Kahane, S. & Polguère, A. (Eds, 1998): Proc. of COLING-ACL Workshop
"Processing of Dependency-Based Grammars", pages 29–38

How to define a context-free backbone for DGs: Implementing a DG in the LFG formalism

Norbert Bröker
Universität Stuttgart
Azenbergstr. 12
D-70174 Stuttgart
NOBI@IMS.UNI-STUTTGART.DE

Abstract

This paper presents a multidimensional Dependency Grammar (DG), which decouples the dependency tree from word order, such that surface ordering is not determined by traversing the dependency tree. We develop the notion of a *word order domain structure*, which is linked but structurally dissimilar to the syntactic dependency tree. We then discuss the implementation of such a DG using constructs from a unification-based phrase-structure approach, namely Lexical-Functional Grammar (LFG). Particular attention is given to the analysis of discontinuities in DG in terms of LFG's functional uncertainty.

1 Introduction

Recently, the concept of valency has gained considerable attention. Not only do all linguistic theories refer to some reformulation of the traditional notion of valency (in the form of θ-grid, subcategorization list, argument list, or extended domain of locality); there is a growing number of parsers based on binary relations between words (Eisner, 1997; Maruyama, 1990). Even theories based on phrase structure may have processing models based on relations between lexical items (Rambow & Joshi, 1994).

Against the background of this interest in the valency concept, and the fact that word order is one of the main difference between phrase-structure based approaches (henceforth PSG) and dependency grammar (DG), this paper will propose a word order description for DG and describe its implementation. First, we will motivate the separation of surface order and dependency relations within DG, and make a specific architectural proposal for their linking. Second, we will briefly sketch Lexical-Functional Grammar (LFG), and then show in detail how one might use the formal constructs provided by LFG to encode the proposed DG architecture.

Our position will be that dependency relations are motivated semantically (Tesnière, 1959), and need not be projective. We argue for so-called *word order domains*, consisting of partially ordered sets of words and associated with nodes in the dependency tree. These order domains constitute a tree defined by set inclusion, and surface word order is determined by traversing this tree. A syntactic analysis therefore consists of two linked, but dissimilar trees.

The paper thus sheds light on two questions. A very early result on the weak generative equivalence of context-free grammars and DGs suggested that DGs are incapable of describing surface word order (Gaifman, 1965). This result has been criticised to apply only to impoverished DGs which do not properly represent formally the expressivity of contemporary DG variants (Neuhaus & Bröker, 1997), and our use of a context-free backbone with further constraints imposed by dependency relations further supports the view that DG is not a notational variant of context-free grammar. The second question addressed is that of efficient processing of discontinuous DGs. By converting a native DG grammar into LFG rules, we are able to profit from the state of the art in context-free parsing technology. A context-free base (or skeleton) has often been cited as a prerequisite for practical applicability of a natural language grammar (Erbach & Uszkoreit, 1990), and we here show that a DG can meet this criterion with ease.

Sec. 2 will briefly review approaches to word order in DG, and Sec. 3 introduces word order domains as our proposal. LFG is briefly introduced in Sec. 4, and the encoding of DG within the LFG framework is the topic of Sec. 5.

2 Word Order in DG

A very brief characterization of DG is that it recognizes only lexical, not phrasal nodes, which are linked by directed, typed, binary relations to form a dependency tree (Tesnière, 1959; Hudson, 1993). If these relations are motivated semantically, such dependency trees can be non-projective. Consider the extracted NP in "*Beans, I know John likes*". A projective tree would require "*Beans*" to be connected to either "*I*" or "*know*" – none of which is conceptually directly related to "*Beans*". It is "*likes*" that determines syntactic features of "*Beans*" and which provides a semantic role for it. The only connection between "*know*" and "*Beans*" is that the finite verb allows the extraction of "*Beans*", thus defining order restrictions for the NP. The following overview of DG flavors shows that various mechanisms (global rules, general graphs, procedural means) are generally employed to lift the limitation of projectivity and discusses some shortcomings of these proposals.

Functional Generative Description (Sgall et al., 1986) assumes a language-independent *underlying order*, which is represented as a projective dependency tree. This abstract representation of the sentence is mapped via *ordering rules* to the concrete surface realization. Recently, Kruijff (1997) has given a categorial-style formulation of these ordering rules. He assumes associative categorial operators, permuting the arguments to yield the surface ordering. One difference to our proposal is that we argue for a representational account of word order (based on valid structures representing word order), eschewing the non-determinism introduced by unary categorial operators; the second difference is the avoidance of an underlying structure, which stratifies the theory and makes incremental processing difficult.

Meaning-Text Theory (Melc'ŭk, 1988) assumes seven *strata of representation*. The rules mapping from the unordered dependency trees of surface-syntactic representations onto the annotated lexeme sequences of deep-morphological representations include *global ordering rules* which allow discontinuities. These rules have not yet been formally specified (Melc'ŭk & Pertsov, 1987p.187f) (but see the proposal by Rambow & Joshi (in print)).

Word Grammar (WG, Hudson (1990)) is based on *general graphs* instead of trees. The ordering of two linked words is specified together with their dependency relation, as in the proposition "`object of verb follows it`". Extraction of, e.g., objects is analyzed by establishing an additional dependency called `visitor` between the verb and the extractee, which requires the reverse order, as in "`visitor of verb precedes it`". Resulting inconsistencies, e.g. in case of an extracted object, are not resolved. This approach compromises the semantic motivation of dependencies by adding *purely order-induced dependencies*.

Dependency Unification Grammar (DUG, Hellwig (1986)) defines a tree-like data structure for the representation of syntactic analyses. Using morphosyntactic features with special interpretations, a word defines *abstract positions* into which modifiers are mapped. Partial orderings and even discontinuities can thus be described by allowing a modifier to occupy a position defined by some transitive head. The approach requires that the *parser* interprets several features in a special way, and it cannot restrict the scope of discontinuities.

Slot Grammar (McCord, 1990) employs a number of rule types, some of which are exclusively concerned with precedence. So-called head/slot and slot/slot *ordering rules* describe the precedence in projective trees, referring to arbitrary predicates over head and modifiers. Extractions (i.e., discontinuities) are merely handled by a mechanism built into the *parser*.

3 Word Order Domains

Extending the previous discussion, we require the following of a word order description for DG:

- not to compromise the semantic motivation of dependencies,
- to be able to restrict discontinuities to certain constructions and delimit their scope,
- to be lexicalized without requiring lexical ambiguities for the representation of ordering alternatives,
- to be declarative (i.e., independent of an analysis procedure), and

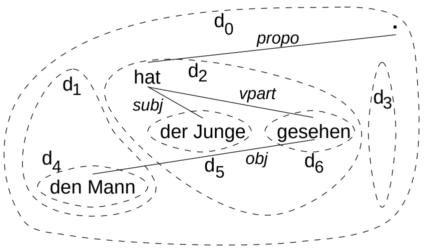

Figure 1: Dependency Tree and Order Domains for (1)

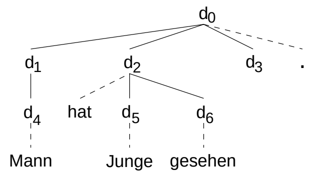

Figure 2: Order Domain Structure for (1)

- to be formally precise and consistent.

The subsequent definition of an order domain structure and its linking to the dependency tree satisify these requirements.

3.1 The Order Domain Structure

A *word order domain* is a set of words, generalizing the notion of positions in DUG. The cardinality of an order domain may be restricted to at most one element, at least one element, or – by conjunction – to exactly one element. Each word is associated with a sequence of order domains, one of which must contain the word itself, and each of these domains may require that its elements have certain features. Order domains can be partially ordered based on set inclusion: If an order domain d contains word w (which is not associated with d), every word w' contained in a domain d' associated with w is also contained in d; therefore, $d' \subset d$ for each d' associated with w. This partial ordering induces a tree on order domains, which we call the *order domain structure*. The order domain structure constitutes a projective tree over the input, where order domains loosely correspond to partial phrases.

(1) Den Mann hat der Junge gesehen.
the man$_{ACC}$ has the boy$_{NOM}$ seen
'The boy has seen the man.'

Take the German example (1). Its dependency tree is shown in Fig. 1, with word order domains indicated by dashed circles. The finite verb, "*hat*", defines a sequence of domains, $\langle d_1, d_2, d_3 \rangle$, which roughly correspond to the topological fields in the German main clause. The nouns and the participle each define a single order domain. Set inclusion gives rise to the domain structure in Fig. 2, where the individual words are attached by dashed lines to their including domains.

3.2 Surface Ordering

How is the surface order derived from an order domain structure? First of all, the ordering of domains is inherited by their respective elements, i.e., "*Mann*" precedes (any element of) d_2, "*hat*" follows (any element of) d_1, etc.

Ordering within a domain, e.g., of "*hat*" and d_6, or d_5 and d_6, is based on precedence predicates (adapting the precedence predicates of WG). There are two different types, one ordering a word with respect to any other element of the domain it is associated with (e.g., "*hat*" with respect to d_6), and another ordering two modifiers, referring to the dependency relations they occupy (d_5 and d_6, referring to `subj` and `vpart`). A verb like "*hat*" introduces three precedence predicates, requiring other words (within the same domain) to follow itself and the participle to follow subject and object, resp.:[1]

$$\begin{array}{rl} \text{“}hat\text{”} \Rightarrow & <_* \\ \wedge & \texttt{subj} < \texttt{vpart} \\ \wedge & \texttt{obj} < \texttt{vpart} \end{array}$$

Informally, the first conjunct is satisfied by any domain in which no word precedes "*hat*", and the second and third conjuncts are satisfied by any domain in which no `subject` or `object` follows a participle (`vpart`). The `obj` must be mentioned for "*hat*", although "*hat*" does not directly govern objects, because objects may be placed by "*hat*", and not their immediate governors. The domain structure in Fig.2 satisfies these restrictions since nothing follows the participle, and because "*den Mann*" is not an element of d_2, which contains "*hat*". This is an important interaction of order domains and precedence predicates: Order domains define scopes

[1]For more details on the exact syntax and the semantics of these propositions, see (Bröker, 1998b).

for precedence predicates. In this way, we take into account that dependency trees are flatter than PS-based ones[2] and avoid the formal inconsistencies noted above for WG.

3.3 Linking Domain Structure and Dependency Tree

Order domains easily extend to discontinuous dependencies. Consider the non-projective tree in Fig.1. Assuming that the finite verb governs the participle, no projective dependency between the object "*den Mann*" and the participle "*gesehen*" can be established. We allow non-projectivity by loosening the linking between dependency tree and domain structure: A modifier (e.g., "*Mann*") may not only be inserted into a domain associated with its direct head ("*gesehen*"), but also into a domain of a transitive head ("*hat*"), which we will call the *positional head*.

The possibility of inserting a word into a domain of some transitive head raises the questions of how to require continuity (as needed in most cases), and how to limit the distance between the governor and the modifier. Both questions will be solved with reference to the dependency relation. From a descriptive viewpoint, the *syntactic construction* is often cited to determine the possibility and scope of discontinuities (Bhatt, 1990; Matthews, 1981). In PS-based accounts, the construction is represented by phrasal categories, and extraction is limited by bounding nodes (e.g., Haegeman (1994), Becker et al. (1991)). In dependency-based accounts, the construction is represented by the dependency relation, which is typed or labelled to indicate constructional distinctions which are configurationally defined in PSG. Given this correspondence, it is natural to employ dependencies in the description of discontinuities as follows: For each modifier, a set of dependency types is defined which may link the direct head and the positional head of the modifier ("*gesehen*" and "*hat*", respectively). If this set is empty, both heads are identical and a continuous attachment results. The impossibility of extraction from, e.g., a finite verb phrase follows from the fact that the dependency embedding finite verbs, `propo`, may not appear on any path between a direct and a positional head.

[2]Note that each phrasal level in PS-based trees defines a scope for linear precedence rules, which only apply to sister nodes.

4 A Brief Review of LFG

This section introduces key concepts of LFG which are of interest in Sec. 5 and is necessarily very short. Further information can be found in Bresnan & Kaplan (1982) and Dalrymple et al. (1995).

LFG posits several different representation levels, called *projections*. Within a projection, a certain type of linguistic knowledge is represented, which explains differences in the formal setup (data types and operations) of the projections. The two standard projections, and those used here, are the constituent (c-) structure and the functional (f-) structure (Kaplan (1995) and Halvorsen & Kaplan (1995) discuss the projection idea in more detail). C-structure is defined in terms of context-free phrase structure rules, and thus forms a projective tree of categories over the input. It is assumed to encode language particularities with respect to the set of categories and the possible orderings. The f-structure is constructed from additional annotations attached to the phrase structure rules, and has the form of an attribute-value matrix or feature structure. It is assumed to represent more or less language-independent information about grammatical functions and predicate-argument structure. In addition to the usual unification operation, LFG employs existential and negative constraints on features, which allow the formulation of constraints about the existence of features without specifying the associated value.

Consider the following rules, which are used for illustration only and do not constitute a canonical LFG analysis.

$$
\begin{array}{rccc}
\mathrm{S} \Rightarrow & \mathrm{NP} & \mathrm{VP} & \\
 & (\uparrow\textsc{obj})=\downarrow & \uparrow=\downarrow & \\
 & (\downarrow\textsc{case})=\mathrm{acc} & & \\
 & & & \\
\mathrm{NP} \Rightarrow & \mathrm{Det} & \mathrm{N} & \\
 & (\uparrow\textsc{spec})=\downarrow & \uparrow=\downarrow & \\
 & & & \\
\mathrm{VP} \Rightarrow & \mathrm{V} & \mathrm{NP} & \mathrm{V} \\
 & \uparrow=\downarrow & (\uparrow\textsc{subj})=\downarrow & (\uparrow\textsc{vcomp})=\downarrow \\
 & (\downarrow\textsc{tense}) & (\downarrow\textsc{case})=\mathrm{nom} & \sim(\downarrow\textsc{tense})
\end{array}
$$

Assuming reasonable lexical insertion rules, the context-free part of these rules assigns the c-structure to the left of Fig. 3 to example (1). The annotations are associated with right-hand side elements of the rules and define the

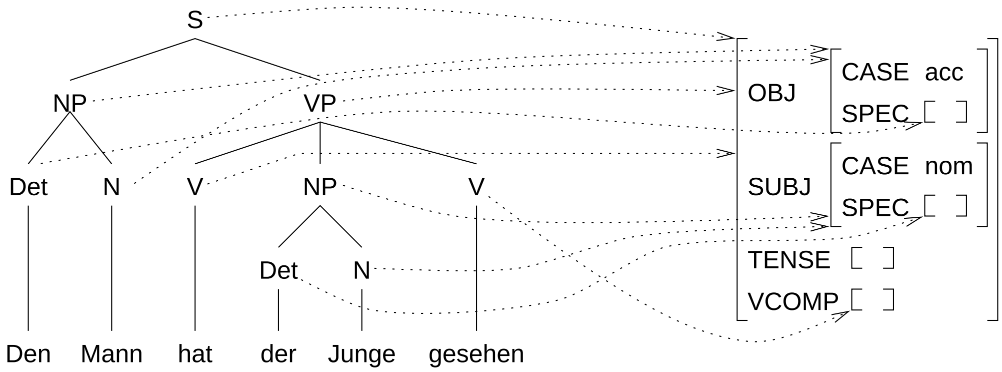

Figure 3: C-structure (left) and f-structure (right) for (1)

f-structure of the sentence, which is displayed to the right of Fig. 3. Each c-structure node is associated with an f-structure node as shown by the arrows. The f-structure node associated with the left-hand side of a rule may be accessed with the ↑ metavariable, while the f-structure node of a right-hand side element may be accessed with the ↓ metavariable. The mapping from c-structure nodes to f-structure nodes is not one-to-one, however, since the feature structures of two distinct c-structure nodes may be identified (via the ↑=↓ annotation), and additional embedded features may be introduced (such as CASE). Assuming that only finite verbs carry the TENSE feature, the existential constraint (↓TENSE) requires a finite verb at the beginning of the VP, while the negative constraint ∼(↓TENSE) forbids finite verbs at the end of the VP. Note that unspecified feature structures are displayed as [ ] in the figure, and that much more information (esp. predicate-argument information) will come from the lexical entries.

Another important construct of LFG is functional uncertainty (Kaplan & Zaenen, 1995; Kaplan & Maxwell, 1995). Very often (most notably, in extraction or control constructions) the path of f-structure attributes to write down is indeterminate. In this case, one may write down a description of this path (using a regular language over attribute names) and let the parser check every path described (possibly resulting in ambiguities warranted by f-structure differences only). Our little grammar may be extended to take advantage of functional uncertainty in two ways. First, if you want to permute subject and object (as is possible in German), you might change the S rule to the following:

$$\mathrm{S} \Rightarrow \underset{(\uparrow\{\textsc{obj} \mid \textsc{subj}\})=\downarrow}{\mathrm{NP}} \quad \underset{\uparrow=\downarrow}{\mathrm{VP}}$$

The f-structure node of the initial NP may now be inserted in either the OBJ or the SUBJ attribute of the sentence's f-structure, which is expressed by the disjunction {OBJ|SUBJ} in the annotation. (Of course, you have to restrict the CASE feature suitably, which can be done in the verb's subcategorization.) The other regular notation which we will use is the Kleene star. Assume a different f-structure analysis, where the object of infinite verbs is embedded under VCOMP. The S rule from above would have to be changed to the following:

$$\mathrm{S} \Rightarrow \underset{(\uparrow\{(\textsc{vcomp})\ \textsc{obj} \mid \textsc{subj}\})=\downarrow}{\mathrm{NP}} \quad \underset{\uparrow=\downarrow}{\mathrm{VP}}$$

But this rule will only analyse verb groups with zero or one auxiliary, because the VCOMP attribute is optional in the path description. Examples like *Den Mann will der Junge gesehen haben* with several auxiliaries are not covered, because the main verb is embedded under (VCOMP VCOMP). The natural solution is to use the Kleene star as follows, which allows zero or more occurrences of the attribute VCOMP.

$$\mathrm{S} \Rightarrow \underset{(\uparrow\{\textsc{vcomp}^*\ \textsc{obj} \mid \textsc{subj}\})=\downarrow}{\mathrm{NP}} \quad \underset{\uparrow=\downarrow}{\mathrm{VP}}$$

A property which is important for our use of functional uncertainty is already evident from these examples: Functional uncertainty is non-constructive, i.e., the attribute paths derived from such an annotation are not constructed anew (which in case of the Kleene star would lead to infinitely many solutions), but must already exist in the f-structure.

5 Encoding DG in LFG

5.1 The Implementation Plattform

The plattform used is the Xerox Linguistic Environment (XLE, see also `http://www.parc.xerox.com/istl/groups/nltt/xle/`), which implements a large part of LFG theory plus a number of abbreviatory devices. It includes a parser, a generator, support for two-level morphology and different types of lexica as well as a user-friendly graphical interface with the ability to browse through the set of analyses, to work in batch mode for testing purposes, etc.

We will be using two abbreviatory devices below, which are shortly introduced here. Both do not show up in the final output, rather they allow the grammar writer to state various generalizations more succintly. The first is the so-called *metacategory*, which allows several c-structure categories to be merged into one. So if we are writing (2), we introduce a metacategory `domVfin` (representing the domain sequence of finite verbs) to be used in other rules, but we will never see such a category in the c-structure. Rather, the expansion of the metacategory is directly attached to the mother node of the metacategory (cf. Fig. 4).

(2) `domVfin = domINITIAL domMIDDLE domFINAL.`

The second abbreviatory construct is the *template*, which groups several functional annotations under one heading, possibly with some parameters. A very important template is the `VALENCY` template defined in (3), which defines a dependency relation on f-structure (see below for discussion). We require three parameters (each introduced by underscore), the first of which indicates optionality (`opt` vs. `req` values), the second gives the name of the dependency relation, and the third the word class required of the modifier. (4) shows a usage of a template, which begins with an @ (at) sign and lists the template name with any parameters enclosed in parentheses.

(3)
```
VALENCY (_o _d _c) = {   _o = opt
                         ~(↑_d)
                       | (↑_d CLASS) = _c
                         (↑_d LEXEME) }.
```

(4) `@(VALENCY req OBJ N).`

5.2 Topological fields

As we have seen in Sec. 3, the order domain structure is a projective tree over the input. So it is natural to encode the domain structure in context-free rules, resulting in a tree as shown in Fig. 4. Categories which have a status as order domains are named `dom*`, to be distinguishable from preterminal categories (such as `Vfin`, `I`, ...; these cannot be converted to metacategories). As notational convention, `domC` will be the name of the (meta)category defining the order domain sequence for a word of class `C`. Eliminating the preterminal categories yields exactly the domain structure given in Fig. 2.

A complete algorithmic description of how to derive phrase-structure rules from order domain definitions would require a lenghty introduction to more of XLE's c-structure constructs, and therefore we illustrate the conversion with hand-coded rules. For example, a noun introduces one order domain without cardinality restrictions. Assuming a metacategory `DOMAIN` standing for an arbitrary domain, we define the following rules for the domain sequences of nouns, full stops, and determiners:

(5)
```
domN ⇒ DOMAIN* N DOMAIN*.
domI ⇒ DOMAIN I.
domD ⇒ D.
```

A complex example is the finite verb, which introduces three domains, each with different cardinality restrictions. This is encoded in the following rules:

(6)
```
domVfin     = domINITIAL domMIDDLE domFINAL.
domINITIAL ⇒ DOMAIN.
domMIDDLE  ⇒ DOMAIN* Vfin DOMAIN*.
domFINAL   ⇒ ( DOMAIN ).
```

Note the use of a metacategory here, which does not appear in the c-structure output (as seen in Fig. 4), but still allows you to refer to all elements placed by a finite verb in one word. The definition of `DOMAIN` is trivial: It is just a metacategory expandable to every domain:[3]

[3] A number of efficiency optimizations can be directly compiled into these c-structure rules. Mentioning `DOMAIN` is much too permissive in most cases (e.g., within the NP), and can be optimized to allow only domains introduced by words which may actually be modifiers at this point.

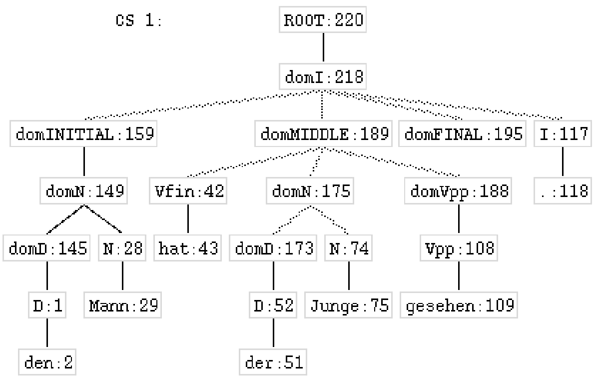

Figure 4: C-structure for (1)

(7) `DOMAIN = { domVfin | domI | domN | domD }.`

5.3 Valencies and Dependency Relations

The dependency tree is, at least in our approach, an unordered tree with labelled relations between nodes representing words. This is very similar to the formal properties of the f-structure, which we will therefore use to encode it. We have already presented the `VALENCY` template in (3) and will now explain it. $\{\cdots \mid \cdots\}$ represents a disjunction of possibilities, and the parameter `_o` (for optionality) controls their selection. In case we provide the `opt` value, there is an option to forbid the existence of the dependency, expressed by the negative constraint `~(↑_d)`. Regardless of the value of `_o`, there is another option to introduce an attribute named `_d` (for dependency) which contains a `CLASS` attribute with a value specified by the third parameter, `_c`. The existential constraint for the `LEXEME` attribute requires that some other word (which specifies a `LEXEME`) is unified into the feature `_d`, thereby filling this valency slot. The use of a defining constraint for the `CLASS` attribute constructs the feature, allowing non-constructive functional uncertainty to fill in the modifier (as explained below).

A typical lexical entry is shown in (8), where the surface form is followed by the c-structure category and some template invocations. These expand to annotations defining the `CLASS` and `LEXEME` features, and use the `VALENCY` template to define the valency frame.

(8)
```
hat Vfin  @(Vfin aux-perfect_)
          @(VALENCY req SUBJ N)
          @(VALENCY req VPART Vpp).
```

5.4 Continuous and Discontinuous Attachment

So far we get only a c-structure where words are associated with f-structures containing valency frames. To get the f-structure shown in Fig. 5 (numbers refer to c-structure node numbers of Fig. 4) we need to establish dependency relations, i.e., need to put the f-structures associated with preterminal nodes together into one large f-structure. Establishing dependency relations between the words relies heavily on the mechanism of functional uncertainty. First, we must identify on f-structure the head of each order domain sequence. For this, we annotate in every c-structure rule the category of the head word with the template `@(HEAD)`, which identifies the head word's f-structure with the order domain's f-structure (cf. (9)). Second, all other c-structure categories (which represent modifiers) are annotated with the `@(MODIFIER)` template defined in (10). This template states that the f-structure of the modifier (referenced by ↓) may be placed under some dependency attribute path of the f-structure of the head (referenced by ↑). These paths are of the form `p d`, where `p` is a (possibly empty) regular expression over dependency attributes, and `d` is a dependency attribute. `d` names the dependency relation the modifier finally fills, while `p` describes the path of dependencies which may separate the positional from the direct head of the modifier. The `MODIFIER` template thus completely describes the legal discontinuities: If `p` is empty for a dependency `d`, modifiers in dependency `d` are always continuously attached (i.e., in an order domain defined by their direct head). This is the case for the subject (in dependency `SUBJ`) and the determiner (in dependency `SPEC`), in this example. On the other hand, a non-empty path `p` allows the modifier to 'float up' the dependency tree to any transitive head reachable via `p`. In our example, objects depending on participles may thus float into domains of the finite verb (across `VPART` dependencies), and relative clauses (in dependency `RELA`) may float from the noun's domain into the finite verb's domains.

(9) `HEAD = ↓=↑.`

```
                ORDER   [RELFLAG []
                        [REL -, FIELD middle]
                        [ORDER   [REL -, FIELD middle]
                SUBJ    [SPEC 51[LEXEME definit_, CLASS D, CASE nom]
                      74[LEXEME Junge_, CLASS N, CASE nom
PROPO                   [ORDER [REL -, FIELD middle]
                        [        [ORDER [FIELD initial
                VPART   [OBJ     [      [REL   [42-ORDER-RELFLAG]]
                        [        [SPEC 1[LEXEME definit_, CLASS D, CASE acc]
                        [      28[LEXEME Mann_, CLASS N, CASE acc
                     108[LEXEME sehen_, CLASS Vpp
              42[LEXEME aux-perfect_, CLASS Vfin
117[LEXEME aussage_, CLASS I
```

Figure 5: F-structure for (1)

(10)
```
MODIFIER = ↓=(↑{PROPO
                |SUBJ
                |VPART* OBJ
                |VPART
                |SPEC
                |{SUBJ|OBJ|VPART}* RELA}).
```

The grammar defined so far overgenerates in that, e.g., relative clauses may be placed into the middle field. To require placement in specific domains, additional features are used, which distinguish topological fields (e.g., via `(↓FIELD) = middle` annotations on c-structure). A relative clause can then be constrained to occur only in the final field by adding constraints on these features. This mechanism is very similar to describing agreement or government (e.g., of case or number), which also uses standard features not discussed here. With these additions, the final rules for finite verbs look as follows:

(11)
```
domINITIAL ⇒  DOMAIN:@(MODIFIER)
                     (↓FIELD) = initial.
domMIDDLE ⇒      Vfin:@(HEAD)
                     (↓FIELD) = middle;
              DOMAIN*:@(MODIFIER)
                     (↓FIELD) = middle;
domFINAL ⇒  ( DOMAIN:@(MODIFIER)
                     (↓FIELD) = final ).
```

5.5 Missing Links

As is to be expected if you use something for purposes it was not designed to be used for, there are some missing links. The most prominent one is the lack of binary precedence predicates over dependency relations. There is, however, a close relative, which might be used for implementing precedence predicates. Zaenen & Kaplan (1995) introduced *f-precedence* $<_f$ into LFG, which allows to express on f-structure constraints on the order of the c-structure nodes mapping to the current f-structure. So we might write the following annotations to order the finite verb with respect to its modifiers, or to order subject and object.

(12) (↑) $<_f$ (↑{SUBJ|OBJ|VPART}).
(↑SUBJ) $<_f$ (↑OBJ).

The problem with f-precedence, however, is that is does not respect the scope restrictions which we defined for precedence predicates. I.e., a topicalized object is not exempt from the above constraints, and thus would result in parsing failure. To restrict the scope of f-precedence to order domains (aka, certain c-structure categories) would require an explicit encoding of these domains on f-structure.

6 Conclusion

We have presented a new approach to word order which preserves traditional notions (semantically motivated dependencies, topological fields) while being fully lexicalized and formally precise (Bröker, 1997). Word order domains are sets of partially ordered words associated with words. A word is contained in an order domain of its head, or may float into an order domain of a transitive head, resulting in a discontinuous dependency tree while retaining a projective order domain structure. Restrictions on the floating are expressed in a lexicalized fashion in terms of dependency relations. We have also shown how the order domains can be used to define a context-free backbone for DG, and used a grammar development environment for annotated phrase-structure grammars to encode the DG.

A number of questions immediately arise, some of which will hopefully be answered until the time of the workshop. On the theoretical side, this work has argued for a strict separation of precedence and categorial information in LFG (or PSG in general, see (Bröker, 1998a)). Can these analyses and insights be transferred? On the practical side, can the conversion we sketched be used to create efficient large-scale DGs? Or will the amount of f-structural indeterminacy introduced by our use of functional uncertainty lead to overly long processing? And, last and most challenging, when will the first large treebank with dependency annotation be available, and will it be derived from XLE's f-structure output?

References

Becker, T., A. Joshi & O. Rambow (1991). Long-Distance scrambling and tree-adjoining grammar. In *Proc. 5th Conf. of the European Chapter of the ACL*, pp. 21–26.

Bhatt, C. (1990). *Die syntaktische Struktur der Nominalphrase im Deutschen*. Studien zur deutschen Grammatik 38. Tübingen: Narr.

Bresnan, J. & R. Kaplan (Eds.) (1982). *The Mental Representation of Grammatical Relations*. Cambridge, MA: MIT Press.

Bröker, N. (1997). *Eine Dependenzgrammatik zur Kopplung heterogener Wissenssysteme auf modallogischer Basis*. Dissertation, Deutsches Seminar, Universität Freiburg.

Bröker, N. (1998a). A Projection Architecture for Dependency Grammar and How it Compares to LFG. In *Proc. 1998 Int'l Lexical-Functional Grammar Conference*. (accepted as alternate paper) Brisbane/AUS: Jun 30-Jul 2, 1998.

Bröker, N. (1998b). Separating Surface Order and Syntactic Relations in a Dependency Grammar. In *COLING-ACL 98 - Proc. of the 17th Intl. Conf. on Computational Linguistics and 36th Annual Meeting of the ACL*. Montreal/CAN, Aug 10–14, 1998.

Dalrymple, M., R. Kaplan, J. Maxwell & A. Zaenen (Eds.) (1995). *Formal Issues in Lexical-Functional Grammar*. CSLI Lecture Notes 47, Stanford/CA: CSLI.

Eisner, J. (1997). Bilexical Grammars and a Cubic-Time Probabilistic Parser. In *Proc. of Int'l Workshop on Parsing Technologies*, pp. 54–65. Boston/MA: MIT.

Erbach, G. & H. Uszkoreit (1990). *Grammar Engineering: Problens and Prospects*. CLAUS Report 1. Saarbrücken/DE: University of Saarbrücken.

Gaifman, H. (1965). Dependency Systems and Phrase Structure Systems. *Information and Control*, 8:304–337.

Haegeman, L. (1994). *Introduction to Government and Binding*. Oxford/UK: Basil Blackwell.

Halvorsen, P.-K. & R. Kaplan (1995). Projections and Semantic Description in Lexical-Functional Grammar. In M. Dalrymple, R. Kaplan, J. I. Maxwell & A. Zaenen (Eds.), *Formal Issues in Lexical-Functional Grammar*, pp. 279–292. Stanford University.

Hellwig, P. (1986). Dependency Unification Grammar. In *Proc. 11th Int'l Conf. on Computational Linguistics*, pp. 195–198.

Hudson, R. (1990). *English Word Grammar*. Oxford/UK: Basil Blackwell.

Hudson, R. (1993). Recent developments in dependency theory. In J. Jacobs, A. v. Stechow, W. Sternefeld & T. Vennemann (Eds.), *Syntax. Ein internationales Handbuch zeitgenössischer Forschung*, pp. 329–338. Berlin: Walter de Gruyter.

Kaplan, R. (1995). The formal architecture of Lexical-Functional Grammar. In M. Dalrymple, R. Kaplan, J. I. Maxwell & A. Zaenen (Eds.), *Formal Issues in Lexical-Functional Grammar*, pp. 7–27. Stanford University.

Kaplan, R. & J. Maxwell (1995). An Algorithm for Functional Uncertainty. In M. Dalrymple, R. Kaplan, J. I. Maxwell & A. Zaenen (Eds.), *Formal Issues in Lexical-Functional Grammar*, pp. 177–198. Stanford University.

Kaplan, R. & A. Zaenen (1995). Long-distance Dependencies, Constituent Structure, and Functional Uncertainty. In M. Dalrymple, R. Kaplan, J. I. Maxwell & A. Zaenen (Eds.), *Formal Issues in Lexical-Functional Grammar*, pp. 137–166. Stanford University.

Kruijff, G.-J. v. (1997). *A Basic Dependency-Based Logical Grammar*. Draft Manuscript. Prague: Charles University.

Maruyama, H. (1990). Structural Disambiguation with Constraint Propagation. In *Proc. 28th Annual Meeting of the ACL*, pp. 31–38. Pittsburgh/PA.

Matthews, P. (1981). *Syntax*. Cambridge Textbooks in Linguistics, Cambridge/UK: Cambridge Univ. Press.

McCord, M. (1990). Slot Grammar: A System for Simpler Construction of Practical Natural Language Grammars. In R. Studer (Ed.), *Natural*

Language and Logic, pp. 118–145. Berlin, Heidelberg: Springer.

Melc'ŭk, I. (1988). *Dependency Syntax: Theory and Practice*. Albany/NY: State Univ. Press of New York.

Melc'ŭk, I. & N. Pertsov (1987). *Surface Syntax of English: A Formal Model within the MTT Framework*. Philadelphia/PA: John Benjamins.

Neuhaus, P. & N. Bröker (1997). The Complexity of Recognition of Linguistically Adequate Dependency Grammars. In *Proc. 35th Annual Meeting of the ACL and 8th Conf. of the EACL*, pp. 337–343. Madrid, July 7-12, 1997.

Rambow, O. & A. Joshi (1994). A Processing Model for Free Word Order Languages. In C. J. Clifton, L. Frazier & K. Rayner (Eds.), *Perspectives on Sentence Processing*. Hillsdale/NJ: Lawrence Erlbaum.

Rambow, O. & A. Joshi ((in print)). A Formal Look at Dependency Grammars and Phrase-Structure Grammars, with special consideration of word-order phenomena. In L. Wanner (Ed.), *Current Issues in Meaning-Text-Theory*. London: Pinter.

Sgall, P., E. Hajicova & J. Panevova (1986). *The Meaning of the Sentence in its Semantic and Pragmatic Aspects*. Dordrecht/NL: D.Reidel.

Tesnière, L. (1959). *Eleménts de syntaxe structurale*. Paris: Klincksiek.

Zaenen, A. & R. Kaplan (1995). Formal Devices for Linguistic Generalizations: West Germanic Word Order in LFG. In M. Dalrymple, R. Kaplan, J. Maxwell & A. Zaenen (Eds.), *Formal Issues in Lexical-Functional Grammar*, pp. 215–240. CSLI Lecture Notes 47, Stanford/CA: CSLI.