n2,sq:^===v1] @ [v1:^CLAUSE_TYPE=INTERROGATIVE] .
SQ:sq	>	VB:v1 [NP:n1|NP-A:n1|NPB:n1] VP:v2 * @ [v2:SUBJ===n1,s1:^ _AUX_FORM_===v1:PRED,sq:^===v2] @ [v2:^CLAUSE_TYPE=INTERROGATIVE,v2:^TNS_ASP TENSE=PRES] .
SQ:sq	>	VBP:v1 [NP:n1|NP-A:n1] [NP:n2|NP-A:n2] @ [v1:SUBJ===n1,v1:OBJ===n2,sq:^===v1] @ [v1:^CLAUSE_TYPE=INTERROGATIVE] .
UCP:ucp	>	* ADJP:a1 CC:c1 NP:n1 * @ [n1:$===a1,n1:CONJ_FORM===c1,ucp:^===n1] .
VP,VP-A:vp	>	* [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] (PRT:a1) [NP:n1|NPB:n1|NP-A:n1] * @ [v1:OBJ===n1,vp:^===v1] .
VP,VP-A:vp	>	* [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] (PRT:a1) [NP-A:n1|NPB:n1|NP:n1] [NBP:n2|NP:n2|NP-A:n2] * @ [v1:OBJ===n1,v1:OBJ2===n2,vp:^===v1] .
VP,VP-A:vp	>	* [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] * [VP-A:v2|VP:v2] * (SBAR:s1) @ [v2:COMP$===s1,v2:TNS_ASP===v1:TNS_ASP,vp:^===v2] .
VP,VP-A:vp	>	* [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] * S-A:s1 * @ [v1:XCOMP===s1,vp:^===v1]

```
                @ [v1:^XCOMP SUBJ PRED = 'pro',v1:^XCOMP
                SUBJ PRONTYPE = NULL] .
VP,VP-A:vp  >    * [VBZ:v1|VBD:v1] [ADVP:a1|RB:a1]
                [NP:n1|NP-A:n1] @ [v1:ADJUNCT
                ADV$===a1,v1:PREDLINK===n1,vp:^===v1] @
                [v1:!PRED =c 'be_v'] .
VP,VP-A:vp  >    * VP:v1 * CC:c1 * VP:v2 * @
                [v2:$===v1,v2:CONJ_FORM===c1,vp:^===v2] .
VP,VP-A:vp  >    * VP:v1 * VP:v2 * @
                [v2:$===v1,v2:CONJ_FORM===c1,vp:^===v2] .
VP:vp       >    * [ADVP:a1|RB:a1] *
                [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1
                |VP:v1] * @ [v1:^ ADJUNCT
                ADV$===a1,vp:^===v1] .
VP:vp       >    * [ADVP:a1|RB:a1] * VP-A:v1 * @
                [v1:ADJUNCT ADV$===a1,vp:^===v1] .
VP:vp       >    *
                [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1
                ] ([ADVP:a1|PRT:a1]) NP-PRD:n1 * @
                [v1:PREDLINK===n1,vp:^===v1] .
VP:vp       >    *
                [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1
                ] (PRT:a1) NP-A:n1 NPB:n2 * @
                [v1:OBJ===n1,v1:OBJ2===n2,vp:^===v1] .
VP:vp       >    *
                [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1
                ] (PRT:a1) NPB:n1 * @
                [v1:OBJ===n1,vp:^===v1] .
VP:vp       >    *
                [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1
                ] * [S:s1|SBAR:s1] * @
                [v1:COMP$===s1,vp:^===v1] .
VP:vp       >    *
                [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1
                ] * [SBAR:s1|SBAR-A:s1|SBAR-A-g:s1] * @
                [v1:COMP$===s1,vp:^===v1] .
VP:vp       >    *
                [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1
                ] * ADVP:a1 * @ [v1:ADJUNCT
                ADV$===a1,vp:^===v1] .
VP:vp       >    *
                [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1
                ] * NP-TMP:n1 * @
                [v1:ADJUNCT$===n1,vp:^===v1] .
VP:vp       >    *
                [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1
```

		] * PP:p1 * (PP:p2 *) @ [v1:ADJUNCT PREP$===p1,v1:ADJUNCT PREP$===p2,vp:^===v1] .
VP:vp	>	* [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] ADJP:a1 * @ [v1:PREDLINK===a1,vp:^===v1] .
VP:vp	>	* [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] SG-A:s1 * @ [v1:XCOMP===s1,vp:^===v1] @ [v1:^XCOMP SUBJ PRED = 'pro',v1:^XCOMP SUBJ PRONTYPE = NULL].
VP:vp	>	* MD:m1 * [VP-A:v1|VP:v1] * @ [vp:^===m1,vp:^===v1] .
VP:vp	>	* RB:r1 * VP-A:v1 * @ [v1:ADJUNCT ADV$===r1,vp:^===v1] .
VP:vp	>	[VBZ:v1|VBD:v1] SBARQ:s1 @ [v1:COMP===s1,vp:^===v1] .
VP:vp	>	TO:t1 * VP-A:v1 * @ [vp:^===v1,v1:INF===t1:INF] .
VP:vp	>	VBP:v1 ? * ADJP:a1 @ [vp:^ ADJUNCT ADJ$===a1] .
VP:vp	>	VP:v1 * SBAR:s1 * @ [vp:^ COMP$===s1] .
VP-A,VP:vp-a	>	* [ADVP:a1|RB:a1] * @ [vp-a:^ ADJUNCT ADV$===a1] .
VP-A,VP:vp-a	>	* [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] (PRT:a1) NPB:n1 * @ [v1:OBJ===n1,vp-a:^===v1] .
VP-A,VP:vp-a	>	* [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] (PRT:a1) NPB:n1 * @ [v1:OBJ===n1,vp-a:^===v1] .
VP-A,VP:vp-a	>	* [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] (PRT:a1) NPB:n1 NP-A:n2 * @ [v1:OBJ===n2,v1:OBJ2===n1,vp-a:^===v1] .
VP-A,VP:vp-a	>	* [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] * [SBAR-A:s1|SBAR:s1] * @ [v1:COMP===s1,vp-a:^===v1] .
VP-A,VP:vp-a	>	* [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] * ADVP:a1 * @ [v1:ADJUNCT ADV$===a1,vp-a:^===v1] .
VP-A,VP:vp-	>	*

a	[VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] * PP:p1 * @ [v1:ADJUNCT PREP$===p1,vp-a:^===v1] .
VP-A,VP:vp-a	> * [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] * PRT:p1 * @ [v1:PART===p1,vp-a:^===v1] .
VP-A,VP:vp-a	> * [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] * SG:s1 * @ [v1:XCOMP===s1,vp-a:^===v1] @ [v1:^XCOMP SUBJ PRED = 'pro',v1:^XCOMP SUBJ PRONTYPE = NULL].
VP-A,VP:vp-a	> * [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] * SG:s1 SG:s2 * @ [vp-a:^XCOMP===s1,vp-a:^XADJUNCT===s2,vp-a:^===v1] @ [v1:^XCOMP SUBJ PRED = 'pro',v1:^XCOMP SUBJ PRONTYPE = NULL,v1:^XADJUNCT SUBJ PRED = 'pro',v1:^XDJUNCT SUBJ PRONTYPE = NULL] .
VP-A,VP:vp-a	> * [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] * SG-A:s1 * @ [v1:XCOMP===s1,vp-a:^===v1] @ [v1:^XCOMP SUBJ PRED = 'pro',v1:^XCOMP SUBJ PRONTYPE = NULL].
VP-A,VP:vp-a	> * [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] [VP-A:v2|VP:v2] * @ [v2:TNS_ASP===v1:TNS_ASP,vp-a:^===v2] .
VP-A,VP:vp-a	> * [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] PRT:p1 * @ [v1:PART===p1,vp-a:^===v1] .
VP-A,VP:vp-a	> * PP:p1 * PP:p2 * @ [vp-a:^ ADJUNCT PREP$===p2,vp-a:^ ADJUNCT PREP$===p1] .
VP-A:vp-a	> * [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] ADJP:a1 * @ [v1:PREDLINK===a1,vp-a:^===v1] .
VP-A:vp-a	> * [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] CC:c1 [VB:v2|VBD:v2|VBG:v2|VBN:v2|VBP:v2|VBZ:v2] * @ [vp-a:^$===v2,vp-a:^CONJ_FORM===c1,vp-a:^$===v1] .
VP-A:vp-a	> * ADVP:a1 * [VB:v1|VBD:v1|VBG:v1|VBN:v1|VBP:v1|VBZ:v1] * @ [v1:ADJUNCT ADV$===a1,vp-a:^===v1]

		.
VP-A:vp-a	>	* ADVP:a1 * SBAR:s1 * @ [vp-a:^ COMP$===s1,vp-a:^===a1] .
VP-A:vp-a	>	VB:v1 CC:c1 VB:v2 NP:n1 @ [v2:CONJ_FORM===c1,v2:$===v1,v2:OBJ===n1, vp-a:^===v2] .
WHADJP:whadjp	>	* WRB:w1 * JJ:j1 * @ [w1:ADJUNCT ADJ$===j1,whadjp:^===w1] .
WHNP:whnp	>	* WHADJP:w1 * JJR:j1 * @ [w1:ADJUNCT ADJ$===j1,whnp:^===w1] .
WHNP:whnp	>	* WHNP:w1 * PP:p1 * @ [w1:^ ADJUNCT PREP$===p1,whnp:^===w1] .
WHNP:whnp	>	* WPS:w1 * [NNS:n1|NN:n1] @ [n1:DET GENITIVE===w1,whnp:^===n1] .