A polynomial parsing algorithm for the topological model
Synchronizing Constituent and Dependency Grammars, Illustrated by German Word Order Phenomena

Abstract

This paper describes a minimal topology driven parsing algorithm for topological grammars that synchronizes a rewriting grammar and a dependency grammar, obtaining two linguistically motivated syntactic structures. The use of non-local *slash* and *visitor* features can be restricted to obtain a CKY type analysis in polynomial time. German long distance phenomena illustrate the algorithm, bringing to the fore the procedural needs of the analyses of syntax-topology mismatches in constraint based approaches like for example HPSG.

1 Introduction

In this paper we investigate the computational problems resulting from the discrepancy between the surface organization of a sentence and its syntactic structure expressing functional relations between words. These syntactic phenomena are addressed under terms like unbounded dependencies or scrambling. This discrepancy is the source of complex mechanisms such as movement in Chomsyan model, functional uncertainty in the functional equations of LFG, or non local features (like slash) in HPSG. Algorithms for these mechanisms are NP-complete and it is well known that we have to introduce upper bounds on simultaneous mismatches in order to account for these phenomena in polynomial time. The resulting complexity is of O(n^{K+5}) with K being this upper bound[1].

We consider the so-called topological grammars characterized by the synchronization of two algebraic grammars (equivalent CFG), one generating the surface structure (the topological phrase structure), the other generating the deeper structure (the functional or dependency structure). The parallel construction of the surface and the deeper structure minimally handles the problem of the discrepancy encountered in all formal descriptions of non-local word order phenomena.
The lexicalized version of topological grammars presented in this paper lends itself well to be used in a CKY type algorithm. The description of this algorithm sheds light on the procedural role of the concept underlying the *slash* feature and the dual role of the *visitor* feature introduced here (following Hudson 2000).
Formal topological grammars have been introduced independently by Debusmann & Duchier 2001 and Gerdes & Kahane 2001. Reape 1994 and Kathol 1995 have formalized the classical concept of topology in HPSG without, however, explicitly considering an independent topological phrase structure. More or less complete topological grammars have been developed for Czech, Dutch, German, Modern Greek, Korean, Arabic, and French (Bojar 2004, Yoo & Gerdes 2004, El Kassas 2005, Gerdes & Kahane 2006). Different

[1] The linguistic relevance of this upper bound has been discussed for example in Becker *et al.* 1991, arguing that Tree Adjoining Grammar capture the performance limit on extractions in German.

implementations for topological grammars exist: TDG (Debusmann et al. 2003) uses the unspecialized Oz constraint based language, making TDG parsing NP-hard as show Koller & Striegnitz 2002 (although one obtains polynomial time in the average case, Duchier 2003); DepLin is a generation system based on a procedural linearization grammar (Gerdes & Yoo 2003), and Clément et al. 2002 and Frank 2003 translate topological grammars into the LFG opposition between c- and f-structures allowing the use of LFG analyzers (for which exist NP hardness results). To our knowledge, the HPSG approach has not been implemented. None of these approaches has explored the theoretical algorithmic properties of an exact topological analysis (leaving aside stochastic approaches). However, the theoretical problems of the upper bounds for the number of extractions have been studied in neighboring dependency-based formalisms by Kahane et al. 1998 and Bröker 2000.

Our contribution shows how the topology-syntax discrepancy can minimally be seen as two grammars that synchronously construct two independent structures. We believe that the introduction of the dual slash and visitor features allows for a better comprehension of the procedural mechanisms at stake in the unbounded dependencies handling and the (not necessarily apparent) similarities between formalisms such as LFG, HPSG, or dependency grammars.

Section 2 presents the grammar formalism, exemplifying it by a German toy grammar and Section 3 is devoted to the parsing algorithm we propose.

2 The grammar formalism

Our grammar contains three modules: a syntactic grammar, a topological grammar, and the topology-syntax interface. We will present these three modules, exemplifying each of them by a toy grammar for German. Although very simple, this grammar covers a great part of the verbal syntax of German (Bech 1955), including the main scrambling phenomena. For a more complete grammar of German and for grammars of other languages in the same theoretical framework, see the references in our introduction.

It should be noted that the formalism we propose here differs slightly from the previous formal presentation of the topological model, bringing to the fore the synchronization of two grammars and the interface grammar.

2.1 The syntactic grammar

The syntactic module is a classic dependency grammar and generates unordered dependency trees. The parameters to instantiate are the vocabulary $\mathcal{V}$, the set of (lexical) categories $\mathcal{C}$, the initial category IC, the set of syntactic roles $\mathcal{R}$, and the set of lexical rules. A lexical rule assigns a category and a valence list to a word. A valence slot is a couple (*r*,C) where *r* is a syntactic role and C a category.[2] The initial category IC give the possible category of the root of the dependency tree.

Example

$\mathcal{V}$ = the German words

$\mathcal{C}$ = { Vfin, Vzu, Vinf, Vpp, N}

(Vfin = finite verb, Vzu = infinitive with *zu*, Vinf = bare infinitive, Vpp = past participle)[3]

$\mathcal{R}$ = { *subj, obj, vcomp* }

IC = Vfin

Dependency rules

hat 'has':Vfin,val:<(*subj*,Nnom),(*vcomp*,Vpp)>

gelesen 'read': Vpp, val:<(*obj*,Nacc)>

The last rule says that *gelesen* is a past participle governing a nominal object at the accusative case. Our grammar generates dependency tree such as the one of Fig. 1 for the sentence (1):

(1)	*Den Roman hat diesem Mann niemand*	*zu lesen versprochen*
	the novel has to-this man nobody	to read promised

[2] We do not present the treatment of modifiers when the governor is selected by the dependent. This does not pose any technical problems but it necessitates particular rules that we will not present here (for the treatment of modifier in a dependency grammar see for example Nasr 1995; various propositions in HPSG can also be adapted here). Neither do we expatiate upon the optionality of some syntactic arguments.

[3] For the sake of simplicity we give a very rough presentation of the category. For nouns, cases are added in their names (Nnom, Ngen, Ndat, and Nacc).

‘Nobody promised to this man to read the novel’.

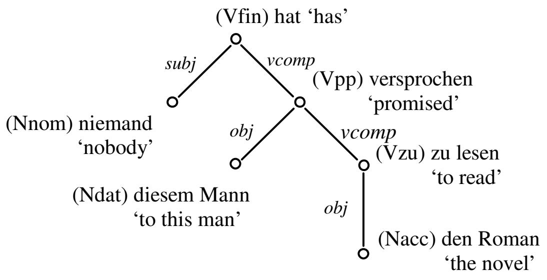


Fig. 1. A dependency tree

2.2 The topological grammar

The topological grammar proper generates the topological structures, which are ordered constituent trees. Such a grammar differs slightly from traditional CFGs by distinguishing constituents from positions for constituents, i.e. boxes (= topological constituent) and fields (= positions in a box). A grammar rule indicates for each box what its list of fields is and how many boxes it can contain for each field. Three values for the filling parameter of a field are possible: exactly one element (!), at most one element (?) and any number of elements (∗). A field is called obligatory if its filling parameter is !.

The parameters to instantiate are the set of box names $\mathcal{B}$, the set of field names $\mathcal{F}$, the initial field (if), and the set of rules.

For the sake of simplicity, in this presentation, we adopt a flat structure, with exactly one box headed by each word of the sentence. We lose a part of the economy of the system (that reuses the same boxes at different levels) but the presentation of the parsing algorithm will be clearer.

Example

$\mathcal{B}$ = { md, ed, vc, np }

(md = main domain, ed = embedded domain, vc = verb cluster, np = nominal phrase)

$\mathcal{F}$ = { vf, mf, nf, rb, of, • }

(vf = Vorfeld, mf = Mittelfeld, nf = Nachfeld, rb = right bracket, of = Oberfeld, • = head field)

ib = md

<u>Topological rules</u>

md $\rightarrow$ $vf^{!} \bullet mf^{*} rb^{?} nf^{*}$

ed $\rightarrow$ $mf^{*} of^{?} \bullet nf^{*}$

vc $\rightarrow$ $of^{?} \bullet$

Our first rule is the classical topological model of German: a main domain is composed of five fields and the main verb occupies the second field, the first field vf containing exactly one element. In the embedded domain, the head occupies the right bracket, which is then the head field. A verb in the right bracket offers a place to its left called the Oberfeld (of) for a verbal dependent. Fig. 2 is a graphical representation of the topological structure of the non marked sentence (1). Boxes are represented by circles and fields by squares.

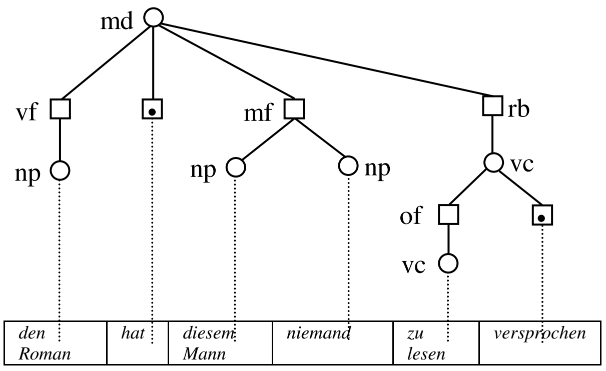


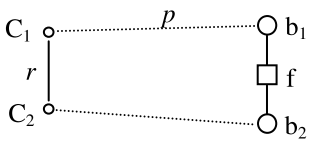
Fig. 2. The topological structure of (1)

2.3 The topology-syntax interface

The topology-syntax interface synchronizes the syntactic grammar and the topological grammar. An interface rule associates the positioning of a dependency node with the positioning of the corresponding topological box.

Each box b∈ $\mathcal{B}$ is associated to an integer p(b), called its permeability, controlling which constituent can emancipate from it.

An interface rule is a 7-tuple (C_1,r,C_2,b_1,f,b_2,p), where C_1,C_2∈ $\mathcal{C}$, r∈ $\mathcal{R}$, b_1,b_2∈ $\mathcal{B}$, f∈ $\mathcal{F}$, and p is an integer called the permeability level. The rule can be read in two equivalent ways: 1) if a word w_2 of category (cat) C_2 depends on a word w_1 of cat C_1 by a syntactic relation r, then w_2 can head a box b_2 placed in a field f of a box b_1 containing w_1 and separated from b_1 by boxes of permeability $\leq p$ (synthesis reading); 2) if a word w_2 of cat C_2 heads a box b_2 placed in a field f of a box b_1 containing a word w_1 of cat C_1 and separated from b_1 by boxes of permeability $\leq p$, then w_2 can depend on w_1 by a syntactic relation *r*. (analysis reading). The rule is schematized in the following figure:



Example

Permeability
p(vc)=1, p(ed)=p(xp)=2, p(md)=3.

Interface rules

For a noun we have one basic rule: a noun depending on a verb can head an NP[4] in any major field (vf/mf/nf) wherever is the verb; it can cross over vc and ed boxes:

(V, *subj/obj*, N, md/ed/vc, vf/mf/nf, np, 2)

Note that in German, contrarily to English, the placement of a NP does not actually depend on its syntactic role.
For a non-finite verb we have two rules:
-one is similar to the rule for nouns: a verb can head an embedded domain in any major field:

(V, *vcomp*, V¬fin, md/ed/vc, vf/mf/nf, ed, 2)

- the other is specific to the German(ic) syntax: a non-finite verb can be placed in the right bracket (rb) or at the left of its verbal governor (in the of field) if this governor is already in the right bracket:

(V, *vcomp*, V¬fin, md/ed/vc, of/rb, vc, 0)

This last rule can be applied recursively, forming a string of verbs called a verb cluster. The dependents of verbs of a same cluster can be freely shared out in the major fields of the same domain. This property produces what is called scrambling. In our model this requires an extensive use of emancipation, allowing any dependent of a verb to be placed in a domain headed by a verbal ancestor. Emancipation is also possible outside the embedded domain, although this would require specific strong information packaging constraints (not reflected in this toy grammar).

3. The parsing algorithm

We begin with a presentation of the algorithm when there is no emancipation. In this case the topological structure and the dependency structure are built in parallel, i.e. each combination of linear segments corresponds to a functional combination. When emancipations are allowed the parsing will be driven by the topological structure only.

3.1 The algorithm without emancipation

The philosophy of a CKY algorithm is to begin parsing one word segments of the sentence, to store the minimum of information in a parse matrix, and to parse bigger and bigger segments by concatenation of segments previously parsed.
In the algorithm for CFG, if we have two consecutive segments from *i* to *j* and from *j*+1 to *k* of cat C_1 and C_2 and if we have a rule $C \rightarrow C_1C_2$, then we postulate a segment from *i* to *k* of cat C. The recurrence step is then:

$$[i,j,\text{cat}:C_1] \oplus [j+1,k,\text{cat}:C_2] \otimes (C \rightarrow C_1C_2) = [i,k,\text{cat}:C]$$

In our case, the entries of our parse matrix are of the form [*i,j*,cat:C,val:X,box:b,fieldsY] where *i* and *j* delimit the segment, C is the category of the head, X is the list of free valence slots, b is the topological box name of the segment, and Y the list of non-saturated fields of b (including the head field •, indicating which fields are on the left or on the right)

Initialization step
If the *i*-th word of the sentence we want to parse can have 1) the cat C, 2) the valence X and 3) a word of cat C can head a box b and if 4) there is a topological rule b→Y, then we store the segment [*i,i*,cat:C,val:X,box:b,fields:Y].

Recurrence step
We combine two consecutive segments by applying an interface rule. One of the two segments must be saturated which means: 1) all the valence slots of the head have been filled (and thus val is an empty list) ; 2) all the fields of the box are potentially saturated, that is, there is no field with the value ! left.
In the following recurrence step, we suppose that the second segment is saturated and we note *elist* the empty valence list and *sat* the saturated field list.

[4] We do not develop the nominal topology in this example grammar.

[*i,j*,cat:C_1,val:X,box:b_1,fields:Y]
$\oplus$ [*j*+1,*k*,cat:C_2,val:*elist*,box:b_2,fields:*sat*]
$\otimes$ (C_1, *r*, C_2, b_1, f, b_2, *p*)

=[*i,k*,cat:C_1,val:X$\ominus$ <(*r*,C_2)>,box:b_1,fields:Y$\ominus$ f]

This step is possible if X contains a valence slot
(*r*,C_2) and then X$\ominus$ <(*r*,C_2)> is the list X reduced by (*r*,C_2). In the same way, Y must contain a field f at the left of the head field; the fields between • and f must be non obligatory and are suppressed in Y$\ominus$ f; moreover, the filling parameter of f is adjusted according to the fact that f now contains a box.
The parsing succeeds if our parse matrix contains at least one segment [1,*n*,cat:IC,val:*elist*, box:ib,fields:*sat*]. If we keep backpointers at each step in the algorithm, we have a compact representation of the parse forest.

3.2 The algorithm with emancipation

An emancipated constituent is not in the maximal projection of its governor, i.e. it is not in the box headed by its governor. Let is see what are the consequences. Consider the basic example (2):

(2) *Den Roman hat Maria gelesen.*
The novel has Maria read
'Maria read the novel'

In (2), *den Roman*, which depends on the past participle *gelesen*, is placed in a field of the main domain headed by the auxiliary *hat*, while the verb cluster headed by *gelesen* is unsaturated. Suppose we want to apply our previous algorithm (the CKY parsing without emancipation). We can easily parse the segments *den Roman, hat Maria,* and *gelesen*, but neither *den Roman* and *hat Maria* (no valence for *den Roman*), nor *hat Maria* and *gelesen* (*gelesen* not saturated) can be combined.

Our parsing will be driven by the topological structure and the condition of the topological saturation of the dependent is maintained. Two cases of combination of segments are possible.
The first case is illustrated by the combination between *hat Maria* and *gelesen*, where *gelesen* still expects a dependent. Therefore we do not require the valence of the topological phrase to be saturated and we must percolate it in a special feature similar to the slash feature of G/HPSG (Gazdar et al. 1985, Pollard & Sag 1994)
The second case is illustrated by the combination between *den Roman* and *hat Maria*. In this case we do not trigger a correspondence rule because no dependency must be built. We must store *den Roman* in a special feature we call *visitor* (see Hudson 2000 for a similar device), which is the converse of the slash feature. The slash feature allows us to lift up a need (a valence slot to be filled), while the visitor feature allows handing down a resource (that will fill a valence slot). Or more precisely in our case: The visitor allows a governor to keep a non-solicited segment while waiting for an element that can take this element in its valence.
Note however that, with our conditions on the saturation of topological constituents, the two strategies are not interchangeable and they are both necessary. Let us consider two new examples.

(3) *Maria hat den Roman gelesen.*
Maria has the novel read
'Maria read the novel'

Although the sentence (3) is projective, *den Roman* must be analyzed as an emancipated constituent. Indeed, *gelesen* is in the right bracket of the main domain and the maximal projection of *gelesen*, the verb cluster, does not contain its dependent *den Roman*, which is in the Mittelfeld of the main domain headed by *hat*. From a topological point of view, *den Roman* can only combine with *hat* but it is not in the valence of *hat* and it must be considered a visitor.

(4) *Ich glaube, dass Maria den Roman*
I think that Maria the novel

gelesen hat.
read has

'I think that Maria read the novel'

In (4), *gelesen hat* froms a verb cluster in the right bracket of a complementizer phrase. The NP *den Roman* is still emancipated and, from the topological viewpoint, it cannot combine with its governor *gelesen*. It also cannot combine with *hat* because they are separated by *gelesen*. The smallest topological phrase containing *den Roman* and *gelesen* also contains *hat*. Therefore the slash strategy is needed and *gelesen* and *hat* must combine before combining with *den Roman*.

We make two major changes in our previous algorithm. On the one hand we replace the val feature by two new features:

- the vis(itor) feature, which stores triples (C,b,f) indicating that a box b of head C has been placed in the field f;
- the slash feature, which stores valence slots of the head as well as the non saturated slots of its dependents.

On the other hand we proceed in two steps: first we combine consecutive segments without triggering interface rules, by storing all information in our vis and slash features; second we trigger interface rules to reduce the contents of vis and slash features.

Initialization step
The segment [*i,i*,cat:C,val:X,box:b,fields:Y] of the previous algorithm is replaced by [*i,i*,cat:C, vis:*elist*,slash:X',box:b,fields:Y] where each valence slot (*r*,C') of X gives a slash slot (C,*r*,C',0), with C the cat of the head and 0 indicating that this slash slot has not emancipated across any box.

Combination step
The combination is driven by the topological structure, so one of the two segments combined must be topologically saturated (fields:*sat*) but we no longer require that the valence of this segment is saturated: its free valence slot will be slashed.

[*i*,*j*,cat:C_1,vis:Z,slash:X_1,box:b_1,fields:Y] ⊕ [*j*+1,*k*,cat:C_2,vis:*elist*,slash:X_2,box:b_2,fields:*sat*] = [*i*,*k*,cat:C_1,vis:Z⊕<(C_2,f,b_2)>,slash:X_1⊕X_2', box:b_1,fields:Y⊖ f], where each 4-tuple (C,*r*, C',*p*) of X2 gives a 4-tuple (C,*r*,C',*p*') with *p*' the max of *p* and p(b1), the permeability of b1.

In other words the second segment is placed in the field f and this is stored in the visitor feature (cf. (C_2,f,b_2)). At the same time the slash content X2 of the second segment is added to the slash content of the first segment, but we must indicate that these valence slots have crossed the box b1 and this is why the permeability level is adjusted.

Reduction step
[*i*, *j*, cat:C, vis:Z⊕<(C_2,f,b_2)>, slash:X⊕ <(C_1,*r*,C_2,*p*')>, box:b_1, fields:Y]
⊗ (C_1, *r*, C_2, b_1, f, b_2, *p*)
= [*i, j*, cat:C, vis:Z, val:X, box:b_1, fields:Y]
provided that $p' \leq p$.

A reduction is possible if vis and slash contain elements referring to the same category C2: the vis element (C_2,f,b_2) says that we have encountered a box b2 of head C2 in the field f , while the slash element (C_1,*r*,C_2,*p*') indicates that a word of cat C2 is required to fill the *r* valence slot of a word of cat C1. If furthermore the slashed slot has not crossed over boxes of permeability greater than *p* ($p' \leq p$), then the interface rule (C_1, *r*, C_2, b_1, f, b_2, *p*) can apply and the segment can be reduced.

The parsing succeeds if the parse matrix contains a segment [1,*n*,cat:IC,vis:*elist*,slash:*elist*, box:ib,fields:*sat*], where *n* is the length of the sentence.

Example: Parsing of *Den Roman hat Maria zu lesen versprochen* (cf. (1)). We focus on the combination of the segment *den Roman hat Maria* (where *den Roman* is a visitor) with the verb cluster *zu lesen versprochen*.

den Roman hat Maria: S_1 = [1,4, cat:Vfin, vis:<(Nacc,vf,np)>, slash:<(Vfin,aux,Vpp,0)>, box:md, fields: •$mf^{*}rb^{?}nf^{*}$]

zu lesen versprochen: S_2 = [5, 7, cat:Vpp, vis:*elist*, slash:<(Vzu,dobj,Nacc,1)>, box:vc, fields:•]

The segment S_2 is topologically saturated, so S1 and S_2 can combine.

S= S_1⊕S_2 = [1,7, cat:Vfin, vis:<(Nacc,vf,np), (Vpp,rb,vc)>, slash:<(Vfin,aux,Vpp,0), (Vzu,obj,Nacc,1)>, box:md, fields: •nf^{*}]

S can be reduced twice
- by merging (aux,Vfin,Vpp,0) and (Vpp,rb,vc) using the interface rule (V,aux,V¬fin, md, rb, vc, 0)
- and by merging (Vzu,dobj,Nacc,vc) and (Nacc,vf,np) using the interface rule (V,dobj,N,md,vf,vc,2).

After reduction, S=[1,7,cat:Vfin, vis:*elist*, slash:*elist*, box:md, fields: •nf^{*}], proving that (1) is a grammatical sentence of German.

3.3 Complexity

The parse matrix of a CKY algorithm has less than n^2 entries, where n is the length of the sentence parsed. In the CFG case, the number of possible segment descriptions is bounded by the number of categories, which we call C. To fill a new entry in the parse matrix at least n combinations of two entries must be considered and the number of operations is bounded by GC^2n^3, where G is the number of rules (each combination involves a grammar rule).

In the algorithm without emancipation the number of possible descriptions of segments is still bounded and the complexity remains $O(n^3)$. However if we want to retrieve the dependency forest (Nasr 2003) we need to store backpointers, that is the place of the head in each segment description; the number of segment descriptions goes up by a factor n and the complexity becomes $O(n^5)$.

The slash and visitor features are more expensive: We assume the slash and visitor sets to be bounded by K, i.e. we suppose that we do not need to keep more than K entries in the slash and visitor sets at a time. Consequently the number of segment descriptions remains bounded by CV^K, where V is the number of valence slot descriptions, and the algorithm complexity is still of type $O(n^3)$. But if we introduce backpointers to retrieve the dependency forest, we need to keep them in valence slot descriptions in order to remember which word has a valence slot to fill. The number of segment descriptions is thus bounded by CV^Kn^{K+1} and the time complexity of the algorithm is in $O(n^{K+5})$. We avoid exponential growth only because we restrict the number of slash and visitor entries of each configuration.

4 Conclusion

We have proposed a parsing algorithm for the topological model that is minimal in the sense that its additional exponential growth (the factor K) corresponds exactly to the number of mismatches between functional dependency and topological constituency. In different terms, given we want to construct the topological and the dependency structures, two independent and linguistically significant structures, and given two separate grammars expressing the constraints on the construction of these structures, then the cost of adding the interface constraints is exponential precisely in the number of memory positions needed in order to keep track of the differences between the two structures. Precise parsing of these three grammars cannot do with less.

We could hypothesize that the need of differentiating more or less independent levels of syntactic analysis (e.g. surface vs. deep structure, …), which is at the origin of formalisms like LFG or HPSG, algorithmically boils down to this exponentiality, at least concerning the syntax/surface expressive needs. It might be interesting to compare these results with efficiency considerations for HPSG as in Nishida et al. 2001 and for restricted graph grammars for dependency-orientated generation (Bohnet & Wanner 2001).

Our bottom-up strategy driven by the topological structure forces us to introduce tools equivalent to the slash feature of G/HPSG. We hope that this presentation sheds light on the procedural role of the slash feature, and on the complementary possibility of a linguistic analysis using a visitor feature.

It should be noted that, in spite of its simplicity, the German topological grammar presented as an example allows the control of syntactic constraints on phenomena like scrambling, partial VP fronting, and auxiliary flip, which demonstrates the expressivity of the topological approach. The grammars of languages like Czech and Modern Greek show that the topological approach allows for a straightforward integration of information structure in the interface constraints.

Work is in progress on experimental implementations of the presented algorithm and on choosing useful and linguistically accessible input and output formats. Real values on efficiency will not be available as long as the grammar does not surpass experimental size. A linguistic study on corpora might determine what types of elements are actually emancipated and in particular what types of elements can be emancipated simultaneously, i.e. what list of slashed element are possible, given that this is the main factor of complexity of the algorithm (see Kiefer et al. 1999 for similar heuristic considerations for HPSG parsing).

References

Bech, Gunnar, 1955, *Studien über das deutsche Verbum infinitum*, 2nd edition 1983, Linguistische Arbeiten 139, Niemeyer, Tübingen.

Becker, Tilman, Aravind K. Joshi, Owen Rambow, 1991, "Long-Distance Scrambling and Tree Adjoining Grammars". *EACL 1991*

Bohnet, Bernd and Leo Wanner, 2001, "On Using a Parallel Graph Rewriting Grammar Formalism in Generation". *Proceedings of the 8th European Natural Language Generation Workshop (at ACL)*, Toulouse.

Bojar, Ondřej, 2004, "Problems of Inducing Large Coverage Constraint-Based Dependency Grammar for Czech." *International Workshop on Constraint Solving and Language Processing*, Universitet, Roskilde, pp. 29-42.

Bröker, Norbert, 1998, "Separating Surface Order and Syntactic Relations in a Dependency Grammars", *COLING-ACL'98*, 174-180.

Debusmann, Ralph, Denys Duchier and Joachim Niehren, 2004, "The XDG Grammar Development Kit", *Second International Mozart/Oz Conference*, Charleroi

Drach, Erich, 1937, *Grundgedanken der deutschen Satzlehre*, Diesterweg, Frankfurt/M..

Duchier, Denys, Ralph Debusmann, 2001, "Topological Dependency Trees: A Constraint-Based Account of Linear Precedence", *ACL 2001*, 180-87.

Duchier, Denys, 2003, "Configuration of labelled trees under lexicalized constraints and principles", *Journal of Research on Language and Computation*, Sep 2003.

El Kassas, Dina, Sylvain Kahane, 2004, Modélisation de l'ordre des mots en arabe standard, *JEP-TALN, Workshop on Arabic Language Processing*, Fez, 259-264.

Frank, Anette, 2003, "Projecting LFG F-Structures from Chunks" *LFG 2003,* Saratoga Springs, New York, pp. 217-237.

Gazdar, Gerald, Ewan Klein, Geoffrey Pullum and Ivan Sag *Generalized Phrase structure grammar*, Harvard University Press, Cambridge MA, 1985.

Gerdes, Kim, Sylvain Kahane, 2001, "Word Order in German: A Formal Dependency Grammar Using a Topological Hierarchy", *ACL 2001*, 220-27.

Gerdes, Kim, Sylvain Kahane, 2006, L'amas verbal au cœur d'une modélisation topologique, *Linguisticae Investigationes*, 29:1, 101-114.

Gerdes, Kim & Hi-Yon Yoo, 2003, La topologie comme interface entre syntaxe et prosodie, un système de génération appliqué au grec moderne, *TALN 2003*, Batz-sur-Mer, 125-134.

Hudson Richard, 2000, "Discontinuity", *Dependency Grammars*, *T.A.L.*, 41(1): 15-56, Hermès, Paris.

Kahane, Sylvain, Alexis Nasr, Owen Rambow, 1998, "Pseudo-Projectivity: a Polynomially Parsable Non-Projective Dependency Grammar", *COLING-ACL'98*, Montreal, 646-52.

Kathol Andreas, 1995*, Linearization-based German Syntax*, PhD thesis, Ohio State University.

Kiefer, B., H.-U. Krieger, J. Carroll, and R. Malouf, 1999, "A bag of useful techniques for efficient and robust parsing". *ACL 1999*, 473–480.

Alexander Koller and Kristina Striegnitz. 2002. "Generation as dependency parsing". *ACL 2002*.

Nasr Alexis, 1995, "A formalism and a parser for Lexicalised Dependency Grammars". *4th Int. Workshop on Parsing Technologies*, State University of NY Press.

Nasr, Alexis, 2003, Factoring surface syntactic structures. In First International Conference on Meaning-Text Theory, pages 249-258, Paris.

Nishida, Kenji, Kentaro Torisawa and Jun'ichi Tsujii., 2001, "Compiling an HPSG-based grammar into more than one CFG". *PACLING 2001*. pp. 199--206.

Pollard, C. and I. Sag. (1994) *Head-Driven Phrase Structure Grammar*, Chicago: University of Chicago Press, and Stanford: CSLI Publications.

Yoo, Hiyon & Kim Gerdes, 2004, "A dependency account of Korean Word Order" *Linguistic Society of Korea 2004*, Seoul