Linear Precedence Constraints in "Lean Formalisms"
Part II

Hans Uszkoreit
Gregor Erbach
Wojciech Skut
Universität des Saarlandes, Computational Linguistics, and
Deutsches Forschungszentrum für Künstliche Intelligenz
D-66041 Saarbrücken, Germany
uszkoreit@coli.uni-sb.de

February 28, 1994

1. Introduction

In this report, we demonstrate the practical application of the methods outlined in Part I by the implementation of a grammar that covers the ordering of adjuncts and complements in the German Mittelfeld.

Originally, we had envisaged to allow the statement of sets of LP rules and their automatic compilation into the rules of an ALEP grammar in the form of a level 2 extension. The methods for such a compilation step have already been described in the paper by Engelkamp et al. (1992). An automatic compilation would work well for cases where all LP rules apply simultaneously, as in GPSG, but not for the complex kinds of LP rules needed for languages like German.

The additional complexity needed to handle German word order phenomena comes from the need to account for

• conflicting LP rules, and

• interactions between LP rules.

Conflicts arise in cases where only some of the LP rules can be satisfied. For example, if we have the LP-rules

Definite < Indefinite

OBJ2 < OBJ

and the indirect object (OBJ2) is indefinite, and the direct object (OBJ) is definite, there is no linearisation which does not violate one of the rules. This is a problem for parsing, but especially for generation, as there would be no way to generate such a sentence.

The need for weighted LP constraints to deal with these situations has been noted in the linguistic literature (e.g. Uszkoreit 1987), but has not found its way into computational implementations of grammars.

In the grammar described in this report, conflicts are handled by using several rules, each of which enforces only a subset of the ordering constraints. The subsets are chosen in such a way that the "weaker" rules are only applied if the stronger rules do not apply.

Interactions between LP constraints arise with the following set of LP rules:

Pronoun < Non-Pronoun

OBJ2 < OBJ

In fact, these rules are too simple, as the ordering between OBJ2 and OBJ is reversed in case both are pronominal. Such interactions are explicitly covered in the grammar we provide.

The grammar we provide has been developed with both linguistic adequacy and efficient processing in mind. We are not aware of any other implemented grammar for German that provides an equally detailed coverage of word order regularities. Underspecification is used as much as possible to avoid the creation of a large search space. The multiplication of rules does also improve the efficiency of the grammar as the specialized rules can help to rule out failing branches of the search space as early as possible.

In the following sections, we describe the coverage and implementation of the grammar, and discuss the runtime efficiency of the grammar.

2. Preliminaries

In order to provide an adequate description for linear precedence constraints, two facts should be taken into consideration. Firstly, there are several factors that may influence the ordering of constituents, e.g.

- syntactic category
- grammatical relations/functions (SUBJ < OBJ2 < OBJ < POBJ)
- case (nominative < dative < accusative)
- thematic role (source < goal)
- pronominal vs. non-pronominal forms (pronoun < non-pronoun)
- discourse function (topic < focus)
- definitness (definite NP < indefinite NP)

Secondly, since all these factors work simultaneously and cumulatively, their interaction should be described, as well. It has been observed that some ordering criteria carry more weight then others, as the following examples show:

(1) der Mann zeigte ihn der Frau
the[nom] man showed him[acc] the[dative] woman
the man showed him to the woman

vs.

(2) *der Mann zeigte der Frau ihn

Here the pronominal accusative NP precedes the dative NP though the general rule for the ordering of objects states that the accusative (direct) object should follow the dative (indirect) object. Therefore, the particular word order factors must be assigned different priorities, which would allow to overwrite the weaker constraint obj2 < obj, violated in (1).

3. Encoding of Relevant Information in Feature Structures

Many of the abovementioned criteria overlap as nominal phrases in topic are usually definite, dative NPs perform the grammatical function of the indirect object, syntactic subjects represent semantic agents etc. Thus, we take only four factors into account which we regard as very important and relatively independent of each other, namely grammatical function, syntactic category, pronominality and discourse structure. In addition, a morphonological criterion can be useful to express the constraint that the enclitic accusative pronoun 'es' cannot be stressed, which is also of importance for the constituent ordering, because only focussed and stressed accusative pronouns can follow pronominal dative NPs.

In order to encode it in feature structures, we define the lp_relevant_type[1] :

```
type(lp_relevant_type:
        focus => boolean([{plus,minus,nil}]),
        pro => boolean([{plus,minus,nil}]),
        clitic => boolean([{plus,minus,nil}]) ).
```

One of the criteria to be taken into account, namely the grammatical relations, need not be introduced explicitly on this level because our grammar is based on the notion of grammatical functions, i.e. all arguments are treated separately depending on their syntactic function, such as subject, object, object2 etc. Each complement is then associated with a structure of the lp_relevant_type:

```
type(comp_type:
        val => type(sign_or_nil_type: { }),
        bound_off => boolean([{plus,minus}]),
```

[1] All attributes of the lp_relevant_type are three-valued. The third value, nil, is used to express the absence of the corresponding constituent. This facilitates an efficient encoding of ordering rules (see also below).

lp_relevant => type(lp_relevant_type: {})).

where the value of **val** must unify with the corresponding complement itself. The feature **bound_off** indicates whether the complement has already been found.

In general, we distinguish four types of complements: subject, direct object (obj), indirect object (obj2) and prepositional object (pobj). The data structure we employ for valence is then a conjunction of four attributes:

```
type(subcat_type:
        subj => type(comp_type: { } ),
        obj => type(comp_type: { } ),
        obj2 => type(comp_type: { } ),
        pobj => type(comp_type: { } ) ).
```

4. Integration of LP Constraints into Grammar Rules

Another piece of the word order relevant information missing here, namely the syntactic category, is contained in the grammar rules in form of categorial annotations. Thus, the grammar consists of rules partially instantiated with categorial symbols. The grammatical function performed by the non-head daughter is also specified.

The mechanism we employ for handling linear precedence constraints resembles the method proposed in Engelkamp et al. (1992). Binary branching syntactic structures are licensed by rules that perform an additional function: they not only describe local trees but also incrementally collect the word order relevant information and check the possibility of the current constituent ordering within a head domain.

Therefore, whenever a head combines with one of its complements the **lp_relevant** attribute that is associated with the complement becomes instantiated (as long as the complement has not combined with the head the attributes focus, pro, clitic have the value nil). For the sake of simplicity, we treat the information about the word order relevant properties of the argument as a head feature, which we label as **lp_head.** The information about the other complements is inherited without change:

```
sign_type:
        {syn => syn_type:
                {head => HEAD,
                 subcat => subcat_type:
                                {subj => comp_type:
                                        {val => SUBJ_VAL,
                                         bound_off => yes,
                                         lp_relevant => LPR_COMP },
                                 obj => OBJ,
```

```
                                obj2 => OBJ2,
                                pobj => POBJ }
                        <
[@SUBJ_VAL sign_type:
        {syn => syn_type:
                {head => nom_head_type:
                        {lp_head => @LPR_COMP lp_relevant_type: { }}}},
sign_type:
        {syn => syn_type:
                {head => @HEAD verbal_head_type: { },
                 subcat => subcat_type:
                        {subj => comp_type:
                                {val => SUBJ_VAL,
                                 bound_off => no,
                                 lp_relevant => lp_relevant_type:
                                                {focus => nil,
                                                 pro => nil,
                                                 clitic => nil },
                        obj => OBJ,
                        obj2 => OBJ2,
                        pobj => POBJ } ].
```

Here a phrase final verbal head combines with its subject (index variable **SUBJ_VAL**). The value of the path **syn:subcat:subj:bound_off** on the head daughter is **no**, which means that the subject has not been bound off yet. The same path on the mother node becomes instantiated with **yes** in order to ensure the coherence of the maximal verbal projection. The information about the word order relevant properties of the subject, contained in the value of the path **syn:head:lp_head** on the complement daughter, is percolated to the mother and then to the top node of the whole head domain (index variable **LPR_COMP**). The corresponding features of the other complements (the variables **OBJ, OBJ2, POBJ**) are inherited without change so that for each complement ***COMP*** all head nodes below the one where ***COMP*** is bound off satisfy the equations

syn:subcat:*COMP* :lp_relevant:focus => nil,
syn:subcat:*COMP* :lp_relevant:pro => nil,
syn:subcat:*COMP* :lp_relevant:clitic => nil,

and all head nodes above it bear the word order relevant information introduced by ***COMP***.

The second aspect of the proposed mechanism, i.e. the imposing of linear precedence constraints, can now be easily integrated into the grammar because each local tree contains the information about the word order relevant properties of both the complement and the part of the head domain dominated by the head daughter. It suffices to instantiate the **lp_relevant** features of the **subj**, **obj**, **obj2** and **pobj** attributes on the head daughter as well

as the value of the **lp_head** attribute on the complement[2] . A disallowed combination will then result in unification failure. For instance, in order to rule out the ungrammatical ordering

* der Frau[obj2, non-pro] < ihn[obj, pro]

we instantiate the **lp_relevant** and **lp_head** attributes in the (head-final) head-object2 rule as follows:

```
sign_type: {  }
                <
[ @OBJ2 sign_type:
        {syn => syn_type:
                {head => nom_head_type:
                        {lp_head => lp_relevant_type:
                                   {focus => _,
                                    pro => minus,
                                    clitic => _    }}}},
sign_type:
        {syn => syn_type:
                {head => verbal_head_type: { },
                 subcat => subcat_type:
                        {obj => comp_type:
                              {lp_relevant => lp_relevant_type:
                                             {focus => _,
                                              pro => ~plus,
                                              clitic => _ }},
                         obj2 => comp_type:
                              {val => @OBJ2,
                               lp_relevant => lp_relevant_type:
                                             {focus => nil,
                                              pro => nil,
                                              clitic => nil}}}}} ].
```

The path equation **syn:subcat:obj:lp_relevant:pro => ~plus** is satisfied, only if the value of this path is **nil** (no direct object is present within the head phrase) or **minus** (the direct object is non-pronominal). This formulation therefore prevents a non-pronominal indirect object from combining with a (verb-final) verbal projection containing a pronominal direct object.

2. Unlike in Engelkamp et al. (1992), the grammatical representation of linguistic signs does not bear any information about right and left context restrictions. Linear precedence constraints are directly expressed in the rules so that linguistic objects serves only as a source of word order relevant data such as pronominality, discourse function, morphophonological properties, etc.

On the other hand, there are orderings acceptable only in some (pragmatic) contexts. For instance, the "neutral" ordering

NP(dat, non-pro) < NP(acc, non-pro)

may be reversed when the dative object is focussed and the accusative NP remains in the topic, cf.

(3) daß der Mann das Buch dem Kind gibt
that the[nom] man the[acc] book the[dat] child gives
that the man gives the book to the child

To license this structure, we introduce the following instance of the right branching head-object rule:

```
sign_type: {  }
                <
[ @OBJ sign_type:
        {syn => syn_type:
                {head => nom_head_type:
                        {lp_head => lp_relevant_type:
                                        {focus => minus,
                                         pro => minus,
                                         clitic => minus     }}}},
sign_type:
        {syn => syn_type:
                {head => verbal_head_type: { },
                 subcat => subcat_type:
                                {obj => comp_type:
                                        {val => @OBJ,
                                         lp_relevant => lp_relevant_type:
                                                        {focus => nil,
                                                         pro => nil,
                                                         clitic => nil}},
                                 obj2 => comp_type:
                                        {lp_relevant => lp_relevant_type:
                                                       {focus => plus,
                                                        pro => no,
                                                        clitic => minus}}}}}} ].
```

With this rule, a non-pronominal dative NP will be marked as **focus +** when it is preceded by a direct non-prominal object, whose grammatical representation in turn becomes instantiated with **focus -**.

5. Descriptive Power

A consequence of this approach is the fact that each rule must be split into several cases depending on the syntactic category of the daughter nodes and on those of their properties which are important for the ordering of constituents. Such an instance of a general syntactic structure describes the interaction of some lp-relevant factors in a particular case so that assigning priorities to linear precedence generalizations is not difficult, anymore: when a grammar rule is instantiated with a stronger constraint the weaker ones can be omitted. Moreover, our approach works even if a word order criterion turns into its inversion under the influence of another factor. In German, the constraints for grammatical function (or case) and pronominality behave in this manner, cf.

(4) daß er der Frau den Mann zeigt - dat < acc
that he[nom] the[dat] woman the[acc] man shows
that he shows the man to the woman

but

(5) daß er ihn ihr zeigt - acc < dat
that he[nom] him[acc] her[dat] shows
that he shows him to her

Structures like (5) can be described by the following rule instance:

```
sign_type: {  }
              <
[ @OBJ sign_type:
      {syn => syn_type:
            {head => nom_head_type:
                  {lp_head => lp_relevant_type:
                          {focus => _,
                           pro => plus,
                           clitic => _    }}}},
sign_type:
      {syn => syn_type:
            {head => verbal_head_type: { },
             subcat => subcat_type:
                     {obj => comp_type:
                          {val => @OBJ,
                           lp_relevant => lp_relevant_type:
                                    {focus => nil,
                                     pro => nil,
                                     clitic => nil}},
                      obj2 => comp_type:
                          {lp_relevant => lp_relevant_type:
                                    {focus => _,
                                     pro => plus,
                                     clitic => _}}}}} ].
```

In addition, we analyse the German verb initial and verb-final sentences as resp. left and right branching structures[3] . This make us distinguish between two types of grammar rules for verbal projections since a head final rule contains only the information about the constituents following the complement daughter and since there is no such information in a head initial structure. Thus, a left branching rule should check whether the complement may be preceded and a right branching one if it may be followed by the elements of the head domain.

Furthermore, the particular instances of general syntactic rules should be disjoint. Otherwise, the parser would provide multiple analyses for unambiguous structures or instantiate attributes that can remain underspecified.

So, the three general types of the head-complement structure, i.e. head-subject, -object and -object2 rules, has been splitted into 18 cases: 8 for verb final and 10 for verb initial phrases. This rule corpus covers a majority of the possible complement orderings in the "Mittelfeld". In particular, our grammar allows among other things

- for the "neutral" ordering subj < obj2 < obj,
- for the reversed ordering of pronominal objects,
- for a focussed subject to follow non-focussed objects,
- for pronominal objects to precede the subject,
- for a pronominal direct object to precede the non-pronominal indirect object,
- for a focussed indirect object to follow the non-focussed direct object.

Unacceptable orderings like

- a non-pronominal indirect object in topic following the non-pronominal focussed direct object,
- non-pronominal objects preceding a pronominal subject, or
- the enclitic accusative pronoun "es" following a dative pronoun

are ruled out.

What should be done in addition in order to complete this approach, is a refinement of our simplified treatment of topic constituents, as their ordering seems sensitive to some discourse relations that are much more complex than the simple topic/focus dichotomy. However, such an extension would presumably require the introduction of an enlarged representation level for semantics and pragmatics. In fact, no satisfactory formalization of discourse semantics has been proposed yet, so that we are forced to take account of only one pragmatic factor.

6. Efficiency

---

[3]. The verb-second sentences are analysed as verb initial clauses with a topicalized "Vorfeld"-constituent.

The relatively large number of rules thus created does not necessarily lead to a proportional deterioration in the computational complexity of the syntactic analysis. In the bottom-up processing, each grammar rule is applied to syntactic structures that have already been instantiated with lexical information. Hence, most of inadequate rule applications would immediately result in a unification failure without causing much harm to the efficiency of parsing. Instantiated rule schemata prove disadvantageous only in one case, namely when ambiguous structures are to be dealt with. Among the five factors that we regard as most relevant for the constituent ordering, solely two may happen to be insufficiently instantiated with lexical information, namely grammatical function/case and topic/focus[4]. However, the latter can be handled quite efficiently by our grammar since there is no need for immediate desambiguation - except in the case of some marked orderings, cf. (3) - and the feature **focus** can generally remain underspecified.

On the other hand, the complements that are ambiguous with regard to the grammatic function they perform cause much more problems for syntactic processing. Since ALEP does not support set operations, only lists or finite conjunctions of attributes can be used to describe valence frames of linguistic objects. An underspecified description of these data structures - e.g. regular path expressions - is not possible, either. Therefore, the formalism provides no means for expressing generalizations of grammar rules, as the rules usually access the subcategorization structure. Now, when a complement is ambiguous with regard to its grammatical function - for instance the German NP "die Frau", which can be either nominative (subject) or accusative (direct object) - the ambiguity must be resolved as soon as the complement and the head are matched with a grammar rule. The result of such a desambiguation is then a kind of disjunctive normal form[5], and all disjuncts must be processed separately, which is usually very inefficient.
Hence, dealing with case/grammatical function ambiguities is a general problem for parsing and not only one for expressing linear precedence constraints. Moreover, as the other word order factors do not cause such difficulties, the grammar augmented with context restrictions has proved to be faster then the unrestricted version allowing all permutations of nominal complements in the "Mittelfeld": the parse time wasted for search in a larger corpus of rules has become compensated by a better filtering of ungrammatical substructures. The exact results of the comparison are listed below:

Input	Parse Time with the Restricted Grammar	Parse Time with the Unrestricted Grammar
gibt er es ihr	0.733 sec.	1.234 sec.
daß es der Mann sieht	4.100 sec.	6.917 sec.
daß er es ihr gibt	1.917 sec.	3.583 sec.
gibt ihm der Mann das Buch	6.750 sec.	38.433 sec.
*gibt er es ihr (ill-formed)	0.734 sec.	1.217 sec.

[4] . Of course, words ambiguous with regard to the other factors can be found, too, but such ambiguities are less frequent then the two ones mentioned above.
[5] . ALEP does not allow for another treatment of disjunctions.

As the lp-unrestricted grammar covers only a subset of all the phenomena that the restricted one does (e.g. topicalization or adjunction are not dealt with), the real improvement in efficiency is larger than the direct comparison suggests.

7. Ordering of Adjuncts

A similar mechanism can be employed for treating the ordering of adjuncts. Until now, we have implemented rules applying linear precedence constraints to source and goal modifiers. The corresponding representation level is of the type **adjuncts_type**:

```
type(adjuncts_type:

        source => type(adj_type: { }),
        goal => type(adj_type: { })).
```

with:

```
type(adj_type:
        adj_focus => boolean([{plus/minus/nil}]).
```

Since pronominality and cliticization are irrelevant for modifier ordering, we regard the discourse function focus as the only factor that should be taken into consideration. The word order constraint for the two grammatical functions is quite simple: a goal adjunct may precede a source adjunct only if the latter is focussed and the first remains in the topic, cf.

(6) Peter faehrt von Saarbruecken nach Berlin
Peter goes from Saarbruecken to Berlin

vs.

(7) Peter faehrt nach Berlin von Saarbruecken

Both orderings are licensed by instances of the general head-adjunct structure, instantiated with the adjunct type information (source/goal adjunct). The constraint stated above is encoded in the rule in the same way as the left and right context restrictions for complements are.

As the next step, we intend to augment the rules with annotations that would permit to treat linear precedence constraints for complements and adjuncts at once. The general mechanism presented in our paper can be adopted for this purpose by instantiating the complement information in adjunct rules and vice versa.

8. Conclusion

By adapting the method described in Part I of the report, we have developed a grammar which handles most of the ordering phenomena among complements and adjuncts in the German Mittelfeld.

We have successfully dealt with conflicts and interactions of different LP rules, which have been noted in the linguistic literature, but have not been treated appropriately in computational grammars.

With respect to efficiency, it turns out that our grammar with LP constraints has a better runtime behaviour than a grammar which allows all orderings, because our grammar leads to a reduction of the search space, unlike previous implementations which check LP constraints only after parsing or generation.

By implementation of a grammar for German Mittelfeld ordering phenomena, we have shown that a formalism like ALEP that is based on phrase structure rules, can well be used for the treatment of languages which allow for partially free word order.

We know of no other implemented grammar that provides an equally detailed coverage of the complex word order regularities of German or similar languages. Thus, our implementation serves as an example for how a "lean" formalism like ALEP can be utilized for the description of complex grammatical phenomena.

REFERENCES

Engelkamp et al. (1992)
J. Engelkamp, G. Erbach, H. Uszkoreit. Handling Linear Precedence Constraints by Unification. Proceedings 30th Annual Meeting of the Association for Computational Linguistics, Newark, Delaware.

Uszkoreit, H. (1987).
Word Order and Constituent Structure in German. Stanford, CA, Center for the Study of Language and Information.

APPENDIX A

Macros

```
# define AGR( CASE, GEND, PER, NUM ,DECL)
     infl_type: {   case => CASE,
                gend => GEND,
                per => PER,
                num => NUM,
                decl => DECL }

# define MIN(FC,DET)
     minor_type: {fc => FC, det => DET}

# define INFL_HEAD(MAJ,INFL,MOD,LPH)
      infl_head_type: {infl => INFL,
                          maj => MAJ,
                       mod => MOD,
                       lp_head => LPH }

# define V_HEAD(INIT,INFL, VFORM ,LPH)
      verbal_head_type: {infl => INFL,
                           initial => INIT,

                           maj => verb,

                           mod => mod_type: {val =>nil_type,
                                             function => nil },
                           vform => VFORM,
                           lp_head => LPH }


#define MOD(FUN,VAL)
                mod_type: {val => VAL,
                           function => FUN }
#define NO_MOD()
                mod_type: {val =>nil_type,
                           function => nil }


# define P_HEAD(PFORM,MOD,LPH )
      prep_head_type: {pform => PFORM,
                        mod => MOD ,
                        lp_head => LPH }

# define COMP_HEAD(MOD,LPH )
      complementizer_head_type: { mod => MOD ,
                                  lp_head => LPH }
```

```
# define HEAD(MAJ,MOD,LPH)
      head_type: {maj => MAJ,
                  mod => MOD,
                  lp_head => LPH }

# define SIGN( STRUCT, ORDER, SYN , NONLOC, SEM)
     sign_type: {order => ORDER,
                 syn => SYN,
                 nonloc => NONLOC,
                 sem => SEM,
                 structure => STRUCT }

# define ORDER( STRING, REST )
     order_type: {graph => graph_type: {string => STRING,
                                        rest => REST}}

# define SYN(HEAD,MIN,ADJS,SUBCAT)
     syn_type: {head => HEAD,
                min => MIN,
                adjuncts => ADJS,
                subcat => SUBCAT}

# define NO_SUBCAT()
     subcat_type: {subj => NO_COMP(),
                   obj  => NO_COMP(),
                   obj2 => NO_COMP(),
                   pobj => NO_COMP() }

# define SUBCAT(SUBJ,OBJ,OBJ2,POBJ)
     subcat_type: {subj => SUBJ,
                   obj  => OBJ,
                   obj2 => OBJ2,
                   pobj => POBJ }
#define ADJUNCTS(SOURCE,GOAL,INSTR)
     adjuncts_type: {source => SOURCE,
                     goal => GOAL,
                     instr => INSTR }

#define ADJ(FOCUS)
     adj_type: {focus => FOCUS}

#define NO_ADJS()
     ADJUNCTS(ADJ(nil),ADJ(nil),ADJ(nil))

# define EMPTY_SUBCAT()

SUBCAT(COMP(_,plus,_),COMP(_,plus,_),COMP(_,plus,_),COMP(_,plus
,_))

# define NO_COMP()
     comp_type: {bound_off => plus,
                      val => nil_type,
                      lp_relevant => LP(nil,nil,nil) }
```

```
# define COMP(VAL,BO,LPR)
     comp_type: { bound_off => BO,
                  val => VAL,
                  lp_relevant => LPR}

# define LP(FOCUS,PRO,CLITIC)
     lp_relevant_type: {pro => PRO,
                        focus => FOCUS ,
                        clitic  => CLITIC}

# define NOUN(STRUCT,ORDER,INFL,FC,DET,LPH,SUBCAT,SEM)
SIGN(STRUCT,ORDER,SYN(INFL_HEAD(noun,INFL,NO_MOD(),LPH),MIN(FC,
DET),_,SUBCAT),[],SEM)

# define
LEX_NOUN(STRING,CASE,GEND,NUM,FC,FOCUS,PRO,ES,SUBCAT,SEM)
NOUN(lex,ORDER([STRING|REST],REST),AGR(CASE,GEND,third,NUM,_),F
C,minus,LP(_,PRO,ES),SUBCAT,SEM)

# define COMMON_NOUN(STRING,CASE,GEND,NUM,SEM)
LEX_NOUN(STRING,CASE,GEND,NUM,minus,_,minus,minus,NO_SUBCAT(),
SEM)

# define PROPER_NOUN(STRING,CASE,GEND,NUM,SEM)
LEX_NOUN(STRING,CASE,GEND,NUM,plus,_,minus,minus,NO_SUBCAT(),
SEM)

# define NP(INFL,SEM)
     NOUN(_,_,INFL,plus,_,_,_,SEM)

# define PERSPRO(STRING,CASE,GEND,PER,NUM,ES,SEM)
NOUN(lex,ORDER([STRING|REST],REST),AGR(CASE,GEND,PER,NUM,_),
plus,minus,LP(_,plus,ES),NO_SUBCAT(),SEM)

# define DET(STRING,CASE,GEND,NUM,DECL,FOCUS,SEM)
SIGN(lex,ORDER([STRING|REST],REST),SYN(INFL_HEAD(adj,@AGRF
AGR(CASE,GEND,third,NUM,DECL), MOD(spec,NOUN(_, _, AGRF , _,
minus,_,_,ARG_SEM)),LP(_,minus,minus)),MIN(plus,plus),NO_ADJS()
,NO_SUBCAT() ),[],TERM(SEM,L_FIRST(ARG_SEM,EOL())))

# define ADJECTIVE(STRING,CASE,GEND,NUM,DECL,SEM)
SIGN(lex,ORDER([STRING|REST],REST),SYN(INFL_HEAD(adj,@AGRF
AGR(CASE,GEND,third,NUM,DECL),MOD(prenom,NOUN(_,_,AGRF,FC,minus
,LPH,_,ARG_SEM)),LPH),MIN(FC,minus),_,NO_SUBCAT()),[],TERM(
SEM,L_FIRST(ARG_SEM,EOL())))


# define PREP_NMOD(STRING,PFORM,PCASE,FUN,SEM)
SIGN(lex,ORDER([STRING|REST],REST),SYN(P_HEAD(PFORM,MOD(FUN,
NOUN(_,_,_,_,minus,_,_,M_SEM)),_),MIN(plus,nil),NO_ADJS(),
SUBCAT(NO_COMP(),COMP(NP(AGR(PCASE,_,_,_,_),C_SEM),minus,_),
NO_COMP(),NO_COMP())),[],TERM(SEM,L_FIRST(M_SEM,L_NEXT(C_SEM,
EOL()))))
```

```
# define PREP(STRING,PFORM,PCASE,FUN,SEM)
SIGN(lex,ORDER([STRING|REST],REST),SYN(P_HEAD(PFORM,MOD(FUN,
VERB(_,_,fin,_,_,_,_,[],M_SEM)),_),MIN(plus,nil),NO_ADJS(),
SUBCAT(NO_COMP(),COMP(NP(AGR(PCASE,_,_,_,_),C_SEM),minus,_),
NO_COMP(),NO_COMP())),[],TERM(SEM,L_FIRST(C_SEM,L_NEXT(M_SEM,
EOL()))))

# define VERB(PER,NUM,VFORM,INIT,FC,ADJS,SC,NONLOC,SEM)
SIGN(_,_,SYN(V_HEAD(INIT,AGR(nil,nil,PER,NUM,nil),VFORM,_),MIN(
FC,nil),ADJS,SC),NONLOC,SEM)

# define LEX_VERB(STRING,PER,NUM,VFORM,SC,NONLOC,SEM)
SIGN(lex,ORDER([STRING|REST],REST),SYN(V_HEAD(INIT,AGR(nil,nil,
PER,NUM,nil),VFORM,_),MIN(INIT,nil),NO_ADJS(),SC),NONLOC,SEM)


# define I_VERB(STRING,PER,NUM,VFORM,SEM)
LEX_VERB(STRING,PER,NUM,VFORM,SUBCAT(COMP(NP(AGR(nom,_,PER,NUM,
_),SUBJ_SEM),minus,LP(nil,nil,nil)),NO_COMP(),NO_COMP(),
NO_COMP()),[],TERM(SEM,L_FIRST(SUBJ_SEM,EOL())))

# define T_VERB(STRING,PER,NUM,VFORM,SEM)
LEX_VERB(STRING,PER,NUM,VFORM,SUBCAT(COMP(NP(AGR(nom,_,PER,NUM,
_),SUBJ_SEM),minus,LP(nil,nil,nil)) ,
COMP(NP(AGR(acc,_,_,_,_),DOBJ_SEM),minus,LP(nil,nil,nil)),
NO_COMP(),NO_COMP()),[],TERM(SEM,L_FIRST(SUBJ_SEM,L_NEXT(
DOBJ_SEM, EOL()))))

# define DIT_VERB(STRING,PER,NUM,VFORM,SEM)
LEX_VERB(STRING,PER,NUM,VFORM,SUBCAT(COMP(NP(AGR(nom,_,PER,NUM,
_),SUBJ_SEM),minus,LP(nil,nil,nil)) ,
COMP(NP(AGR(acc,_,_,_,_),DOBJ_SEM),minus,LP(nil,nil,nil)),COMP(
NP(AGR(dat,_,_,_,_),IOBJ_SEM),minus,LP(nil,nil,nil)),NO_COMP())
, [],TERM(SEM,L_FIRST(SUBJ_SEM,L_NEXT(IOBJ_SEM,L_NEXT(DOBJ_SEM,
EOL()))))

# define COMPLEMENTIZER(STRING)
SIGN(lex,ORDER([STRING|REST],REST),SYN(COMP_HEAD(MOD(spec,
VERB(_,_,fin,minus,minus,_,EMPTY_SUBCAT(),[],SEM)),_),MIN(plus,
nil),NO_ADJS(),NO_SUBCAT()),[],SEM)

# define LIST(ELEMENTS)
     [ ELEMENTS ]

# define L_FIRST(EL,REST)
      EL REST

# define L_NEXT(EL,REST)
      ,EL REST

# define EOL()

# define L_TAIL( REST )
     | REST
```

```
# define TERM(FUNCTOR,ARGUMENTS)
     FUNCTOR(ARGUMENTS)
```

APPENDIX B

Lexicon

```
#include "macros"

lexicon:[cat,analysis].

dass~
     COMPLEMENTIZER(dass).
er~
     PERSPRO(er,nom,mas,third,sg,minus,pro(sg(_))).
ihm~
     PERSPRO(ihm,dat,(mas;ntr),third,sg,minus,pro(sg(_))).
ihn~
     PERSPRO(ihn,acc,mas,third,sg,minus,pro(sg(_))).
es~
     PERSPRO(es,(nom;acc),ntr,third,sg,plus,pro(sg(_))).
sie~
     PERSPRO(sie,(nom;acc),fem,third,sg,minus,pro(sg(_))).
ihr~
     PERSPRO(ihr,dat,fem,third,sg,minus,pro(sg(_))).
mann~
     COMMON_NOUN(mann, ~gen, mas, sg, mann(sg(_))).
mannes~
     COMMON_NOUN(mannes, gen, mas, sg, mann(sg(_))).
maenner~
     COMMON_NOUN(maenner, ~dat, mas, pl , mann(pl(_))).
maennern~
     COMMON_NOUN(maennern, dat, mas, pl , mann(pl(_))).
buch~
     COMMON_NOUN(buch, ~gen, ntr, sg , buch(sg(_)) ).
buches~
     COMMON_NOUN(buches, gen, ntr, sg , buch(sg(_)) ).
buecher~
     COMMON_NOUN(buecher, ~dat, ntr, pl , buch(pl(_)) ).
buechern~
     COMMON_NOUN(buechern, dat, ntr, pl , buch(pl(_)) ).
peter~
     PROPER_NOUN(peter, ~gen, mas, sg , peter(sg(_)) ).
berlin~
     PROPER_NOUN(berlin, ~gen, ntr, sg , berlin(sg(_)) ).
muenchen~
     PROPER_NOUN(muenchen, ~gen, ntr, sg , muenchen(sg(_)) ).
mit~
     PREP(mit,mit,dat,instr,mit).
von~
     PREP(von,von,dat,source,von).
nach~
```

```
     PREP(nach,nach,dat,goal,nach).
schlaeft~
     I_VERB(schlaeft,third,sg,fin,schlafen).

faehrt~
     I_VERB(faehrt,third,sg,fin,fahren).
sieht~
     T_VERB(sieht,third,sg,fin,sehen).
gibt~
     DIT_VERB(gibt,third,sg,fin,geben).

der1~
     DET(der, nom, mas, sg, weak, plus, the ).
der2~
     DET(der, (gen;dat), fem, sg, weak, plus, the ).
der3~
     DET(der, gen, (mas;fem;ntr), pl , weak, plus, the ).
das1~
     DET(das, (nom;acc), ntr, sg, weak, plus, the ).
die1~
     DET(die, (nom;acc), fem, sg, weak, plus, the ).
die2~
     DET(die, (nom;acc), (mas;fem;ntr), pl , weak, plus, the ).
des1~
     DET(des, gen, (mas;ntr), sg, weak, plus, the ).
dem1~
     DET(dem, dat, (mas;ntr), sg, weak, plus, the ).
den1~
     DET(den, acc, mas, sg, weak, plus, the ).
den2~
     DET(den, dat,(mas;fem;ntr), pl, weak, plus, the ).
ein1~
     DET(ein, nom, mas, sg, strong, minus, exists ).
ein2~
     DET(ein, (nom;acc), ntr, sg, strong, minus, exists ).
einer2~
     DET(einer, (gen;dat), fem, sg, weak, minus, exists ).
eine1~
     DET(eine, (nom;acc), fem, sg, weak, minus, exists ).
eines1~
     DET(eines, gen, (mas;ntr), sg, weak, minus, exists ).
einem1~
     DET(einem, dat, (mas;ntr), sg, weak, minus, exists ).
einen1~
     DET(einen, acc, mas, sg, weak, minus, exists ).
schoene1~
     ADJECTIVE(schoene,nom,mas,sg,weak, schoen).
schoene2~
     ADJECTIVE(schoene,(nom;acc),ntr,sg,weak, schoen).
schoene3~
     ADJECTIVE(schoene,(nom;acc),fem,sg,(weak;strong), schoen).
schoenen1~
     ADJECTIVE(schoenen,(gen;dat),(mas;fem;ntr),sg,weak,
schoen).
```

```
schoenen2~
     ADJECTIVE(schoenen,acc,mas,sg,(weak;strong), schoen).
schoenen3~
     ADJECTIVE(schoenen,gen,(mas;ntr),sg,strong, schoen).
schoenen4~
     ADJECTIVE(schoenen,(nom;gen;dat;acc),(mas;fem;ntr),pl,weak
, schoen).
schoenen5~
     ADJECTIVE(schoenen,dat,(mas;fem;ntr),pl,strong, schoen).
schoenem1~
     ADJECTIVE(schoenem,dat,(mas;ntr),sg,strong, schoen).
schoener1~
     ADJECTIVE(schoener,nom,mas,sg,strong, schoen).
schoener2~
     ADJECTIVE(schoener,(gen;dat),fem,sg,strong, schoen).
schoener3~
     ADJECTIVE(schoener,gen,(mas;fem;ntr),pl,strong, schoen).
schoenes~
     ADJECTIVE(schoenes,nom,mas,sg,strong, schoen).
gute1~
     ADJECTIVE(gute,nom,mas,sg,weak, gut).
gute2~
     ADJECTIVE(gute,(nom;acc),ntr,sg,weak, gut).
gute3~
     ADJECTIVE(gute,(nom;acc),fem,sg,(weak;strong), gut).
guten1~
     ADJECTIVE(guten,(gen;dat),(mas;fem;ntr),sg,weak, gut).
guten2~
     ADJECTIVE(guten,acc,mas,sg,(weak;strong), gut).
guten3~
     ADJECTIVE(guten,gen,(mas;ntr),sg,strong, gut).
guten4~
     ADJECTIVE(guten,(nom;gen;dat;acc),(mas;fem;ntr),pl,weak,
gut).
guten5~
     ADJECTIVE(guten,dat,(mas;fem;ntr),pl,strong, gut).
gutem~
     ADJECTIVE(gutem,dat,(mas;ntr),sg,strong, gut).
guter1~
     ADJECTIVE(guter,nom,mas,sg,strong, gut).
guter2~
     ADJECTIVE(guter,(gen;dat),fem,sg,strong, gut).
guter3~
     ADJECTIVE(guter,gen,(mas;fem;ntr),pl,strong, gut).
gutes~
     ADJECTIVE(gutes,nom,mas,sg,strong, gut).
```

APPENDIX C

Rules

```
# include "macros"

grammar:[cat,analysis].
```

filler_head_rule =

```
SIGN(filler_head,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
         MINOR,
         ADJS,
         SUBCAT(C1,C2,C3,C4)),
     [],
     SEM)
                    ->
     [

     @FILLER  SIGN(_,
                   ORDER(S0,S1),
                   SYN(HEAD(_,_,_),_,_,_),
                   [],
                   _),

     SIGN(_,
          ORDER(S1,S2),
          SYN(@HEAD_FEAT V_HEAD(plus,_,_,_),
          MINOR,
          ADJS,
          SUBCAT(@C1 COMP(_,plus,_),
                 @C2 COMP(_,plus,_),
                 @C3 COMP(_,plus,_),
                 @C4 COMP(_,plus,_))),
          [FILLER],
          SEM) ] head 2.
```

subj_trace_rule =

```
SIGN(trace,
     ORDER(S0,S1),
     SYN( HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(COMP(VAL,
                      plus,
                      LPR),
```

```
                C2,
                C3,
                C4)),
     [VAL],
     SEM)      ->
     [

          SIGN(_,
               ORDER(S0,S1),
               SYN(@HEAD_FEAT V_HEAD(_,_,_,_),
               MINOR,
               ADJS,
               SUBCAT(COMP(VAL,
                           minus,
                           LPR),
                      @C2 COMP(_,plus,_),
                      @C3 COMP(_,plus,_),
                      @C4 COMP(_,plus,_))),
               [],
               SEM)
     ] head 1.
```

obj_trace_rule =

```
SIGN(trace,
     ORDER(S0,S1),
     SYN( HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(C1,
                 COMP(VAL,plus, LPR),
                 C3,
                 C4)),
     [VAL],
     SEM)

               ->


     [         SIGN(_,
               ORDER(S0,S1),
               SYN(@HEAD_FEAT V_HEAD(_,_,_,_),
               MINOR,
               ADJS,
               SUBCAT(@C1 COMP(_,plus,_),
                      COMP(VAL,
                           minus,
                           LPR),
                      @C3 COMP(_,plus,_),
                      @C4 COMP(_,plus,_))),
               [],
               SEM)
     ] head 1.
```

obj2_trace_rule =

```
SIGN(trace,
     ORDER(S0,S1),
     SYN( HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(C1,
                 C2,
                 COMP(VAL,plus, LPR),
                 C4)),
     [VAL],
     SEM)      ->

     [         SIGN(_,
               ORDER(S0,S1),
               SYN(@HEAD_FEAT V_HEAD(_,_,_,_),
               MINOR,
               ADJS,
               SUBCAT(@C1 COMP(_,plus,_),
                      @C2 COMP(_,plus,_),
                      COMP(VAL,
                           minus,
                           LPR),
                      @C4 COMP(_,plus,_))),
               [],
               SEM)      ]         head 1.
```

head_subj_rule1 =

```
SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(   COMP(VAL,plus,LPR),
                    C2,
                    C3,
                    C4)),
     NONLOC,
     SEM)
               ->

     [    SIGN(_,
               ORDER(S0,S1),
               SYN(@HEAD_FEAT V_HEAD(plus,_,_,_),
                    MINOR,
                    ADJS,
                    SUBCAT(   COMP(VAL,minus,_),
```

```
                              @C2 COMP(_,_,LP(nil,nil,nil)),
                              @C3COMP(_,_,LP(nil,nil,nil)),
                              C4)),
                    NONLOC,
                    SEM),

               @VAL SIGN(_,ORDER(S1,S2),
                         SYN(HEAD( _,
                                   _,
                                   @LPR LP(_,plus,_))
                                  ,_,_,_)
                         ,_,_)
          ] head 1.
```

head_subj_rule2 =

```
SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(COMP(VAL,plus,LPR),
                    C2,
                    C3,
                    C4)),
     NONLOC,
     SEM)
          ->

     [    SIGN(_,
               ORDER(S0,S1),
               SYN(@HEAD_FEAT V_HEAD(plus,_,_,_),
                    MINOR,
                    ADJS,
                    SUBCAT(COMP(VAL,minus,_),
                           @C2 COMP(_,_,LP(minus,minus,_)),
                           @C3 COMP(_,_,LP(minus,minus,_)),
                           C4)),
     NONLOC,
     SEM),

          @VAL SIGN(_,
                    ORDER(S1,S2),
                    SYN(HEAD( _,
                              _,
                    @LPR LP(plus,minus,_))
                         ,_,_,_)
               ,_,_)

] head 1.
```

head_subj_rule3 =

```
SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(COMP(VAL,plus,LPR),C2,C3,C4)),
     NONLOC,
     SEM)
          ->

     [    SIGN(_,
               ORDER(S0,S1),
               SYN(@HEAD_FEAT V_HEAD(plus,_,_,_),
               MINOR,
               ADJS,
               SUBCAT(COMP(VAL,minus,_),
                         @C2 COMP(_,_,LP(minus,minus,_)),
                         @C3 COMP(_,_,LP(_,~minus,_)),
                         C4)),
          NONLOC,
          SEM),


          @VAL SIGN(_,
                    ORDER(S1,S2),
                    SYN(HEAD(_,_,@LPR LP(plus,minus,_))
                              ,_,_,_)
                    ,_,_)] head 1.
```

head_subj_rule4 =

```
SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(COMP(VAL,plus,LPR),C2,C3,C4)),
     NONLOC,
     SEM)  ->

     [    SIGN(_,
               ORDER(S0,S1),
               SYN(@HEAD_FEAT V_HEAD(plus,_,_,_),
                    MINOR,
                    ADJS,
                    SUBCAT(COMP(VAL,minus,_),
                              @C2 COMP(_,_,LP(_,~minus,_)),
```

```
                              @C3 COMP(_,_,LP(minus,minus,_)),
                              C4)),
                    NONLOC,
                    SEM),

               @VAL SIGN(_,
                         ORDER(S1,S2),
                         SYN(HEAD(_,_,@LPR LP(plus,minus,_))
                              ,_,_,_)
                         ,_,_)     ] head 1.
```

head_subj_rule5 =

```
SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(COMP(VAL,plus,LPR),C2,C3,C4)),
     NONLOC,
     SEM) ->

     [    SIGN(_,
               ORDER(S0,S1),
               SYN(@HEAD_FEAT V_HEAD(plus,_,_,_),
                    MINOR,
                    ADJS,
                    SUBCAT(COMP(VAL,minus,_),
                              @C2 COMP(_,_,LP(_,~minus,_)),
                              @C3 COMP(_,_,LP(_,~minus,_)),
                              C4)),
               NONLOC,
               SEM),

          @VAL SIGN(_,ORDER(S1,S2),SYN(HEAD(_,_,@LPR
LP(_,minus,_)),_,_,_),_,_)     ] head 1.
```

subj_head_rule =

```
SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(COMP(VAL,plus, LPR),C2,C3,C4)),NONLOC,SEM) ->

     [    @VAL SIGN(_,
               ORDER(S0,S1),
               SYN(HEAD(_,_,LPR),_,_,_)
               ,_,_),

          SIGN(_,
               ORDER(S1,S2),
               SYN(@HEAD_FEAT V_HEAD(minus,_,_,_),
```

```
                    MINOR,
                    ADJS,
                    SUBCAT(COMP(VAL,minus,_),C2,C3,C4)),
               NONLOC,
               SEM) ] head 2.
```

head_source_adj_rule1 =

```
SIGN(head_adj,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,MINOR,ADJUNCTS(ADJ(_),ADJ(nil),ADJ(INSTR)),
     SC),
     [],
     SEM)
          ->

     [    @ HEAD_DTR SIGN(~filler_head,
                          ORDER(S0,S1),
                          SYN(@HEAD_FEAT V_HEAD(plus,_,_,_),
                               MINOR,
                               ADJUNCTS(ADJ(_),
                                        ADJ(nil),
                                        ADJ(INSTR)),
                               SC),
                               [],
                               _),

          SIGN(_,
               ORDER(S1,S2),
               SYN(HEAD(_,MOD(source,HEAD_DTR),_),_,_,_),
               _,
               SEM) ] head 2.
```

head_source_adj_rule2 =

```
SIGN(head_adj,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,MINOR,ADJUNCTS(ADJ(plus),ADJ(INSTR)),SC),
     [],
     SEM)

          ->

     [    @ HEAD_DTR SIGN(~filler_head,
                          ORDER(S0,S1),
                          SYN(@HEAD_FEAT V_HEAD(plus,_,_,_),
                               MINOR,
                               ADJUNCTS(ADJ(_),
                                        ADJ(plus),
                                        ADJ(INSTR)),
                               SC),
                          [],
                          _),

               SIGN(_,
```

```
                ORDER(S1,S2),
                SYN(HEAD(_,MOD(source,HEAD_DTR),_),_,_,_),
                _,
                SEM) ] head 2.
```

source_adj_head_rule =

```
SIGN(head_adj,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJUNCTS(ADJ(~nil),ADJ(GOAL),ADJ(INSTR)),
          SC),
     [],
     SEM)      ->

     [    SIGN(_,
          ORDER(S0,S1),
          SYN(HEAD(_,MOD(source,HEAD_DTR),_),_,_,_),
          _,
          SEM),


          @ HEAD_DTR SIGN(~filler_head,
                         ORDER(S1,S2),
                         SYN(HEAD_FEAT,
                              MINOR,
                              ADJUNCTS(ADJ(_),
                                       ADJ(GOAL),
                                       ADJ(INSTR)),
                              SC),
                         [],_)] head 1.
```

head_goal_adj_rule1 =

```
SIGN(head_adj,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJUNCTS(ADJ(SOURCE),ADJ(_),ADJ(INSTR)),SC),
     [],
     SEM)      ->

     [    @ HEAD_DTR SIGN(~filler_head,
                       ORDER(S0,S1),
                       SYN(@HEAD_FEAT V_HEAD(plus,_,_,_),
                            MINOR,
                            ADJUNCTS(ADJ(SOURCE),
                                     ADJ(_),
                                     ADJ(INSTR)),
                            SC),
                       [],
                       _),
```

```
          SIGN(_,
               ORDER(S1,S2),
               SYN(HEAD(_,MOD(goal,HEAD_DTR),_),_,_,_)
               ,_,
               SEM) ] head 2.
goal_adj_head_rule1 =

SIGN(head_adj,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJUNCTS(ADJ(nil),ADJ(_),ADJ(INSTR)),
          SC),
     [],
     SEM)      ->

     [    SIGN(_,
          ORDER(S0,S1),
          SYN(HEAD(_,MOD(goal,HEAD_DTR),_),_,_,_)
          ,_,
          SEM),

          @ HEAD_DTR SIGN(~filler_head,
                          ORDER(S1,S2),
                          SYN(HEAD_FEAT,
                               MINOR,
                               ADJUNCTS(ADJ(nil),
                                        ADJ(_),
                                        ADJ(INSTR)),
                               SC),
                          [],
                          _)   ] head 1.

goal_adj_head_rule2 =

SIGN(head_adj,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJUNCTS(ADJ(plus),ADJ(minus),ADJ(INSTR)),
          SC),
     [],
     SEM)  ->

     [    SIGN(_,
          ORDER(S0,S1),
          SYN(HEAD(_,MOD(goal,HEAD_DTR),_),_,_,_)
          ,_,
          SEM),

          @ HEAD_DTR SIGN(~filler_head,
                          ORDER(S1,S2),
                          SYN(HEAD_FEAT,
                               MINOR,
```

```
                              ADJUNCTS(ADJ(plus),
                                       ADJ(_),
                                       ADJ(INSTR)),
                              SC),
                         [],_)] head 1.
```

spec_head_rule =

```
SIGN(head_spec,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,MINOR,_,SC),
     NONLOC,
     SEM) ->

     [    SIGN(_,
               ORDER(S0,S1),
               SYN(HEAD(_,MOD(spec,HEAD_DTR),_),MINOR,_,_)
               ,_,
               SEM),

          @ HEAD_DTR SIGN(_,
                     ORDER(S1,S2),
                     SYN(HEAD_FEAT,_,_,SC),
                     NONLOC,
                     _)   ] head 1.
```

head_obj_rule1 =

```
SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(C1,COMP(VAL,plus,LPR),C3,C4)),
          NONLOC,
          SEM) ->

     [    SIGN(_,
               ORDER(S0,S1),
               SYN(@HEAD_FEAT V_HEAD(plus,_,_,_),
                         MINOR,
                         ADJS,
                         SUBCAT(@C1 COMP(_,_,LP(_,_,_)),
                              COMP(VAL,minus,_),
                              @C3 COMP(_,_,LP(nil,nil,nil)),
                              C4)),
               NONLOC,
               SEM),

          @VAL SIGN(_,
                     ORDER(S1,S2),
                     SYN(HEAD(_,_,@LPR LP(_,plus,_)),_,_,_)
                     ,_,_)] head 1.
```

head_obj_rule2 =

```
SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(C1,COMP(VAL,plus,LPR),C3,C4)),
          NONLOC,
          SEM) ->

     [    SIGN(_,
               ORDER(S0,S1),
               SYN(@HEAD_FEAT V_HEAD(plus,_,_,_),
                         MINOR,
                         ADJS,
                         SUBCAT(@C1 COMP(_,_,LP(_,_,_)),
                              COMP(VAL,minus,_),
                              @C3 COMP(_,_,LP(minus,plus,_)),
                              C4)),
               NONLOC,
               SEM),

          @VAL SIGN(_,
                    ORDER(S1,S2),
                    SYN(HEAD(_,_,@LPR LP(plus,plus,minus))
                              ,_,_,_)
                    ,_,_)] head 1.
```

head_obj_rule3 =

```
SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(C1,COMP(VAL,plus,LPR),C3,C4)),
          NONLOC,
          SEM)
                ->

     [    SIGN(_,
               ORDER(S0,S1),
               SYN(@HEAD_FEAT V_HEAD(plus,_,_,_),
                         MINOR,
                         ADJS,
                         SUBCAT(@C1 COMP(_,_,LP(_,_,_)),
                              COMP(VAL,minus,_),
                              @C3 COMP(_,_,LP(_,_,_)),
```

```
                              C4)),
                NONLOC,
                SEM),


          @VAL SIGN(_,
                     ORDER(S1,S2),
                     SYN(HEAD(_,_,@LPR LP(_,minus,_))
                          ,_,_,_)
                     ,_,_)] head 1.
```

obj_head_rule1 =

```
SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(C1,COMP(VAL,plus,LPR),C3,C4)),
     NONLOC,
     SEM)

          ->

     [@VAL SIGN(_,
                ORDER(S0,S1),
                SYN(HEAD(_,_,@LPR LP(_,plus,_)),_,_,_)
                ,_,_),

          SIGN(_,
                ORDER(S1,S2),
                SYN(@HEAD_FEAT V_HEAD(minus,_,_,_),
                     MINOR,
                     ADJS,
                     SUBCAT(@C1 COMP(_,_,LP(_,~plus,_)),
                          COMP(VAL,minus,_),
                          @C3 COMP(_,_,LP(_,_,_)),
                          C4)),
                NONLOC,
                SEM) ] head 2.
```

obj_head_rule2 =

```
SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(C1,COMP(VAL,plus,LPR),C3,C4)),
     NONLOC,
```

```
     SEM)

          ->


     [@VAL SIGN(_,
               ORDER(S0,S1),
               SYN(HEAD(_,_,@LPR LP(minus,minus,_))
                    ,_,_,_)
               ,_,_),

          SIGN(_,
               ORDER(S1,S2),
               SYN(@HEAD_FEAT V_HEAD(minus,_,_,_),
                    MINOR,
                    ADJS,
                    SUBCAT(@C1 COMP(_,_,LP(~minus,~plus,_)),
                              COMP(VAL,minus,_),
                              @C3 COMP(_,_,LP(plus,minus,_)),
                              C4)),
               NONLOC,
               SEM) ] head 2.


obj_head_rule3 =

SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(C1,COMP(VAL,plus,LPR),C3,C4)),
     NONLOC,
     SEM)

          ->

     [@VAL SIGN(_,
               ORDER(S0,S1),
               SYN(HEAD(_,_,@LPR LP(_,minus,_))
                    ,_,_,_)
               ,_,_),


          SIGN(_,
               ORDER(S1,S2),
               SYN(@HEAD_FEAT V_HEAD(minus,_,_,_),
                    MINOR,
                    ADJS,
                    SUBCAT(@C1 COMP(_,_,LP(nil,nil,nil)),
                              COMP(VAL,minus,_),
                              @C3 COMP(_,_,LP(nil,nil,nil)),
                              C4)),
```

```
                NONLOC,
                SEM) ] head 2.


head_obj2_rule1 =

SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(C1,C2,COMP(VAL,plus,LPR),C4)),
     NONLOC,
     SEM)
          ->

     [    SIGN(_,
          ORDER(S0,S1),
          SYN(@HEAD_FEAT V_HEAD(plus,_,_,_),
               MINOR,
               ADJS,
               SUBCAT(@C1 COMP(_,_,LP(_,_,_)),
                         @C2 COMP(_,_,LP(_, ~minus,_)),
                         COMP(VAL,minus,_),
                         C4)),
          NONLOC,
          SEM),

          @VAL SIGN(_,
                    ORDER(S1,S2),
                    SYN(HEAD(_,_,@LPR LP(_,_,_)),_,_,_),_,_)
     ] head 1.


head_obj2_rule2 =

SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(C1,C2,COMP(VAL,plus,LPR),C4)),
     NONLOC,
     SEM)

          ->

     [    SIGN(_,
          ORDER(S0,S1),
          SYN(@HEAD_FEAT V_HEAD(plus,_,_,_),
               MINOR,
               ADJS,
               SUBCAT(@C1 COMP(_,_,LP(_,_,_)),
```

```
                        @C2 COMP(_,_,LP(minus,minus,_)),
                        COMP(VAL,minus,_),
                        C4)),
          NONLOC,
          SEM),
          @VAL SIGN(_,
                    ORDER(S1,S2),
                    SYN(HEAD(_,_,@LPR LP(plus,minus,_))
                         ,_,_,_)
                    ,_,_)] head 1.
```

obj2_head_rule1 =

```
SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(C1,C2,COMP(VAL,plus,LPR),C4)),
     NONLOC,
     SEM) ->

     [    @VAL SIGN(_,
               ORDER(S0,S1),
               SYN(HEAD(_,_,@LPR LP(minus,plus,_))
                    ,_,_,_)
               ,_,_),


          SIGN(_,
               ORDER(S1,S2),
               SYN(@HEAD_FEAT V_HEAD(minus,_,_,_),
                    MINOR,
                    ADJS,
                    SUBCAT(@C1 COMP(_,_,LP(_,~plus,_)),
                              @C2 COMP(_,_,LP(plus,plus,_)),
                              COMP(VAL,minus,_),
                              C4)),
               NONLOC,
               SEM) ] head 2.
```

obj2_head_rule2 =

```
SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(C1,C2,COMP(VAL,plus,LPR),C4)),
     NONLOC,
     SEM)      ->
```

```
     [    @VAL SIGN(_,
               ORDER(S0,S1),
               SYN(HEAD(_,_,@LPR LP(_,plus,_))
                    ,_,_,_)
               ,_,_),
          SIGN(_,
               ORDER(S1,S2),
               SYN(@HEAD_FEAT V_HEAD(minus,_,_,_),
                    MINOR,
                    ADJS,
                    SUBCAT(   @C1 COMP(_,_,LP(_,~plus,_)),
                              @C2 COMP(_,_,LP(_,~plus,_)),
                              COMP(VAL,minus,_),
                              C4)),
                    NONLOC,
                    SEM) ] head 2.
```

obj2_head_rule3 =

```
SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(C1,C2,COMP(VAL,plus,LPR),C4)),
     NONLOC,
     SEM)
          ->

     [    @VAL SIGN(_,
               ORDER(S0,S1),
               SYN(HEAD(_,_,@LPR LP(_,minus,_))
                    ,_,_,_)
               ,_,_),

          SIGN(_,
               ORDER(S1,S2),
               SYN(@HEAD_FEAT V_HEAD(minus,_,_,_),
                    MINOR,
                    ADJS,
                    SUBCAT(@C1 COMP(_,_,LP(nil,nil,nil)),
                              @C2 COMP(_,_,LP(_,~plus,_)),
                              COMP(VAL,minus,_),
                              C4)),
               NONLOC,
               SEM) ] head 2.
```

obj2_head_rule4 =

```
SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
```

```
          ADJS,
          SUBCAT(C1,C2,COMP(VAL,plus,LPR),C4)),
     NONLOC,
     SEM)
          ->

     [    @VAL SIGN(_,
               ORDER(S0,S1),
               SYN(HEAD(_,_,@LPR LP(minus,minus,_))
                    ,_,_,_)
               ,_,_),

          SIGN(_,
               ORDER(S1,S2),
               SYN(@HEAD_FEAT V_HEAD(minus,_,_,_),
                    MINOR,
                    ADJS,
                    SUBCAT(@C1 COMP(_,_,LP(plus,minus,_)),
                              @C2 COMP(_,_,LP(_,~plus,_)),
                              COMP(VAL,minus,_),
                              C4)),
                    NONLOC,
                    SEM) ] head 2.
```

prep_noun_rule =

```
SIGN(head_comp,
     ORDER(S0,S2),
     SYN(HEAD_FEAT,
          MINOR,
          ADJS,
          SUBCAT(C1,COMP(VAL,plus,LPR),C3,C4)),
     NONLOC,
     SEM)

          ->

     [    SIGN(_,
               ORDER(S0,S1),
               SYN(@HEAD_FEAT P_HEAD(_,_,_),
               MINOR,
               ADJS,
               SUBCAT(C1,COMP(VAL,minus,_),C3,C4)),
          NONLOC,
          SEM),

          @VAL SIGN(_,
                    ORDER(S1,S2),
                    SYN(HEAD(_,_,LPR),_,_,_),_,_) ] head 1.
```

Index