Semi-Automatic Lexical Functional Grammar Development

Umer Khalid, Nayyara Karamat, Shahid Iqbal and Sarmad Hussain
Center for Research in Urdu Language Processing,
National University of Computer and Emerging Sciences, Lahore, Pakistan
{umer.khalid, nayyara.karamat, shahid.iqbal, sarmad.hussain}@nu.edu.pk

Abstract

We are presenting a semi-automatic Lexical Functional Grammar (LFG) development system. The context free grammar (CFG) is extracted automatically from the given parse tree and then the f-description is added to it using manually developed meta rules to form LFG. Regular expressions are used to define generic meta rules. This annotation system may be used potentially in any LFG parsing model for Machine Translation System.

1. Introduction

In this paper we are discussing a semi-automatic approach for developing Lexical Functional Grammar (LFG) [1, 6, 9] for English. The grammar is being used in an LFG based English to Urdu machine translation system.

In the traditional approach a manually developed grammar of the language is used in the parser. Grammar development for any language requires time and effort involving linguistic expertise [2]. F-description annotation adds more complexity to grammar development in case of lexical functional grammar. Furthermore, as the grammar rules increase, to improve the coverage of the language, computational effort for parsing increases affecting the efficiency of the system.

To overcome these issues faced with earlier system, a hybrid approach is developed by combining automatic grammar extraction and semi automatic annotation of CFG to build equivalent LFG. In this paper, we describe the annotation system used for semi automatic annotation of CFG to convert it to LFG.

Section 2 describes the related research. In Section 3 architecture of the system and syntax of the rules used in the system are described. Section 4 presents the experimental results and Section 5 discusses different issues faced during the system development and finally Section 6 concludes the paper.

2. Literature review

Lexical Functional Grammar (LFG) is a unification-based linguistic formalism which is suitable for computational purposes. LFG uses different structures for representing different levels of linguistic information in a sentence, for example, constituent structure (c-structure) and functional structure (f-structure) [10]. Context free grammar (CFG) is used to represent c-structure, whereas additional annotation called f-description is used to represent f-structure.

A treebank can be used to automatically extract the CFG grammar of a language [12]. This CFG can be annotated with f-description to obtain LFG. As reported by Sadler et al [14] it is shown in the previous work [1, 11, 13] that for configurational languages like English it is possible to define high level correspondence rules between c-structure and f-structure.

Van Genabith et al. [15, 16, 17] suggested a technique of creating F-structure annotated corpus by first extracting CFG rules from a tree bank, manually annotating these rule with f-descriptions and then reparsing the treebank corpus with the annotated grammar creating f-structures for the whole treebank. This idea was further enhanced by Sadler et al. [14], Frank [7] and Frank et al [8] by proposing automatic techniques for the annotation of CFG rules.

Sadler et al. [14] proposed a technique to automate the annotation process using the meta rules based on regular expressions. This is a semi-automatic technique in which manually developed meta rules are used to automatically annotate the CFG rules with f-descriptions. The rule format followed is stated below.

(1) Rule Format Proposed by Sadler et al [14]

Lhs > Rhs @ Anno

Lhs and Rhs are regular expressions which are to be matched with the grammar rule to apply the

@Anno f-description on it. These meta rules capture linguistic generalizations and multiple meta rules can be applied on a grammar rule to completely annotate it. 93.38 % precision and 91.58 % recall is reported for Sadler et al.'s [14] approach.

In a companion paper, Frank [7] discusses a different automatic annotation approach which is correspondence based in contrast with Sadler's rule based approach. Direct correspondence rules are applied to the tree structure instead of phrase structure rules which generate f-structure without doing reparsing which is required in Sadler's approach.

Cahill et al. [2, 3, 4] present an algorithmic approach for annotation. Cahill et al. [2] discusses two parsing architectures using this approach, *pipeline model* and *integrated model*. In pipeline model, Probabilistic Context Free Grammar (PCFG) is extracted from the treebank and is used to parse the sentence. The most probable parse tree is, then, annotated with f-descriptions to form LFG representation. These f-descriptions are then solved using a constraint solver to generate an f-structure. In this approach, only sentence specific CFG rules are annotated. In contrast, in integrated model, the whole treebank is first annotated with f-descriptions and an annotated probabilistic grammar A-PCFG is extracted from it. This A-PCFG can then be used to parse text and generate f-structures along with CFG trees (c-structure).

3. Annotation system

The annotation system uses regular expression technique proposed by Sadler et al. [14] for annotation. Pipeline model suggested by Cahill et al. [2] is followed in parsing. Probabilistic model of Collins [5] is used to generate most probable parse tree.

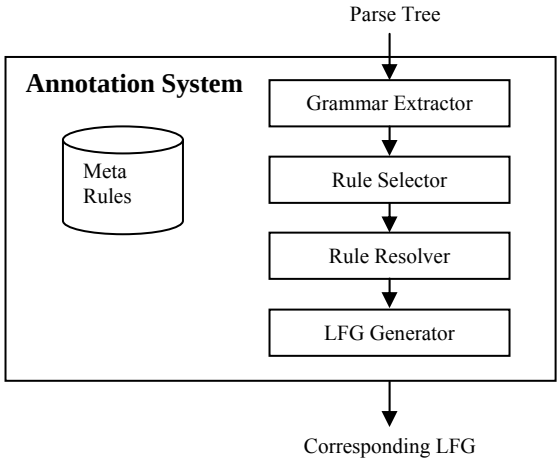

Figure 1: Architecture of annotation system

Architecture of our annotation system can be divided into four components, namely grammar extractor, rule selector, rule resolver and LFG generator. Figure 1 shows the components.

Grammar Extractor is used to extract CFG grammar from the tree generated by Collins [5] parser. Appropriate meta rules are selected for each grammar rule using Rule Selector module. One grammar rule may be annotated using a combination of appropriate rules. This combination is determined by Rule Resolver. Once meta rules for all the grammar rules are decided, LFG Generator annotates the grammar rules with f-descriptions according to meta rules and outputs LFG against the tree. A sample tree and its corresponding generated grammar is given in Appendix A. Section 3.1 describes syntax for meta rules used in the system. Then Section 3.2, Section 3.3, Section 3.4 and Section 3.5 describe the components of the system.

3.1. Meta rules

Structure followed for meta rules is largely same as in Sadler et al [14]. One difference in the main structure is that annotation (Anno) part is subdivided into two parts. The new structure is given below.

(2) Rule Format of our system

Lhs → Rhs @ Anno1 (@ Anno2)

@Anno1 is same as that of Sadler's technique but the @Anno2 is added to define f-descriptions which does not involve variable at their right hand side. This part is optional.

Regular expression operators Kleene star * (without argument), optional (…), and disjunction | are allowed at Rhs of the meta rules.

Symbols in Lhs and Rhs consist of two parts separated by ':', e.g. NP:n1. First part is the CFG symbol which should be present in the CFG rule whereas second part denotes a variable to which an f-structure will be associated.

Syntax of annotation rules is given in Appendix B. Following are some sample meta rules.

(3) VP:vp > * MD:m1 * [VP-A:v1|VP:v1] *
@[vp:^===m1,vp:^===v1].

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

(5) NPB:npb > * (JJ:j1) NN:n1 *

@ [npb:^===n1, n1:ADJUNCT ADJ===j1].

(6) NPB:npb > * JJ:j1 [NN:n1 | NNS:n1] *
@ [npb:^===n1, n1:ADJUNCT ADJ===j1].

(7) PP:pp > ? S-A:s1 @ [pp:^COMP===s1].

Meta rules are developed by manually observing the parse trees of Penn Treebank II. First 100 parse trees were analyzed to develop 225 meta rules.

One meta rule can be applied multiple times on a grammar rule. Multiple rules may be applied in combination to annotate a single grammar rule. There is a possibility of multiple valid meta rules or rule combinations which can be applied on a grammar rule. In such cases, Rule Resolver module explained in Section 3.4 chooses one of the valid combinations. Currently, it arbitrarily picks one of the valid options.

It was observed that for some Treebank phrases, there can be similar meta rules. For example following rule is equally applicable to NP, NP-A and NPB.

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

In such cases, all the possible Lhs symbols of grammar rules are written at Lhs of meta rule in the form of comma separated list. This way, rule writer need not duplicate these rules for each phrase. During processing, these rules are expanded to individual rules.

3.2. Grammar extractor

The Grammar Extraction module takes the normalized output of the of Collins parser [5] and extracts the context free grammar rules from the parse tree. Following is the example of a parse tree and the extracted grammar for the sentence “he ate apples with me.

(9) Output of Collin’s Parser

```
(S~ate~2~2
        (NP-A~he~1~1
            (NPB~he~1~1 he/PRP ) )
        (VP~ate~3~1
            ate/VBD
            (NP-A~apples~1~1
                (NPB~apples~1~1
                    apples/NNS ) )
            (PP~with~2~1
                with/IN
                (NP-A~me~1~1
                    (NPB~me~1~1
                        me/PRP ) ) ) ) ) )
```

(10) Derived CFG Rules

1) S → NP-A VP
2) NP-A → NPB
3) NPB → prp
4) VP → vbd NP-A PP
5) NP-A → NPB
6) NPB → nns
7) PP → in NP-A
8) NP-A → NPB
9) NPB → prp

In the above grammar, there are multiple rules for same non-terminal. This causes extra processing during tree analysis for f-structure generation due to possible ambiguity. This problem is solved by appending a unique number to each non-terminal as shown in (11).

(11) Unambiguous CFG Rules

1) S0 → NP-A-1 VP-2
2) NP-A-1 → NPB-3
3) NPB-3 → prp
4) VP-2 → vbd NP-A-4 PP-5
5) NP-A-4 → NPB-6
6) NPB-6 → nns
7) PP-5 → in NP-A-7
8) NP-A-7 → NPB-8
9) NPB-8 → prp

3.3. Rule selector

Rule selector chooses applicable meta rules for each grammar rule extracted from the tree on the basis of valid regular expression matching of Rhs. The generic process for rule selection is as follows.

For each CFG rule

For each meta rule whose LHS = CFG rule’s LHS

If (meta rule’s RHS matches CFG rule’s RHS)

Add meta rule in the list attached with CFG rule

While attaching the meta rule with the CFG rule, it aligns the f-descriptions provided by meta rule with their respective non-terminals of CFG rule.

The worst case time for annotating ‘g’ grammar rules with ‘n’ meta rules is O (g*n*w) where ‘w’ is maximum number of symbols in the CFG rules which is about four symbols on average.

3.4. Rule resolver

Once meta rules are selected for each CFG rule, it is quite possible that more than one meta rules are selected for a single CFG rule. Therefore, we need to find the most suitable set of f-descriptions to annotate the corresponding CFG rule. The idea is to merge the non-clashing f-descriptions to completely annotate the grammar rule. The clash is determined on the basis that only one meta rule can add f-description to particular non-terminal of Rhs. When two or more meta rules are being combined, they are contributing to different non-terminals of the rule. If two meta rules contain the f-description for the same non-terminal, it should be the same to merge these two rules.

The following example describes the above mentioned solution.

Assume an abstract grammar rule is:

S →	WP	XP	YP	ZP

Suppose that there are four meta rules a, b, c and d which match with CFG rule given above. The capital letters signify the f-descriptions which the respective rule attaches with the particular non-terminal.

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

Meta rule ‘a’ annotates first two symbols of the CFG rule (WP, XP). Rule ‘b’ annotates second, third and fourth symbol. Meta rule ‘c’ covers first, second and fourth symbol. Lastly, meta rule ‘d’ covers first, third and fourth symbol of the CFG rule.

Rule ‘a’ can only be merged with rule ‘d’ as it clashes with rules ‘b’ and ‘c’ second non-terminal. Rule ‘a’ and ‘d’ combined can annotate the rule as follows.

S →	WP	XP	YP	ZP
1)	A	B	D	E

Another possible combination is rule ‘b’ and ‘c’ with the following result.

S →	WP	XP	YP	ZP
2)	A	C	D	E

A group of meta rules which provides maximum coverage (annotates maximum terminals / non-terminals) is selected. In case there exist more than one meta rules providing maximum coverage (as in example above) any of the meta rule combination is selected arbitrarily to annotate the CFG rule at this time.

3.5. LFG generator

LFG generator outputs the formal LFG syntax based on the rules resolved by rule resolver that can be parsed using an LFG parser to generate f-structure.

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 unannotated in the CFG rules. Such symbols are annotated as ADJUNCT taking the assumption that there is more chance for an unknown new element to be an ADJUNCT. Similar approach is followed as by Cahill et al [4].

4. Experimental results

In this section, we discuss the results for automatically applying the meta rules on CFG.

4.1 Test data

Total 103 test sentences were selected from the news websites (BBC, CNN and THENEWS) with average length of 20 words. Total of 363 CFG rules were used in the analysis of these sentences according to Collin’s [5] parser analysis. Among these rules 333 were non singletons (having multiple symbols at Rhs) and 30 were singletons (having single symbol at Rhs)

4.2 Meta rules and annotation evaluation

Only non-singleton rules are taken into consideration during evaluation because singletons are always annotated as root (↑ = ↓) and there is no chance of incorrect annotation in them. The grammar rules are manually inspected for correctness and it was observed that among 333 unique non singletons 21 were annotated incorrect which gives correct rule percentage of 93.69%.

Correct annotation	312
Incorrect annotation	21
% Correct	93.69%

It is noted that correctly annotated rules are far more frequent than incorrectly annotated rules. All the incorrect rules only occurred once in test sentences whereas frequency of correct rules was about five on the average. Total of 1485 instances of non singleton rules were annotated in which only 21 were found

erroneous. This way the system makes correct annotation 98.58% of the times.

Correct rule instances	1485
Incorrect rule instances	21
% Correct	98.58%

This paper has reported precision as the total number of correct annotations given the total annotations our system made. Sadler has reported 93.38% precision of the system. The precision this paper reports is 93.69, which appears almost equal to that of Sadler.

5. Discussion

Pipeline model proposed by Cahill et al. [2] is used by the system because it took less development time and effort. An already existing PCFG model developed by Collins [5] was used for the first phase of CFG parsing and then annotation was done on the most probable tree generated by the model. The meta rules developed for the system are equally applicable to the integrated model of Cahill et al. [2] so the system can be modified to use an integrated model in future.

Reason for 7 wrong annotations was incorrect parse tree generated by the parser. Such incorrect grammar rules were not handled in the meta rules. Such rules can be added to the system if they do not clash with already existing correct rules. For example, the phrase "Armed with knowledge" is expected to be parsed as in (12).

(12) Correct parse of "Armed with knowledge"

```
(NPB Armed/VBN
   (PP-A  with/IN
      (NP-A
         (NPB knowledge/NN) ) ) )
```

But the parser parsed it the following way creating wrong CFG rule which was not handled in meta rules.

(13) Incorrect parse of "Armed with knowledge"

```
(PP Armed/VBN
   (PP-A  with/IN
      (NP-A
         (NPB knowledge/NN) ) ) )
```

There were 12 meta rules which were found missing during testing that caused wrong annotation. Such rules will be added to the system in the next enhancement for increasing the accuracy of the system. There is a chance of occurrence of more such rules as the data observed to develop meta rules is very limited.

One problem encountered is incorrect identification of SUBJ/OBJ in case of wh sentences.

(14) "what is your name"

```
(TOP
  (SBARQ
    (WHNP  what/WP )
    (SQ
       (VP  is/VBZ
          (NP-A
             (NPB  your/PRPS name/NN ) ) ) ) ) )
```

(15) "what is he doing"

```
(TOP
  (SBARQ
    (WHNP  what/WP )
    (SQ
       is/VBZ
       (NP
            (NPB  he/PRP ) )
       (VP  doing/VBG ) ) ) )
```

In sentence (14) "what" is the subject of the sentence whereas in (15) "what" is acting as object of the sentence. But we can not detect it correctly because of the use of the same CFG rule "SBARQ → WHNP SQ" in both sentences.

Currently, in case of multiple applicable meta rules, one set of rules with maximum coverage is selected arbitrarily. Such a selection may result in wrong analysis in some cases. Some probabilistic model can be added to the system for more accurate selection of rules. Only one occurrence of such wrong selection is observed during testing among 21 wrong annotations.

The system is using a subset of regular expression operators. Kleene star '*' with argument, positive Kleene '+', optionality '?' and complement '~' are the operators which are not used in rule writing.

Comparing with Cahill [2], long distance dependencies of traces are not intentionally used. Traces and wh-movements are traced in the lower hierarchies of the tree which is not yet handled in this paper. The paper follows most of the Sadler's work which is not exactly meant to cover the traces of movements in depth of tree. However, Frank [7] has somehow presented the idea to resolve these depth dependencies.

6. Conclusion

In this paper, we have discussed an automatic LFG development methodology. A set of meta rules is developed by manually inspecting the trees of Penn Treebank. These meta rules are applied automatically on CFG rules to annotate with f-description. Results are very encouraging which proves that a small set of meta rules can be developed to annotate a complex CFG with f-structure information. The system is presently being further matured by further testing and addition of meta-rules and is being integrated within the existing MT system.

7. References

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

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

[3] A. Cahill, McCarthy M., van Genabith J. and Way A., "Automatic Annotation of the Penn-Treebank with LFG F-Structure Information", in Proceedings of the *LREC Workshop on Linguistic Knowledge Acquisition and Representation*: Bootstrapping Annotated Language Data, pages 8-15, Las Palmas, Canary Islands, Spain., 2002b.

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

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

[6] M. Dalrymple , *Lexical Functional Grammar*. San Diego, Calif. London, Academic Press, 2001.

[7] A. Frank , "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, 2000.

[8] A. Frank., Sadler L., van Genabith J. and Way A. , "From Treebank Resources to LFG F-Structures" in (ed.)Anne Abeille, *Treebanks: Building and Using Syntactically Annotated Corpora*, pp:367-389, Kluwer Academic Publishers, The Netherlands, 2002.

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

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

[11] T. H. King, , *Configuring Topic and Focus in Russian*. Stanford: CSLI Publications, 1995.

[12] A. Kinyon, Prolo Carlos A. , "A classification of grammar development strategies", in proceedings of *COLING-02 on Grammar engineering and evaluation*, p.1-7, , 2002.

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

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

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

[16] J. van Genabith, , Sadler L., and Way A. , "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, , 1999b

[17] J. van Genabith, , Way A., and Sadler L. , "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, 1999c.

Appendix A: Sample Tree and Corresponding LFG Generated by the System

"He said the tests confirmed Tehran had missiles with a limited range of up to 2,000km"

Input Parse Tree:

```
(TOP~said~1~1
  (S~said~2~2
    (NP-A~He~1~1
      (NPB~He~1~1 He/PRP ) )
    (VP~said~2~1 said/VBD
      (SBAR-A~confirmed~1~1
        (S-A~confirmed~2~2
```

```
(NP-A~tests~1~1
  (NPB~tests~2~2 the/DT tests/NNS ) )
(VP~confirmed~2~1 confirmed/VBD
  (SBAR-A~had~1~1
    (S-A~had~2~2
      (NP-A~Tehran~1~1
        (NPB~Tehran~1~1 Tehran/NNP ) )
      (VP~had~3~1 had/VBD
        (NP-A~missiles~1~1
          (NPB~missiles~1~1 missiles/NNS ) )
        (PP~with~2~1 with/IN
          (NP-A~range~2~1
            (NPB~range~3~3 a/DT limited/VBN
range/NN )
            (PP~of~2~1 of/IN
              (PP-A~up~2~1 up/IN
                (PP-A~to~2~1 to/TO
                  (NP-A~000km~2~1
                    (NPB~000km~2~2 2/CD
000km/CD ) ) ) ) ) ) ) ) ) ) ) )
```

LFG generated:

```
NPB_556 -> cd:!$^ ; cd:!$^ ; .
NP_A_541 -> NPB_556:^ = ! ; PRN_589:!$^
ADJUNCT PRN ; .
PP_A_517 -> to:^ = ! ; NP_A_541:^ OBJ = ! ; .
PP_A_496 -> in:^ = !; PP_A_517:^ OBJ = ! ; .
PP_475 -> in:^ = ! ; PP_A_496:^ ADJUNCT PREP =
! ; .
NPB_430 -> dt:!$^ SPEC DET ; vbn:!$^ ADJUNCT
PARTICIPLE ; nn:^ = ! ; .
NP_A_415 -> NPB_430:^ = ! ; PP_475:!$^
ADJUNCT PREP ; .
PP_389 -> in:^ = ! ; NP_A_415:^ OBJ = ! ; .
NPB_357 -> nns:^ = ! ; .
NP_A_339 -> NPB_357:^ = ! ; .
VP_310 -> vbd:^ = ! ; NP_A_339:^ OBJ = ! ;
PP_389:!$^ ADJUNCT PREP ; .
NPB_281 -> nnp:^ = ! ; .
NP_A_265 -> NPB_281:^ = ! ; .
S_A_248 -> NP_A_265:^ SUBJ = ! ; VP_310:^ = ! ,
^CLAUSE_TYPE=DECLARATIVE ; .
SBAR_A_235 -> S_A_248:^ = ! ; .
VP_203 -> vbd:^ = ! ; SBAR_A_235:!$^ COMP ; .
NPB_160 -> dt:!$^ SPEC DET ; nns:^ = ! ; .
NP_A_145 -> NPB_160:^ = ! ; .
S_A_129 -> NP_A_145:^ SUBJ = ! ; VP_203:^ = ! ,
^CLAUSE_TYPE=DECLARATIVE ; .
SBAR_A_110 -> S_A_129:^ = ! ; .
VP_77 -> vbd:^ = ! ; SBAR_A_110:!$^ COMP ; .
NPB_51 -> prp:^ = ! ; .
NP_A_39 -> NPB_51:^ = ! ; .
S_26 -> NP_A_39:^ SUBJ = ! ; VP_77:^ = ! ,
^CLAUSE_TYPE=DECLARATIVE ; .
S' -> S_26:^ = ! ; .
```

Appendix B

Syntax of Annotation Rules

Lhs → Rhs ‘@’ Anno1 (‘@’ Anno2)

Lhs -> Symbols ’:’ f_var
Symbols -> CFGSymbol | Symbols ’,’ CFGSymbol

Rhs -> ’*’ | ’(’ Rhs ’)’ | RhsStatement

Rhs -> Rhs Rhs

Rhs -> ’[’ RhsStatement ’|’ RhsStatementsRec ’]’

RhsStatementsRec -> RhsStatement '|' RhsStatementsRec

RhsStatement -> CFGSymbol ’:’ f_var

Anno1 -> Anno1 ‘,’ LFGAnnotation
LFGAnnotation -> f_var ‘:’ LFGRelation ‘===’ f_var

Anno2 -> f_var ‘:’ LFGAttribute ‘===’ Value
Anno2 -> Anno2 ‘,’ f_var ‘:’ LFGAttribute ‘===’ Value