Linear Precedence Constraints in "Lean" Formalisms

February 7, 1994

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

1. INTRODUCTION

In this report a method for encoding LP constraints is presented that lends itself for a an easy integration into ALEP grammars since it does not require any changes to the formalism and to the implementation of the current ALEP development platform.

The ALEP system is the result of an effort to provide the European NLP community with a flexible state-of-art development platform that fulfills at least two important functions. Firstly, it is meant to save new projects the immense investments of funds and time involved in developing yet another such system before the actual NL research work can begin. Secondly, if taken up by a sufficient number of projects, it can be seen as a step towards the reusability of grammatical resources.

The basic design philosophy of the ALEP system is based on the assumption that some useful compromise can be offered between the expressive power demanded by the grammar writer and existing methods for efficient syntactic processing on the basis of unification grammars. The compromise rests on encouraging experiences made in exploiting the effiency of Prolog term unification for syntactic processing on the one hand and methods for transforming unification grammars into Prolog DCGs on the other.

In order to ensure flexibility and extendability a three level architecture has been designed. Level 1 is the efficient low-level Prolog grammar. Level 2 is an enriched typed feature-unification grammar that can be compiled into a level 1 format. For future extensions, a third level is foreseen that may contain additional constraint types that cannot be translated into level 1 grammars and therefore require external constraint solvers.

For expressing many cases of word order generalizations, the existing level 2 formalism already permits the formulation of standard ID/LP grammars. In these grammars, unordered phrase structure rules, so-called ID rules may be written whose right-hand sides are linearised by

LP rules. These grammars are compiled into fully ordered level 1 grammars by spelling out all the linearisations of right-hand sides permitted by the LP component into separate rules. Thus the domain of LP rules are restricted to sibling nodes.

For treating word order phenomena in many languages–among them some Germanic and all Slavic languages–this method does not suffice. Since in these languages the complements of the verb and any number of adjuncts can be interspersed and since, moreover, the order of these sequences of complements and adjuncts still have to be ordered by LP rules, the domain of LP rules needs to be extended beyond the right-hand sides of individual rules. We will summarize the most relevant arguments for such an extension in this report.

While the application of word order constraints is useful for parsing as a basis for the selection of readings of ambiguous sentences, their real importance is in the field of language generation, for example in the context of machine translation. In general, a generation component is faced with the problem of choosing among several different linearisations of a sentence. LP rules provide a way to choose among alternative linearisations. Our encoding of LP-constraints is fully declarative and independent of the order ot processing so that is can be used for both parsing and generation. The apparent asymmetry that sentences with certain orderings should be accepted by a parser, but not generated, is explained by the fact that there are LP rules which refer to the discourse functions topic and focus. In parsing, it is not known in advance which discourse function will be filled by a particular constituent, so that different orderings can be accepted. In generation, a discourse planning component will assgin discourse functions to concepts that must be generated, and the LP rules constrain the linear order on the basis of this information.

Our approach for providing the user community of ALEP with the means for writing grammars in an extended ID/LP format conforms with the ALEP design philosophy. We were able to provide an encoding for extended ID/LP grammars within level 2. The approach is based on auxiliary features for the flow of LP information through the tree that the grammar writer will not have to specify. In this report the method is described in detail and discussed with some illustrative examples. In the upcoming final delivery a sample implementation in ALEP will be provided.

In this report we will also indicate–but neither fully specify nor implement–a possible future level 3 extension. The approach builds on the observation that LP constraints with extended domains can be expressed and implemented as finite-state automata, that limit the number of linearizations. This method could be utilized at a later time when hooks for external constraint solvers will be provided. It might become relevant if grammars with extensive LP components involving the topic-focus structure and complex LP constraints would blow up the number of resulting level 1 rules beyond the limits needed for efficient processing.

2. MOTIVATION

This section presents the linguistic and practical motivation for redefining the ordering domains to which LP-statements apply. LP statements in GPSG (Gazdar et al. 1985) constrain the possibility of linearizing immediate dominance (ID) rules. By taking the right-hand sides of ID rules as their domain, they allow only the ordering of sibling constituents. Consequently, grammars must be designed in such a way that all constituents which are to be ordered by LP constraints must be dominated by one node in the tree, so that "flat" phrase structures result, as illustrated in figure 1.

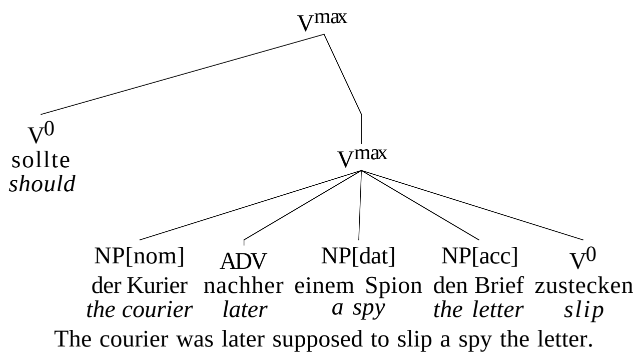


Figure 1

Uszkoreit (1986) argues that such flat structures are not well suited for the description of languages such as German and Dutch. The main reason[1] is so-called complex fronting, i.e., the fronting of a non-finite verb together with some of its complements and adjuncts as it is shown in (1). Since it is a well established fact that only one constituent can be fronted, the flat structure can account for the German examples in (1), but not for the ones in (2).

(1) sollte der Kurier nachher einem Spion den Brief zustecken
zustecken sollte der Kurier nachher einem Spion den Brief
den Brief sollte der Kurier nachher einem Spion zustecken
einem Spion sollte der Kurier nachher den Brief zustecken
nachher sollte der Kurier einem Spion den Brief zustecken
der Kurier sollte nachher einem Spion den Brief zustecken

(2) **den Brief zustecken** sollte der Kurier nachher einem Spion
einem Spion den Brief zustecken sollte der Kurier nachher

[1]Further reasons are discussed in Uszkoreit (1991b).

nachher einem Spion den Brief zustecken sollte der Kurier

In the hierarchical tree structure in figure 2, the boxed constituents can be fronted, accounting for the examples in (1) and (2).

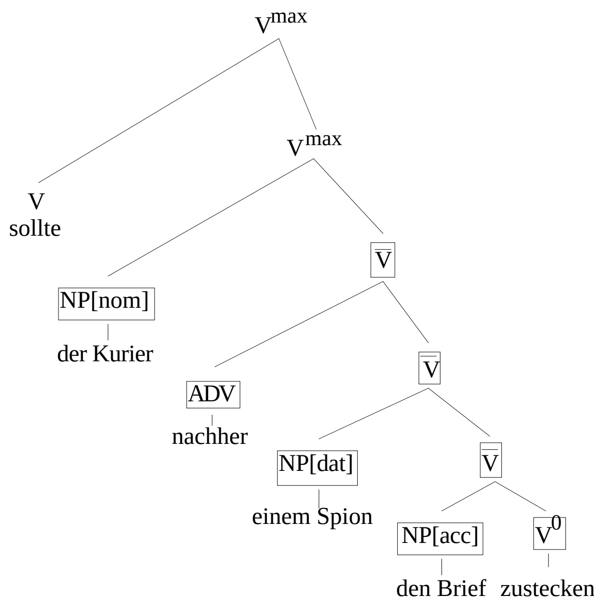


Figure 2

But with this tree structure, LP constraints can no longer be enforced over siblings. Rather, LP constraints must be applied to head domains, defined in section 3.1.2.

It must also be noted that hierarchichal, binary branching structures can arise for processing, even when it seems that the linguistic problems can be worked around by assuming flat structures. This situation occurs when the Kleene-star notatation is used in rules to provide for 0 or more repetitions of a constituent. When this notation is translated to ALEP rules, new non-terminal symbols are introduced in order to handle the repetition. This notation is used, for example in the following phrase structure rule to allow adjuncts (indicated by the nonterminal A) to be interspersed with complements.

S -> A* NP[subj] A* NP[obj] A*

However, it is necessary to enforce ordering constraints among the adjuncts, according to the thematic role that they fill. In German and many other languages, an adjunct associated with the source role is ordered before the goal, as in sentence (3), where the source *von Berlin* must come before the goal *nach München.*

(3) ...hat das Direktorium gestern von Berlin einen Boten nach München geschickt.

has the directorate yesterday from Berlin a messenger to Munich sent.

Ordering constraints are important in parsing ambiguous sentences, where the constraints can help to make a choice between different readings, and even more important in generation (e.g. in a machine-translation context) where it is important to choose the right linearization in order to facilitate understanding of the generated utterance. For example in sentence like (4) where the ordering constraints are violated, the phrase *von Berlin* would naturally be interpreted as an adjunct of *Boten* and not of the verb *geschickt.*

(4) ... hat das Direktorium gestern nach München einen Boten von Berlin geschickt.

has the directorate yesterday to Munich a messenger from Berlin sent.

3. WORD ORDER IN GRAMMATICAL DESCRIPTIONS OF NATURAL LANGUAGES

Given the working assumption that word order reguarities must be stated in a grammatical description and do not follow automatically from the processing model (as in the theory of early immediate constituents (Hawkins 1992)), any grammatical description of word order[2] regularities must fix three parameters:

1. The domain in which the word order constraint should hold, and
2. The language for stating ordering constraints,
3. The criteria for ordering constituents.

We will discuss these three aspects in this section.

3.1. WORD ORDER DOMAINS

3.1.1. Local Trees

Older theories of grammar like GPSG take sibling nodes in a phrase structure tree as the ordering domain. This forces linguistic analyses in which all elements to which ordering

[2]More precisely, one should speak of "constituent order," but we will continue to use the term "word order" which is established usage in theoretical and descriptive linguistics.

principles apply are siblings. As we have shown in the preceding section, flat structures are not adequate to handle German ordering and fronting phenomena.

Initial trees constitute the ordering domain in ID/LP TAGS (Joshi 1987). However, it is also often necessary to state ordering constraints on constituents that are not together in one initial tree, for example on adjuncts that are added by the adjunction operation.

3.1.2. Head Domains

In order to handle the data discussed in section 2, we introduce head domains as the new domain for linear order. A head domain is defined as follows:

> A **head domain** consists of the lexical head of a phrase, and its complements and adjuncts.

LP constraints must be respected within a head domain. Two alternative methods for handling LP constraints in head domains are presented in section 4.

3.1.3. Recursively Defined Domains

Reape (1989,1990) proposes a different strategy for treating partially free word order. His approach also permits the application of LP constraints across local trees. This is achieved by separating word order variation from the problem of building a semantically motivated phrase structure. Permutation across constituents can be described by merging the fringes (terminal yields) of the constituents using the operation of sequence union. All orderings imposed on the two merged fringes by LP constraints are preserved in the merged fringe.

Reape treats clause union and scrambling as permutation that does not affect constituent structure. Although we are intrigued by the elegance and descriptive power of Reape's approach, we keep our bets with our more conservative proposal. The main problem we see with Reape's strategy is the additional burden for the LP component of the grammar. For every single constituent that is scrambled out of some clause into a higher clause, the two clauses need to be sequence-unioned. A new type of LP constraints that refer to the position of the constituents in the phrase or dependency structure is employed for ensuring that the two clauses are not completely interleaved.

3.1.4. Conclusion on Word Order Domains

Local trees as the ordering domain can be adequate if flat structures are used in linguistic analyses. However, for a language like German, we have shown in section 2 that hierarchical structures are to be preferred in linguistic analyses. The method we propose for handling word order constraints in head domains can also be applied to local trees (cf. section 4.1.3).

In current LFG (Kaplan & Zaenen 1988), functional precedence rules apply to functional domains. The approach rests on a mapping from f-structure to c-structure in the projection architecture of LFG, and cannot be readily transferred to a "lean" formalism such as ALEP.

The word order domains of Reape provide very elegant analyses, but are not usable with a system like ALEP which uses concatenation as the only operation for combining strings. Adequate parsing methods for grammars using word order domains and sequence union are not well-established, but an area of current reseach, cf. (Reape 1991, van Noord 93, Bouma and van Noord 93).

In this report, we concentrate on head domains because they allow adequate linguistic analyses, and are usable with grammatical formalisms which use only concatenation as an operation for combining constituents.

3.2. The LANGUAGE FOR STATING ORDERING CONSTRAINTS

3.2.1. LP-Rules

An **LP-rule** is an ordered pair <A,B> of category descriptions, such that whenever a node a subsumed by A and a node b subsumed by B occur within the domain of an LP-rule (in the case of GPSG a local tree, in our case a head domain), a precedes b.

An LP rule <A,B> is conventionally written as A < B. It follows from the definition that B can never precede A in an LP domain. In the next section, we will show how this property is exploited in our encoding of LP constraints.

Any set of LP constraints can be translated to a regular language systematically, as will be shown in section 4.2.

3.2.2. Regular Languages

Regular languages are more expressive than LP rules. For example, the regular language A*B*A* cannot be expressed as an LP rule because in the ordering domain described by this language A can both precede and follow B.

3.2.3. Conclusion on Language for Ordering Constraints

We will provide solution for the more restricted form, LP-rules, and for the more general regular languages.

3.3. ORDERING CRITERIA

The following is a list of ordering criteria that have been proposed in the linguistic literature and used in computational grammars. Word order constraints make reference to

- major grammatical category (noun, verb etc.)
- case (nominative, accusative etc.)
- discourse function (topic/focus)
- grammatical relations (head, subject, object or relative obliqueness, government)
- thematic relations (agent, patient, source, goal etc.)
- pronominal vs. non-pronominal forms
- phonological heavyness

and others.

It is beyond the scope of this report to consider the adequacy of these different criteria. However, it is important to note that all of these criteria (with the exception of phonological heavyness, which is a fuzzy concept), refer to the values of features which can have only a finite (and small) set of values. Both of the implementations we propose in this report depend on the fact that the relevant word order criteria are a small and finite set, which can be represented by either a set of binary-valued features or in the terminal vocabulary of a regular language.

4. COMPUTATIONAL TREATMENT OF LP CONSTRAINTS

In this section, we give implementations of LP constraints in head domains by two methods:

1. The first method (section 4.1) works by keeping track of the LP-relevant features that an head domain contains and of left and right context restrictions of constituents. This method is applicable for LP rules. Our presentation of the method in this report is largely based on the paper by Engelkamp et al. (1992), in which the method was first described. We show how the method is applicable in the ALEP formalism.

2. The second method (section 4.2) is a novel approach which makes use of finite-state automata, and is usable for both LP rules and regular languages. This method has not yet been described elsewhere.

4.1. ENCODING BY OF LP-CONSTRAINTS BY LEFT AND RIGHT CONTEXT RESTRICTIONS

From a formal point of view, we want to encode LP constraints in such a way that

• violation of an LP constraint results in unification failure, and

- LP constraints, which operate on head domains, can be enforced in local trees by checking sibling nodes.

The last condition can be ensured if every node in a projection carries information about which constituents are contained in its head domain.

An LP constraint A < B implies that it can never be the case that B precedes A. We make use of this fact by the following additions to the grammar:

- Every category A carries the information that B must not occur to its left.
- Every category B carries the information A must not occur to its right.

This duplication of encoding is necessary because only the complements/adjuncts check whether the projection with which they are combined contains something that is incompatible with the LP constraints. A projection contains only information about which constituents are contained in its head domain, but no restrictions on its left and right context[3].

In the following example, we assume the LP-rules A<B and B<C. The lexical head of the tree is X^0, and the projections are $\overline{X}$, and X^{max}. The complements are A, B and C. Each projection contains information about the constituents contained in it, and each complement contains information about what must not occur to its left and right. A complement is only combined with a projection if the projection does not contain any category that the complement prohibits on its right or left, depending on which side the projection is added.

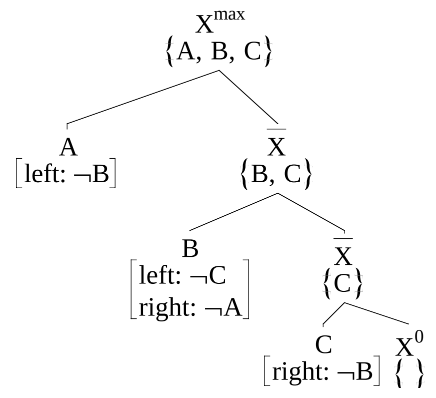


Figure 3

Having now roughly sketched our approach, we will turn to the questions of how a violation of LP constraints results in unification failure, how the information associated with the

[3]Alternatively, the projections of the head could as well accumulate the ordering restrictions while the arguments and adjuncts only carry information about their own LP-relevant features. The choice between the alternatives has no linguistic implications since it only affects the grammar compiled for processing and not the one written by the linguist.

projections is built up, and what to do if LP constraints operate on feature structures rather than on atomic categories.

4.1.1. Violation of LP-constraints as unification failure

As a conceptual starting point, we take a number of LP constraints. For the expository purposes of this paper, we oversimplifiy and assume just the following four LP constraints:

nom < dat	(nominative case precedes dative case)
nom < acc	(nominative case precedes accusative case)
dat < acc	(dative case precedes accusative case)
pro < nonpro	(pronominal NPs precede non-pronominal NPs)

Figure 4

Note that nom, dat, acc, pro and nonpro are not syntactic categories, but rather values of syntactic features. A constituent, for example the pronoun *ihn* (him) may be both pronominal and in the accusative case. For each of the above values, we introduce an extra boolean feature, as illustrated in figure 5.

$$\begin{bmatrix} \text{NOM } \textit{bool} \\ \text{DAT } \textit{bool} \\ \text{ACC } \textit{bool} \\ \text{PRO } \textit{bool} \\ \text{NON-PRO } \textit{bool} \end{bmatrix}$$

Figure 5

Arguments encode in their feature structures what must not occur to their left and right sides. The dative NP *einem Spion* (a spy), for example, must not have any accusative constituent to its left, and no nominative or pronominal constituent to its right, as encoded in the following feature structure. The feature structures that constrain the left and right contexts of arguments only use '-' as a value for the LP-relevant features.

$$\begin{bmatrix} \text{LEFT} \begin{bmatrix} \text{ACC -} \end{bmatrix} \\ \text{RIGHT} \begin{bmatrix} \text{NOM -} \\ \text{PRO -} \end{bmatrix} \end{bmatrix}$$

Figure 6: Feature Structure for *einem Spion*

Lexical heads, and projections of the head contain a feature LP-STORE, which carries information about the LP-relevant information occuring within their head domain (figure 7).

$$\left[\begin{array}{ll} \text{LP-STORE} & \left[\begin{array}{l} \text{NOM} - \\ \text{DAT} - \\ \text{ACC} - \\ \text{PRO} - \\ \text{NON-PRO} - \end{array}\right] \end{array}\right]$$

Figure 7: empty LP-STORE

In our example, where the verbal lexical head is not affected by any LP constraints, the LP-STORE contains the information that no LP-relevant features are present.

For a projection like *einen Brief zusteckt* (a letter[acc] slips), we get the following LP-STORE.

$$\left[\begin{array}{ll} \text{LP-STORE} & \left[\begin{array}{l} \text{NOM} - \\ \text{DAT} - \\ \text{ACC} + \\ \text{PRO} - \\ \text{NON-PRO} + \end{array}\right] \end{array}\right]$$

Figure 8: LP-STORE of *einen Brief zusteckt*

The NP *einem Spion* (figure 6) can be combined with the projection *einen Brief zusteckt* (figure 8) to form the projection *einem Spion einen Brief zusteckt* (a spy[dat] a letter[acc] slips) because the RIGHT feature of *einem Spion* and the LP-STORE of *einen Brief zusteckt* do not contain incompatible information, i.e., they can be unified. This is how violations of LP constraints are checked by unification. The projection *einem Spion einen Brief zusteckt* has the following LP-STORE.

$$\left[\begin{array}{ll} \text{LP-STORE} & \left[\begin{array}{l} \text{NOM} - \\ \text{DAT} + \\ \text{ACC} + \\ \text{PRO} - \\ \text{NON-PRO} + \end{array}\right] \end{array}\right]$$

Figure 9: LP-STORE of *einem Spion einen Brief zusteckt*

The constituent *ihn zusteckt* (figure 10) could not be combined with the non-pronominal NP *einem Spion* (figure 6).

$$\left[\begin{array}{ll} \text{LP-STORE} & \left[\begin{array}{l} \text{NOM} - \\ \text{DAT} - \\ \text{ACC} + \\ \text{PRO} + \\ \text{NON-PRO} - \end{array}\right] \end{array}\right]$$

Figure 10: LP-STORE of *ihn zusteckt*

In this case, the value of the RIGHT feature of the argument *einem Spion* is not unifiable with the LP-STORE of the head projection *ihn zusteckt* because the feature PRO has two different atoms (+ and -) as its value. This is an example of a violation of an LP constraint leading to unification failure.

In the next section, we show how LP-STOREs are manipulated.

4.1.2. Manipulation of the LP-STORE

Since information about constituents is <u>added</u> to the LP-STORE, it would be tempting to add this information by unification, and to leave the initial LP-STORE unspecified for all features. This is not possible because violation of LP constraints is also checked by unification. In the process of this unification, values for features are added that may lead to unwanted unification failure when information about a constituent is added higher up in the tree.

Instead, the relation between the LP-STORE of a projection and the LP-STORE of its mother node is encoded in the argument that is added to the projection. In this way, the argument "changes" the LP-STORE by "adding information about itself". Arguments there-fore have the additional features LP-IN and LP-OUT. When an argument is combined with a projection, the projection's LP-STORE is unified with the argument's LP-IN, and the argument's LP-OUT is the mother node's LP-STORE. The relation between LP-IN and LP-OUT is specified in the feature structure of the argument, as illustrated in figure 11 for the accusative pronoun *ihn*, which is responsible for changing figure 7 into figure 10. No matter what the value for the features ACC and PRO may be in the projection that the argument combines with, it is '+' for both features in the mother node. All other features are left unchanged[4].

$$\left[\begin{array}{ll} \text{LP-IN} & \left[\begin{array}{l} \text{NOM } \boxed{1} \\ \text{DAT } \boxed{2} \\ \text{ACC [ ]} \\ \text{PRO [ ]} \\ \text{NON-PRO } \boxed{3} \end{array}\right] \\ \text{LP-OUT} & \left[\begin{array}{l} \text{NOM } \boxed{1} \\ \text{DAT } \boxed{2} \\ \text{ACC } + \\ \text{PRO } + \\ \text{NON-PRO } \boxed{3} \end{array}\right] \end{array}\right]$$

Figure 11

Note that only a '+' is added as value for LP-relevant features in LP-OUT, never a '-'. In this way, only positive information is accumulated, while negative information is "removed". Positive information is never "removed".

Even though an argument or adjunct constituent may have an LP-STORE, resulting from LP constraints that are relevant within the constituent, it is ignored when the constituent becomes argument or adjunct to some head. Our encoding ensures that LP constraints apply to all head domains in a given sentence, but not across head domains.

[4]Coreference variables are indicated by boxed numbers. [ ] is the feature structure that contains no information (TOP) and can be unified with any other feature structure.

It still remains to be explained how complex phrases that become arguments receive their LP-IN, LP-OUT, RIGHT and LEFT features. These are specified in the lexical entry of the head of the phrase, but they are ignored until the maximal projection of the head becomes argument or adjunct to some other head. They must, however, be passed on unchanged from the lexical head to its maximal projection. When the maximal projection becomes an argument/adjunct, they are used to check LP constrains and "change" the LP-STORE of the head's projection.

Our method also allows for the description of head-initial and head-final constructions. In German, for example, we find prepositions (e.g. *für*), postpositions (e.g. *halber*) and some words that can be both pre- and postpostions (e.g. *wegen*).

The LP-rules would state that a postposition follows everything else, and that a preposition precedes everything else.

$$[\text{PRE} +] < [\,]$$

$$[\,] < [\text{POST} +]$$

Figure 12

The information about whether something is a preposition or a postposition is encoded in the lexical entry of the preposition or postposition. In the following figure, the LP-STORE of the lexical head contains also positive values.

$$\left[\text{LP-STORE}\begin{bmatrix}\text{POST} & + \\ \text{PRE} & -\end{bmatrix}\right]$$

Figure 13: part of the lexical entry of a postposition

$$\left[\text{LP-STORE}\begin{bmatrix}\text{POST} & - \\ \text{PRE} & +\end{bmatrix}\right]$$

Figure 14: part of the lexical entry of a preposition

A word that can be both a preposition and a postposition is given a disjunction of the two lexical entries:

$$\left[\text{LP-STORE}\left\{\begin{matrix}\begin{bmatrix}\text{POST} & - \\ \text{PRE} & +\end{bmatrix} \\ \begin{bmatrix}\text{POST} & + \\ \text{PRE} & -\end{bmatrix}\end{matrix}\right\}\right]$$

Figure 15

All complements and adjuncts encode the fact that there must be no preposition to their right, and no postposition to their left.

$$\begin{bmatrix}\text{RIGHT}\,[\text{PRE} -] \\ \text{LEFT}\,[\text{POST} -]\end{bmatrix}$$

Figure 16

The manipulation of the LP-STORE by the features LP-IN and LP-OUT works as usual.

The above example illustrates that our method of encoding LP constraints works not only for verbal domains, but for any projection of a lexical head. The order of quantifiers and adjectives in a noun phrase can be described by LP constraints.

4.1.3. Application to local trees

It should be noted that the encoding of LP-constraints with right (or left) context restrictions is also applicable to local trees. In this case, there is no need for the feature LP-store associated with intermediate projections in a head-domain. Instead, the LP-in feature of a constituent can be immediately unified with the LP-out feature with the constituent to its right, and the right context restriction of a constituent can be matched against the LP-out value of the constituent to its right.

```
X -> A B C
A right = A lp-in = A lp-out
B right = B lp-in = C lp-out
```

Or more generally for local trees with n daughters, generated by a rule of the form X -> R_1 ... R_n, one needs equations for each i = 1 ... n-1.

R_i right = R_i lp-in = R_{i+1} lp-out

4.1.4. Integration into HPSG

In this section, our encoding of LP constraints is incorporated into HPSG (Pollard & Sag 1987). We deviate from the standard HPSG grammar in the following respects:

- The features mentioned above for the encoding of LP-constraints are added.
- Only binary branching grammar rules are used.
- Two new principles for handling LP-constraints are added to the grammar.

Further we shall assume a set-valued SUBCAT feature as introduced by Pollard (1990) for the description of German. Using sets instead of lists as the values of SUBCAT ensures that the order of the complements is only constrained by LP-statements.

In the following figure, the attributes needed for the handling of LP-constraints are assigned their place in the HPSG feature system.

$$\begin{bmatrix} \text{SYNSEM} \mid \text{LOC} \begin{bmatrix} \text{HEAD} \begin{bmatrix} \text{LP-IN}\,[\;] \\ \text{LP-OUT}\,[\;] \\ \text{LEFT}\,[\;] \\ \text{RIGHT}\,[\;] \end{bmatrix} \\ \text{LP-STORE}\,[\;] \end{bmatrix} \end{bmatrix}$$

Figure 17

The paths SYNSEM|LOC|HEAD|{LP-IN,LP-OUT,RIGHT,LEFT} contain information that is relevant when the constituents becomes an argument/adjunct. They are HEAD features so that they can be specified in the lexical head of the constituent and are percolated via the Head Feature Principle to the maximal projection. The path SYNSEM|LOC|LP-STORE contains information about LP-relevant features contained in the projection dominated by the node described by the feature structure. LP-STORE can obviously not be a head feature because it is "changed" when an argument or adjunct is added to the projection.

In figures 18 and 19, the principles that enforce LP-constraints are given[5]. Depending on whether the head is to the right or to the left of the comple-ment/adjunct, two versions of the principle are dis-tinguished. This distinction is necessary because linear order is crucial. Note that neither the HEAD features of the head are used in checking LP constraints, nor the LP-STORE of the complement or adjunct.

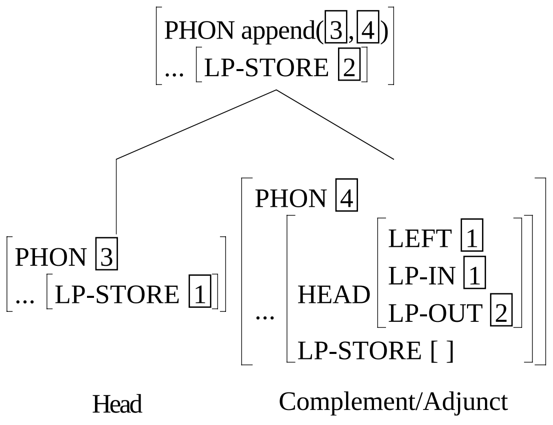


Figure 18: Left-Head LP-Principle

[5]The dots (...) abbreviate the path SYNSEM|LOCAL

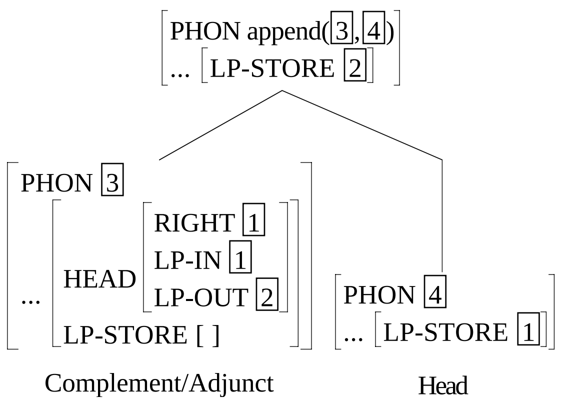


Figure 19: Right-Head LP-Principle

In the following examples, we make use of the parametrized type notation used in the grammar formalism STUF (Dörre 1991). A parametrized type has one or more parameters instantiated with feature structures. The name of the type (with its parameters) is given to the left of the := sign, the feature structure to the right.

In the following we define the parametrized types nom(X,Y), dat(X,Y), pro(X,Y), and non-pro(X,Y), where X is the incoming LP-STORE and Y is the outgoing LP-STORE.

$$\mathrm{nom}\left(\begin{bmatrix}\text{NOM [ ]}\\ \text{DAT } \boxed{1}\\ \text{ACC } \boxed{2}\\ \text{PRO } \boxed{3}\\ \text{NON-PRO } \boxed{4}\end{bmatrix}, \begin{bmatrix}\text{NOM +}\\ \text{DAT } \boxed{1}\\ \text{ACC } \boxed{2}\\ \text{PRO } \boxed{3}\\ \text{NON-PRO } \boxed{4}\end{bmatrix}\right) :=$$

$$\begin{bmatrix}\text{SYNSEM|LOC} & \begin{bmatrix}\text{HEAD} & \begin{bmatrix}\text{CASE nom}\\ \text{LEFT} \begin{bmatrix}\text{DAT -}\\ \text{ACC -}\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Figure 20

$$\mathrm{dat}\left(\begin{bmatrix}\text{NOM } \boxed{1}\\ \text{DAT [ ]}\\ \text{ACC } \boxed{2}\\ \text{PRO } \boxed{3}\\ \text{NON-PRO } \boxed{4}\end{bmatrix}, \begin{bmatrix}\text{NOM } \boxed{1}\\ \text{DAT +}\\ \text{ACC } \boxed{2}\\ \text{PRO } \boxed{3}\\ \text{NON-PRO } \boxed{4}\end{bmatrix}\right) :=$$

$$\begin{bmatrix}\text{SYNSEM|LOC} & \begin{bmatrix}\text{HEAD} & \begin{bmatrix}\text{CASE dat}\\ \text{LEFT | ACC -}\\ \text{RIGHT | NOM -}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Figure 21

$$\text{pro}\left(\begin{bmatrix}\text{NOM }\boxed{1}\\ \text{DAT }\boxed{2}\\ \text{ACC }\boxed{3}\\ \text{PRO [ ]}\\ \text{NON-PRO }\boxed{4}\end{bmatrix},\begin{bmatrix}\text{NOM }\boxed{1}\\ \text{DAT }\boxed{2}\\ \text{ACC }\boxed{3}\\ \text{PRO +}\\ \text{NON-PRO }\boxed{4}\end{bmatrix}\right):=$$

$$\left[\text{SYNSEM|LOC}\left[\text{HEAD}\left[\text{LEFT }|\text{ NON-PRO -}\right]\right]\right]$$

Figure 22

$$\text{non-pro}\left(\begin{bmatrix}\text{NOM }\boxed{1}\\ \text{DAT }\boxed{2}\\ \text{ACC }\boxed{3}\\ \text{PRO }\boxed{4}\\ \text{NON-PRO [ ]}\end{bmatrix},\begin{bmatrix}\text{NOM }\boxed{1}\\ \text{DAT }\boxed{2}\\ \text{ACC }\boxed{3}\\ \text{PRO }\boxed{4}\\ \text{NON-PRO +}\end{bmatrix}\right):=$$

$$\left[\text{SYNSEM|LOC}\left[\text{HEAD}\left[\text{RIGHT }|\text{ PRO -}\right]\right]\right]$$

Figure 23

The above type definitions can be used in the definition of lexical entries. Since the word *ihm*, whose lexical entry[6] is given in figure 24, is both dative case and pronominal, it must contain both types. While the restrictions on the left and right context invoked by dat/2 and pro/2 can be unified[7], matters are not that simple for the LP-IN and LP-OUT features. Since their purpose is to "change" rather than to "add" information, simple unification is not possible. Instead, LP-IN of *ihm* becomes the in-coming LP-STORE of dat/2, the outgoing LP-STORE of dat/2 becomes the incoming LP-STORE of pro/2, and the outgoing LP-STORE of pro/2 becomes LP-OUT of *ihm*, such that the effect of both changes is accumulated.

$$\text{ihm} :=$$
$$\left[\text{SYNSEM|LOC}\left[\text{HEAD}\begin{bmatrix}\text{LP-IN }\boxed{1}\\ \text{LP-OUT }\boxed{3}\end{bmatrix}\right]\right]\wedge$$
$$\text{dat}(\boxed{1},\boxed{2})\wedge\text{pro}(\boxed{2},\boxed{3})$$

Figure 24: lexical entry for *ihm*

After expansion of the types, the following feature structure results. Exactly the same feature structure had been resulted if dat/2 and pro/2 would have been exchanged in the above lexical entry ($\text{pro}(\boxed{1},\boxed{2})\wedge\text{dat}(\boxed{2},\boxed{3})$), because the effect of both is to instantiate a '+' in LP-OUT.

[6]Only the information which is relevant for the processing of LP constraints is included in this lexical entry.

[7]dat/2 means the type dat with two parameters.

$$\left[\text{SYNSEM|LOC}\ \left[\text{HEAD}\ \left[\begin{array}{ll} \text{LP-IN} & \left[\begin{array}{l}\text{NOM } \boxed{1}\\ \text{DAT } [\,]\\ \text{ACC } \boxed{2}\\ \text{PRO } [\,]\\ \text{NON-PRO } \boxed{3}\end{array}\right]\\ \text{LP-OUT} & \left[\begin{array}{l}\text{NOM } \boxed{1}\\ \text{DAT } +\\ \text{ACC } \boxed{2}\\ \text{PRO } +\\ \text{NON-PRO } \boxed{3}\end{array}\right]\\ \text{LEFT} & \left[\begin{array}{l}\text{ACC -}\\ \text{NON-PRO -}\end{array}\right]\\ \text{RIGHT} & [\text{NOM -}]\\ \text{CASE} & \text{dat}\end{array}\right]\right]\right]$$

Figure 25: expanded lexical entry for *ihm*

The next figure shows the lexical entry for a non-pronominal NP, with a disjunction of three cases.

$$\begin{array}{l}\text{Peter} := \\ \left[\text{SYNSEM|LOC}\ \left[\text{HEAD}\ \left[\begin{array}{l}\text{LP-IN } \boxed{1}\\ \text{LP-OUT } \boxed{3}\end{array}\right]\right]\right] \wedge \\ \big(\text{nom}(\boxed{1},\boxed{2}) \vee \text{dat}(\boxed{1},\boxed{2}) \vee \text{acc}(\boxed{1},\boxed{2})\big) \wedge \text{non-pro}(\boxed{2},\boxed{3})\end{array}$$

Figure 26

4.1.5. Application to ALEP

In this section, we illustrate that the technique outlined above is applicable to the ALEP1 formalism by providing example declarations, rules and lexical entries.

In the next delivery, the use of these techniques in an ALEP grammar for a larger fragment of German will be described.

Type and Feature Declarations:

In addition to the type definitions employed in a grammatical description, the type `lp_store` is needed which introduces a set of binary-valued features used to encode the set of features that can appear on the left- and right-hand sides of LP rules.

```
type(lp_store:{
          nom => atom({1,0}),
          dat => atom({1,0}),
          acc => atom({1,0}),
          pro => atom({1,0}),
          non_pro => atom({1,0})}
```

```
        'An lp_store has the features nom, dat, acc, pro, non_pro').
```

The features for handling LP constraints (`lp_in, lp_out, left, right`) must be added to the type of the feature value which is percolated in head domains. In our example, this is the type `hd,` which is used for head features.

```
type(hd:{
        lp_in => type({lp_store:{}}),
        lp_out => type({lp_store:{}}),
        left => type({lp_store:{}}),
        right => type({lp_store:{}}),
        cat => type({syncat:{}})}
        'The type hd has the features lp_in, lp_out, left, right
         for handling LP constraints and the feature cat for the
         usual head features').
```

Rules:

The rules of the grammar must be extended with the necessary features and coreferences for handling LP constraints. Instead of giving all the necessary rules, we provide here two rule schemata that can be used together with binary branching rules. In order to re-use these rule schemata in other rules, we define them as macros.

```
% Mother -> Head Complement/Adjunct
macro(left_head_rule,
sign:{ lp_store => LP_STORE }
<
[ sign:{ head => hd:{ lp_store => HEAD_LP_STORE }
       }
  sign:{ head => hd:{ lp_in => HEAD_LP_STORE,
                     lp_out => LP_STORE,
                     left => HEAD_LP_STORE }
       }
]
).

% Mother -> Complement/Adjunct Head
macro(right_head_rule,
sign:{ lp_store => LP_STORE }
<
[ sign:{ head => hd:{ lp_in => HEAD_LP_STORE,
                     lp_out => LP_STORE,
                     right => HEAD_LP_STORE }
       }
  sign:{ head => hd:{ lp_store => HEAD_LP_STORE }
       }
]
).
```

These macros can then be used in concrete grammar rules. The first macro "left_head_rule" can be part of the specification of any rule where the left daughter is the head and the right daughter a complement or adjunct. The second macro "right_head_rule" can be part of the specification of rules where the head is the right constituent.[8]

Lexical entries:

The following lexical entry for the dative pronoun *ihm* contains the specifcations of the features `lp_in` and `lp_out` which are responsible for "adding" the elements "dative" and "pronominal" to the LP-store, and the left and right context restrictions.

```
ihm ~
```

[8]From BNF given in the ALEP1 documentation, it is not clear that a macro for rules can be called in the definition of a rule.

```
sign:{ synsem => synsem_type:{local => local_type:{
       head => hd:{lp_in => lp_store:{ nom => NOM,
                                      dat => _,
                                      acc => ACC,
                                      pro => _,
                                      non_pro => NONPRO
                                     },
                   lp_out => lp_store:{ nom => NOM,
                                      dat => 1,
                                      acc => ACC,
                                      pro => 1,
                                      non_pro => NONPRO
                                     },
                   left   => lp_store:{ acc => 0,
                                      non_pro => 0
                                    },
                   right  => lp_store:{ nom => 0
                                     }
                  }
        }}}
```

By making use of the macros provided by the ALEP1 formalism, this lexical entry can be made more compact, similar to figures 21, 22, and 24.

An ALEP grammar for a fragment of German with LP rules will be part of the next deliverable, along with an investigation of the possiblities to compile a set of LP constraints to the encoding provided here.

4.1.6. Conclusion on encoding by sets

We have presented a formal method for the treatment of LP constraints, which requires no addition to standard feature unification formalisms. It should be emphasized that our encoding only affects the compiled grammar used for the processing. The linguist does not lose any of the descriptive means nor the conceptual clarity that an ID/LP formalism offers. Yet he gains an adequate computational interpretation of LP constraints.

Because of the declarative specification of LP constraints, this encoding is neutral with respect to processing direction (parsing-generation). It does not depend on specific strategies (top-down vs. bottom-up) although, as usual, some combinations are more efficient than others. This is an advantage over the formalization of unification ID/LP grammars in Seiffert (1991) and the approach by Erbach (1991). Seiffert's approach, in which LP constraints operate over siblings, requires an addition to the parsing algorithm, by which LP constraints are checked during processing to detect violations as early as possible, and again after processing, in case LP-relevant infor-mation has been added later by unification. Erbach's approach can handle LP constraints in head domains by building up a list of constituents over which the LP constraints are enforced, but also requires an addition to the parsing algorithm for checking LP constraints during as well as after processing.

Our encoding of LP constraints does not require any particular format of the grammar, such as left- or right-branching structures. Therefore it can be incorporated into a variety of linguistic analyses. There is no need to work out the formal semantics of LP constraints because feature unification formalisms already have a well-defined formal semantics.

It must be noted that the use of LP-rules requires an underlying grammatical description which allows free ordering of constituents, which is constrained by the LP-rules. Such free constituent order cannot be achieved with fixed subcat lists and a grammatical formalism which uses concatenation as the only operation for combining constituents, and has no procedural attachment for operations on the subcat list (e.g. deletion of an element of the list which is not the first one). Various options for encoding subcategorization in such a way that any element can be accessed are discussed in section 5 of Deliverable B of the project LRE-61-061 "The Reusability of Grammatical Resources" [Erbach et al. 1994].

4.2. LINEAR PRECEDENCE CONSTRAINTS AS FINITE-STATE AUTOMATA

The Basic Idea

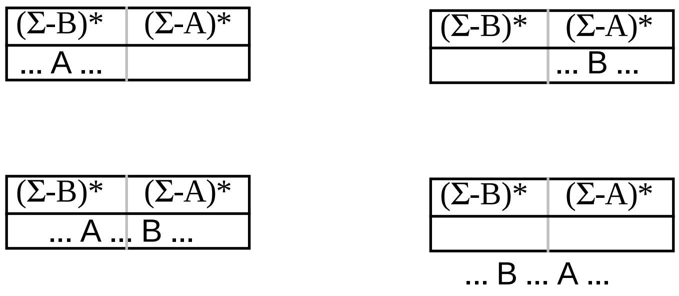
The use of regular expressions for the encoding of linear precedence statements historically precedes the introduction of binary LP rules in the so-called ID/LP format (Kay 1985). However the utilization of the full power of regular expressions exceeds the use of binary LP rules in expressivity. Although the weak generative capacity of the CF grammar is neither influenced by binary LP rules nor by LP rules written as free-form regular expressions, the latter allow for the derivation of PS representations that the former cannot produce.

Assume, e.g., the following regular expression to be used in the LP component of an ID/LP grammar: $A\ B\ A^*\ B^*$. The hypothetical rule may be proposed to enforce the so-called Wackernagel position. If there are any rules in the grammar that place more than one A or more than B in the domain of locality for the LP rules–traditionally the domain of sibling nodes–the application of the LP rule will result in a linearization that could not be enforced with binary LP rules. Binary LP rules could either order all As before all Bs, order all Bs before all As, or permit all permutations of As and Bs.

Unconstrained regular expressions are still used for the encoding of LP statements in implementations of LFG. However, binary LP rules have neither been introduced nor implemented as regular expressions or their corresponding FSAs although the correspondence has been noted (ref).

For our proposed encoding of LP constraints as finite-state machines, we exploit the fact that LP constraints can be written as very simple regular expressions, thus they denote regular languages. The LP constraint A<B stands for the simple regular expression $(\overline{B})^* (\overline{A})^*$ which is equivalent to $\overline{(\Sigma^* \ B \ \Sigma^* \ A \ \Sigma^*)}$. Any sequence of category symbols not containing category B may be followed by any sequence of category symbols not containing A in the domain of the constraint.

The equivalence between the LP constraint and the regular expression $(\overline{B})^* (\overline{A})^*$ is obvious. If the string does neither contain A nor B then it matches either one of the two subexpressions $(\overline{B})^*$ and $(\overline{A})^*$. If there are As or Bs in the string, one of the following four cases applies: … A … , … B … , … A … B … , … B … A …. The first three cases match the regular expression, only for the last one no match can be found.



A simple automaton for the regular expression is given below:

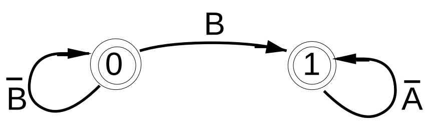


Figure 27

The automata encoding single LP constraints may be combined by intersection. The automaton corresponding to A<B, for instance, may be intersected with the one corresponding to B<C:

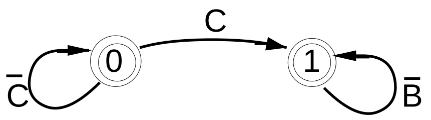


Figure 28

The minimized result of the intersection is the following FSA:

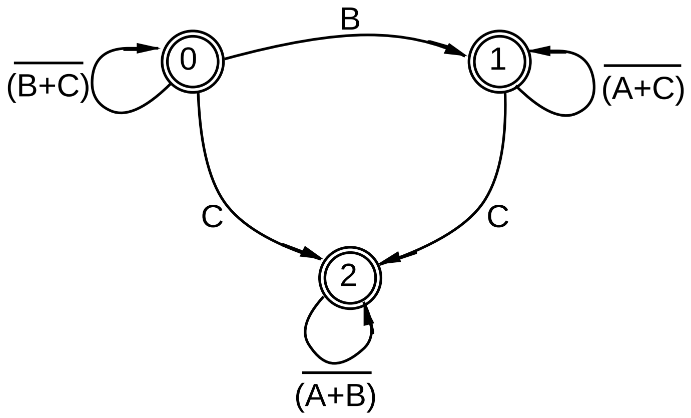


Figure 29

For implementation purposes, we might as well have decided to run the two LP automata in parallel. We will return to the question of intersected vs. parallel automata later. For the time being we will assume the intersection approach.

We call the finite state machine that encodes all LP constraints of a grammar by intersecting them a LP-FSA.

The LP-FSA is an instance of a deterministic finite state machine. Thus it is defined as a 5-tuple (Q,Σ, δ, q_o, F) where Q is the set of states, Σ is the input alphabet, δ: $Q \times \Sigma \rightarrow Q$ is the transition function, q_o is the initial state, and F is the set of final states. For the LP-FSA, Σ is the union of V_T and V_N, for an HPSG Σ would have to be ext(restr(*SIGN*)), where restr is some quotient yielding a finite LP-relevant portion of *SIGN* along the lines of the restrictor approach to Earley-parsing with a unification grammar (Shieber 1985), i.e., all LP-relevant portions of all feature terms that are in the extension of the sort *SIGN.*

The LP-FSA is a filter for each sequence of signs in a word order domain. If the domain is a sequence of sibling daughters, the automaton could be used to test the compliance with the LP constraints during left-to-right parsing. Each state in the automaton stands for a set of categories that must not occur in the remaining portion of the input string. In the worst case the number of states is equal to the cardinality of the powerset of LP-relevant categories, i.e., categories that occur in binary LP rules. However, if categories occur more than once in LP rules, the number can be smaller as in our little example (29).

The automata encoding turns the exponential time problem of normal ID/LP parsing[9] (depending on the size of the grammar) that was noted by Barton (1985) into an exponential space problem. The LP-FSA can also be applied to word order domains whose elements are not sibling in the phrase structure such as the head-domains proposed in section X. We will next show how processing with such LP-FSAs can easily be integrated into normal bottom-up syntactic processing.

For the time being, we will assume that we work with a language that is strictly head-initial, i.e., a language in which the head always precedes its complements and adjuncts. Later we will discuss straightforward extensions to languages with other head positions (head-final and head-medial) and to top-down parsing and generation.

In the case of a head-initial language, we only have to take care of the order in which the syntactic arguments and adjuncts of each head occur.

We assume binary branching structures in which any nonterminal node combines a head with an argument or adjunct. Each node in the tree carries a state symbol. We will call this symbol the lp-state of the node. In processing we will only look at the state symbols of the mother and head nodes.

A binary subtree (M (H A)) will only be accepted if δ(lp-state(H), A) = lp-state(M). Thus the syntactic argument or adjunct serves as the input symbol that triggers the transition from the lp-state of the head to the lp-state of the mother. The lexical head starts out with the initial state of the LP-FSA.

An Example

Assume a language with the two LP constraints A<B and B<C so that we can use the automaton given in figure 29. Let V be a lexical head category and D some argument or adjunct category not mentioned in any LP constraint. We further assume binary a rule schema V → V X or four individual ID rules that combine V with A, B,C, D. If we now parse the two sequences: V A D C, V B C D B, we get the following two parse trees:

[9]An algorithm was first provided by Shieber (1984) as an extension of the Earley algorithm.

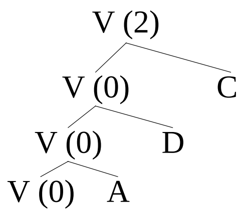


Figure 30

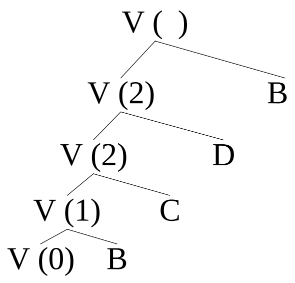


Figure 31

The numbers in parentheses indicate the lp-state. Since there is no lp-state for the top-most node in figure 31, the ungrammtical derivation is blocked.

Several entensions need to be added in order to cover the whole range of word order phenomena accross natural languages. The addition of LP constraints for head-final rule systems is rather easy. Since the mirror image of a regular language is also a regular language, there is no problem to formulate the corresponding automata. Even the extension to languages with head-medial construction does not pose a real obstacle, for it is possible to formulate an automaton that parses a regular input from a fixed position within the input string just like a FSA variant of island parsing.

An Encoding of the LP-FSA in HPSG

At first glance linguists might view the FSA approach to the encoding of LP constraints as a mere computational hack that has a very indirect connection to the rest of the actual grammar. In this subsection we will show that an LP-FSA might also be encoded in the type system of a principle-based grammar. We do not argue for such an encoding in the context af ALEP. Although it would in principle be possible to compile grammars with such automata into level 1 ALEP grammars, the inherent nondeterminism might bring processing to a grinding halt. It would be quite crazy to design a finite-state encoding for LP constraints and then implement the automaton as a highly disjunctive unification grammar.

The following encoding serves the sole purpose to show that the LP-FSA can be viewed as an integral part of the grammatical type system. This is of theoretical relevance, since one of the most-cited reasons for employing typed unification grammars is the clean uniform model-theoretic semantics of the underlying formalism.

We can transform the FSA into an expression in the typed constraint language of some unification grammar such as HPSG. Thus the Linearizing Principle(s) can be encoded the same way any other HPSG principle is encoded.

For the sake of simplicity we will start again with a unidirectional branching.

We will demonstrate this encoding first for a unidirectional LP-FSA without weighted LP Principles.

We start by defining a new feature lp-state that we will place under DTRS for simplicity (fn).

For each member of δ, $((q_i, \alpha), q_j)$, where q_i and q_j are states and α is feature term of type sign, we construct a feature term τ of the type *phrasal-sign,* which takes the form.

$$\left[\text{DTRS} : \begin{bmatrix} \text{H-DTR} : \left[\text{DTRS} : \left[\text{LP-STATE} : q_i\right]\right] \\ \text{A-DTR} : \alpha \\ \text{LP-STATE} : q_j \end{bmatrix}\right]$$

Figure 32

For simplicity, we have decided to place the feature LP-STATE under DTRS. Thus we consider the lp-wellformedness of the subtree to be a property of the phrase structure component of the sign. Although this decision allows a more compact encoding, nothing actually hinges on it. If this term is unified with a full-blown phrasal sign states that the LP-STATE of the constituent is a successor state of the LP-STATE of its head that can be reached by consuming the input symbol α. Thus the LP-STATE of the sign is computed from the LP-STATE of its head and the sign of its argument or adjunct by the LP-FSA.

Every phrasal sign must be licensed by one of the terms τ corresponding to the transitions in the LP-FSA. Therefore, the lp-terms form a disjunctive constraint. Following HPSG practice, the LP-Principle may then be written as:

Linearization Principle:

$$\left[\text{DTRS} : {}_{\textit{headed-structure}}[\;]\right] \Rightarrow \tau_1 \vee \tau_2 \vee \ldots \vee \tau_n$$

Figure 33

Following xxx and also our own practice, the disjunctive constraint in the consequent of the principle can be made part of the definition for the type *phrasal-sign.*

Depending on the aesthetic needs of the grammarian and on the computational use of the principle, the disjunctive constraint may be compacted in several ways by utilizing the equivalences of the underlying feature logic.

The following example shows the encoding of the automaton in figure 29 as a disjunctive feature term. For the sake of easy readability we have used curly braces as circumfix operators for indicating disjunctions.

$$\boxed{\left[\text{DTRS} : \left\{\begin{array}{l}
\left[\begin{array}{l}
\text{H–DTR} : [\text{LP–STATE} : 0] \\
\left\{\begin{array}{l}
\left[\begin{array}{l}\text{A–DTR} : \neg\left\{\begin{array}{c}\text{B}\\ \text{C}\end{array}\right\} \\ \text{LP–STATE} : 0\end{array}\right] \\
\left[\begin{array}{l}\text{A–DTR} : \text{B} \\ \text{LP–STATE} : 1\end{array}\right] \\
\left[\begin{array}{l}\text{A–DTR} : \text{C} \\ \text{LP–STATE}: 2\end{array}\right]
\end{array}\right\}
\end{array}\right] \\
\left[\begin{array}{l}
\text{H–DTR} : [\text{LP–STATE} : 1] \\
\left\{\begin{array}{l}
\left[\begin{array}{l}\text{A–DTR} : \neg\left\{\begin{array}{c}\text{A}\\ \text{C}\end{array}\right\} \\ \text{LP–STATE} : 1\end{array}\right] \\
\left[\begin{array}{l}\text{A–DTR} : \text{C} \\ \text{LP–STATE} : 2\end{array}\right]
\end{array}\right\}
\end{array}\right] \\
\left[\begin{array}{l}
\text{H–DTR} : [\text{LP–STATE} : 2] \\
\left\{\left[\begin{array}{l}\text{A–DTR} : \neg\left\{\begin{array}{c}\text{A}\\ \text{B}\end{array}\right\} \\ \text{LP–STATE} : 2\end{array}\right]\right\}
\end{array}\right]
\end{array}\right\}\right]}$$

Figure 34

REFERENCES

[ALEP 1993]
ALEP System Documentation: Version 1.0. BIM.

[Alshawi et al. 1991]
Alshawi, H., D. J. Arnold, R. Backofen, D. M. Carter, J. Lindop, K. Netter, J. Tsujii and H. Uszkoreit. Eurotra 6/1: Rule Formalism and Virtual Machine Design Study. Final Report. Cambridge, SRI International.

[Bouma and van Noord 1993]
Gosse Bouma and Gertjan van Noord. Head-driven Parsing for Lexicalist Grammars: Experimental Results. Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics. Utrecht.

[Barton 1985]
G. Edward Barton Jr. *On the Complexity of ID/LP Parsing.* In: Computational Linguistics 11:4, 205-218.

[Dörre 1991]
Jochen Dörre. The Language of STUF. In: Herzog, O. and Rollinger, C.-R. (eds.): Text Understanding in LILOG. Springer, Berlin.

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

[Erbach 1991]
Gregor Erbach. A flexible parser for a linguistic experimentation environment. In: Herzog, O. and Rollinger, C.-R. (eds.): *Text Understanding in LILOG*. Springer, Berlin.

[Erbach et al. 1994]
G. Erbach, M. van der Kraan, S. Manandhar, M. A. Moshier, H. Ruessink and C. Thiersch. LRE-61-061: "The Reusability of Grammatical Resources": Deliverable B: Specification of Datatypes.

[Gazdar & Pullum 1982]
Gerald Gazdar, G. K. Pullum. Generalized Phrase Structure Grammar. A Theoretical Synopsis. Indiana Linguistics Club, Bloomington, Indiana.

[Gazdar et al. 1985]
Gerald Gazdar, Ewan Klein, G. K. Pullum, Ivan Sag. Generalized Phrase Structure Grammar. Basil Blackwell, Oxford, UK

[Hawkins 1992]
John A. Hawkins. *A Performance Theory of Order and Constituency.* Cambridge University Press, Cambridge.

[Joshi 1987]
A. K. Joshi. Word-Over Variation in Natural Language Generation. In: Proceedings of AAAI-87, 550-555

[Kaplan & Zaenen 1988]
R. M. Kaplan, A. Zaenen. Functional Uncertainty and Functional Precedence in Continental West Germanic. In: H. Trost (ed.), Proceedings of 4. Österreichische Artificial-Intelligence-Tagung. Springer, Berlin.

[Kay 1985]
Martin Kay. Parsing in Functional Unification Grammar. In: D. Dowty, L. Karttunen and A. Zwicky (eds.), Natural Language Parsing. Cambridge University Press, Cambidge, UK.

[van Noord 1993]
Gertjan van Noord. Reversibility in Natural Language Processing. Doctoral dissertation, Rijksuniversiteit Utrecht.

[Pollard 1990]
Carl Pollard. On Head Non-Movement. In: *Proceedings of the Symposium on Discontinuous Constituency*, Tilburg, ITK.

[Pollard & Sag 1987]
Carl Pollard, Ivan Sag. Information-based syntax and semantics. Vol. 1: Fundamentals. CSLI Lecture Notes No. 13, Stanford, CA.

[Reape 1989]
Mike Reape. A Logical Treatment of Semi-Free Word Order and Bounded Discontinuous Constituency. In: *Proceedings of the 4th Meeting of the European Chapter of the ACL*, Manchester, UK.

[Reape 1990]
Mike Reape. A Theory of Word Order and Discontinuous Constituency in West Germanic. In: E. Engdahl, M. Reape (eds.): Parametric Variation in Germanic and Romance: Preliminary Investigations. ESPRIT Basic Research Action 3175 DYANA, Deliverable R1.1.A

[Reape 1991]
Mike Reape. Word Order in Germanic and Parsing. ESPRIT Basic Research Action 3175 DYANA, Deliverable R1.1.C

[Seiffert 1991]
Roland Seiffert. Unification-ID/LP Grammars: Formalization and Parsing. In: Herzog, O. and Rollinger, C.-R. (eds.): *Text Understanding in LILOG.* Springer, Berlin.

[Shieber 1984]
Stuart M. Shieber. Direct parsing of ID/LP grammars. In: Linguistics and Philosophy 7, 135 - 154.

[Shieber 1985]
Stuart M. Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. Proceedings of ACL 23rd Annual Meeting 145 – 152.

[Simpkins 1992]
Neil Simpkins. ALEP-0 Prototype Virtual Machine: User Guide, Version 2.0. Egham, Surrey. PE International.

[Uszkoreit 1986]
Hans Uszkoreit. Linear Precedence in Discontinuous Constituents: Complex Fronting in German. CSLI Report CSLI-86-47. Stanford, CA.

[Uszkoreit 1991a]
Hans Uszkoreit. Strategies for Adding Control Information to Declarative Grammars. Proceedings of ACL '91, Berkeley.