A grammar formalism and parser for linearization-based HPSG

Michael W. Daniels and **W. Detmar Meurers**
Department of Linguistics
The Ohio State University
222 Oxley Hall
1712 Neil Avenue
Columbus, OH 43210
daniels|dm@ling.osu.edu

Abstract

Linearization-based HPSG theories are widely used for analyzing languages with relatively free constituent order. This paper introduces the Generalized ID/LP (GIDLP) grammar format, which supports a direct encoding of such theories, and discusses key aspects of a parser that makes use of the dominance, precedence, and linearization domain information explicitly encoded in this grammar format. We show that GIDLP grammars avoid the explosion in the number of rules required under a traditional phrase structure analysis of free constituent order. As a result, GIDLP grammars support more modular and compact grammar encodings and require fewer edges in parsing.

1 Introduction

Within the framework of Head-Driven Phrase Structure Grammar (HPSG), the so-called linearization-based approaches have argued that constraints on word order are best captured within domains that extend beyond the local tree. A range of analyses for languages with relatively free constituent order have been developed on this basis (see, for example, Reape, 1993; Kathol, 1995; Müller, 1999; Donohue and Sag, 1999; Bonami et al., 1999) so that it is attractive to exploit these approaches for processing languages with relatively free constituent order.

This paper introduces a grammar format that supports a direct encoding of linearization-based HPSG theories. The Generalized ID/LP (GIDLP) format explicitly encodes the dominance, precedence, and linearization domain information and thereby supports the development of efficient parsing algorithm making use of this information. We make this concrete by discussing key aspects of a parser for GIDLP grammars that integrates the word order domains and constraints into the parsing process.

2 Linearization-based HPSG

The idea of discontinuous constituency was first introduced into HPSG in a series of papers by Mike Reape (see Reape, 1993 and references therein).[1] The core idea is that word order is determined not at the level of the local tree, but at the newly introduced level of an *order domain*, which can include elements from several local trees. We interpret this in the following way: Each terminal has a corresponding order domain, and just as constituents combine to form larger constituents, so do their order domains combine to form larger order domains.

Following Reape, a daughter's order domain enters its mother's order domain in one of two ways. The first possibility, *domain union*, forms the mother's order domain by shuffling together its daughters' domains. The second option, *domain compaction*, inserts a daughter's order domain into its mother's. Compaction has two effects:

Contiguity: The terminal yield of a compacted category contains all and only the terminal yield of the nodes it dominates; there are no holes or additional strings.

LP Locality: Precedence statements only constrain the order among elements within the same compacted domain. In other words, precedence constraints cannot look into a compacted domain.

Note that these are two distinct functions of domain compaction: defining a domain as covering a contiguous stretch of terminals is in principle independent of defining a domain of elements for LP constraints to apply to. In linearization-based HPSG, domain compaction encodes both aspects.

Later work (Kathol and Pollard, 1995; Kathol, 1995; Yatabe, 1996) introduced the notion of *partial compaction*, in which only a portion of the daughter's order domain is compacted; the remaining elements are domain unioned.

[1] Apart from Reape's approach, there have been proposals for a more complete separation of word order and syntactic structure in HPSG (see, for example, Richter and Sailer, 2001 and Penn, 1999). In this paper, we focus on the majority of linearization-based HPSG approaches, which follow Reape.

3 Processing linearization-based HPSG

Formally, a theory in the HPSG architecture consists of a set of constraints on the data structures introduced in the signature; thus, word order domains and the constraints thereon can be straightforwardly expressed. On the computational side, however, most systems employ parsers to efficiently process HPSG-based grammars organized around a phrase structure backbone. Phrase structure rules encode immediate dominance (ID) and linear precedence (LP) information in local trees, so they cannot directly encode linearization-based HPSG, which posits word order domains that can extend the local trees.

The ID/LP grammar format (Gazdar et al., 1985) was introduced to separate immediate dominance from linear precedence, and several proposals have been made for direct parsing of ID/LP grammars (see, for example, Shieber, 1994). However, the domain in which word order is determined still is the local tree licensed by an ID rule, which is insufficient for a direct encoding of linearization-based HPSG.

The LSL grammar format as defined by Suhre (1999) (based on Götz and Penn, 1997) allows elements to be ordered in domains that are larger than a local tree; as a result, categories are not required to cover contiguous strings. Linear precedence constraints, however, remain restricted to local trees: elements that are linearized in a word order domain larger than their local tree cannot be constrained. The approach thus provides valuable worst-case complexity results, but it is inadequate for encoding linearization-based HPSG theories, which crucially rely on the possibility to express linear precedence constraints on the elements within a word order domain.

In sum, no grammar format is currently available that adequately supports the encoding of a processing backbone for linearization-based HPSG grammars. As a result, implementations of linearization-based HPSG grammars have taken one of two options. Some simply do not use a parser, such as the work based on ConTroll (Götz and Meurers, 1997); as a consequence, the efficiency and termination properties of parsers cannot be taken for granted in such approaches.

The other approaches use a minimal parser that can only take advantage of a small subset of the requisite constraints. Such parsers are typically limited to the general concept of resource sensitivity – every element in the input needs to be found exactly once – and the ability to require certain categories to dominate a contiguous segment of the input. Some of these approaches (Johnson, 1985; Reape, 1991) lack word order constraints altogether. Others (van Noord, 1991; Ramsay, 1999) have the grammar writer provide a combinatory predicate (such as concatenate, shuffle, or head-wrap) for each rule specifying how the string coverage of the mother is determined from the string coverages of the daughter. In either case, the task of constructing a word order domain and enforcing word order constraints in that domain is left out of the parsing algorithm; as a result, constraints on word order domains either cannot be stated or are tested in a separate clean-up phase.

4 Defining GIDLP Grammars[2]

To develop a grammar format for linearization-based HPSG, we take the syntax of ID/LP rules and augment it with a means for specifying which daughters form compacted domains. A Generalized ID/LP (GIDLP) grammar consists of four parts: a root declaration, a set of lexical entries, a set of grammar rules, and a set of global order constraints. We begin by describing the first three parts, which are reminiscent of context-free grammars (CFGs), and then address order constraints in section 4.1.

The root declaration has the form $root(S, L)$ and states the start symbol S of the grammar and any linear precedence constraints L constraining the root domain.

Lexical entries have the form $A \rightarrow t$ and link the pre-terminal A to the terminal t, just as in CFGs.

Grammar rules have the form $A \rightarrow \alpha; C$. They specify that a non-terminal A immediately dominates a list of non-terminals α in a domain where a set of order constraints C holds.

Note that in contrast to CFG rules, the order of the elements in α does not encode immediate precedence or otherwise contribute to the denotational meaning of the rule. Instead, the order can be used to generalize the head marking used in grammars for head-driven parsing (Kay, 1990; van Noord, 1991) by additionally ordering the non-head daughters.[3]

[2] Due to space limitations, we focus here on introducing the syntax of the grammar formalism and giving an example. We will also base the discussion on simple term categories; nothing hinges on this, and when using the formalism to encode linearization-based HPSG grammars, one will naturally use the feature descriptions known from HPSG as categories.

[3] By ordering the right-hand side of a rule so that those categories come first that most restrict the search space, it becomes possible to define a parsing algorithm that makes use of this information. For an example of a construction where ordering the non-head daughters is useful, consider sentences with AcI verbs like *I see him laugh*. Under the typical HPSG analy-

If the set of order constraints is empty, we obtain the simplest type of rule, exemplified in (1).

(1) $S \rightarrow NP, VP$

This rule says that an **S** may immediately dominate an **NP** and a **VP**, with no constraints on the relative ordering of **NP** and **VP**. One may precede the other, the strings they cover may be interleaved, and material dominated by a node dominating **S** can equally be interleaved.

4.1 Order Constraints

GIDLP grammars include two types of order constraints: linear precedence constraints and compaction statements.

4.1.1 Linear Precedence Constraints

Linear precedence constraints can be expressed in two contexts: on individual rules (as *rule-level* constraints) and in compaction statements (as *domain-level* constraints). Domain-level constraints can also be specified as *global* order constraints, which has the effect that they are specified for each single domain.

All precedence constraints enforce the following property: given any appropriate pair of elements in the same domain, one must completely precede the other for the resulting parse to be valid. Precedence constraints may optionally require that there be no intervening material between the two elements: this is referred to as immediate precedence. Precedence constraints are notated as follows:

- **Weak precedence:** $A < B$.
- **Immediate precedence:** $A \ll B$.

A pair of elements is considered appropriate when one element in a domain matches the symbol *A*, another matches *B*, and neither element dominates the other (it would otherwise be impossible to express an order constraint on a recursive rule).

The symbols *A* and *B* may be *descriptions* or *tokens*. A category in a domain matches a description if it is subsumed by it; a token refers to a specific category in a rule, as discussed below. A constraint involving descriptions applies to any pair of elements in any domain in which the described categories occur; it thus can also apply more than once within a given rule or domain. Tokens, on the other hand, can only occur in rule-level constraints and refer to particular RHS members of a rule. In this paper, tokens are represented by numbers referring to the subscripted indices on the RHS categories.

In (2) we see an example of a rule-level linear precedence constraint.

(2) $A \rightarrow NP_1, V_2, NP_3;\ 3 < V$

This constraint specifies that the token 3 in the rule's RHS (the second **NP**) must precede any constituents described as **V** occurring in the same domain (this includes, but is not limited to, the **V** introduced by the rule).

4.1.2 Compaction Statements

As with LP constraints, compaction statements exist as rule-level and as global order constraints; they cannot, however, occur within other compaction statements. A rule-level compaction statement has the form $\langle \alpha, A, L \rangle$, where α is a list of tokens, *A* is the category representing the compacted domain, and *L* is a list of domain-level precedence constraints. Such a statement specifies that the constituents referenced in α form a compacted domain with category *A*, inside of which the order constraints in *L* hold. As specified in section 2, a compacted domain must be contiguous (contain all and only the terminal yield of the elements in that domain), and it constitutes a local domain for LP statements.

It is because of partial compaction that the second component *A* in a compaction statement is needed. If only one constituent is compacted, the resulting domain will be of the same category; but when multiple categories are fused in partial compaction, the category of the resulting domain needs to be determined so that LP constraints can refer to it.

The rule in (3) illustrates compaction: each of the **S** categories forms its own domain. In (4) partial compaction is illustrated: the **V** and the first **NP** form a domain named **VP** to the exclusion of the second **NP**.

(3) $S \rightarrow S_1, Conj_2, S_3;$
$1 \ll 2, 2 \ll 3, \langle [1], S, \langle [] \rangle \rangle, \langle [3], S, \langle [] \rangle \rangle$

(4) $VP \rightarrow V_1, NP_2, NP_3;\ \langle [1, 2], VP, \langle [] \rangle \rangle$

One will often compact only a single category without adding domain-specific LP constraints, so we introduce the abbreviatory notation of writing such a compacted category in square brackets. In this way (3) can be written as (5).

(5) $S \rightarrow [S_1], Conj_2, [S_3];\ 1 \ll 2, 2 \ll 3$

sis (Pollard and Sag, 1994), *see* combines in a ternary structure with *him* and *laugh*. Note that the constituent that is appropriate in the place occupied by *him* here can only be determined once one has looked at the other complement, *laugh*, from which it is raised.

A final abbreviatory device is useful when the entire RHS of a rule forms a single domain, which Suhre (1999) refers to as "left isolation". This is denoted by using the token 0 in the compaction statement if linear precedence constraints are attached, or by enclosing the LHS category in square brackets, otherwise. (See rules (13d) and (13j) in section 6 for an example of this notation.)

The formalism also supports *global compaction statements*. A global compaction statement has the form $\langle A, L\rangle$, where A is a description specifying a category that always forms a compacted domain, and L is a list of domain-level precedence constraints applying to the compacted domain.

4.2 Examples

We start with an example illustrating how a CFG rule is encoded in GIDLP format. A CFG rule encodes the fact that each element of the RHS immediately precedes the next, and that the mother category dominates a contiguous string. The context-free rule in (6) is therefore equivalent to the GIDLP rule shown in (7).

(6) $S \rightarrow Nom\ V\ Acc$

(7) $[S] \rightarrow V_1, Nom_2, Acc_3;\ 2 \ll 1, 1 \ll 3$

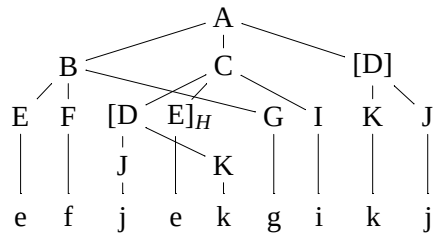
In (8) we see a more interesting example of a GIDLP grammar.

(8) a) root(A, [])
b) $A \rightarrow B_1, C_2, [D_3];\ 2 < 3$
c) $B \rightarrow F_1, G_2, E_3$
d) $C \rightarrow E_1, D_2, I_3;\ \langle[1,2], H, \langle[]\rangle\rangle$
e) $D \rightarrow J_1, K_2$
f) Lexical entries: $E \rightarrow e, \ldots$
g) $E < F$

(8a) is the root declaration, stating that an input string must parse as an **A**; the empty list shows that no LP constraints are specifically declared for this domain. (8b) is a grammar rule stating that an **A** may immediately dominate a **B**, a **C**, and a **D**; it further states that the second constituent must precede the third and that the third is a compacted domain. (8c) gives a rule for **B**: it dominates an **F**, a **G**, and an **E**, in no particular order. (8d) is the rule for **C**, illustrating partial compaction: its first two constituents jointly form a compacted domain, which is given the name **H**. (8e) gives the rule for **D** and (8f) specifies the lexical entries (here, the preterminals just rewrite to the respective lowercase terminal). Finally, (8g) introduces a global LP constraint requiring an **E** to precede an **F** whenever both elements occur in the same domain.

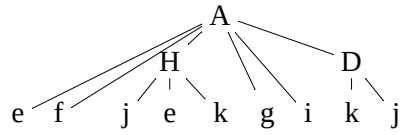
Now consider licensing the string **efjekgikj** with the above grammar. The parse tree, recording which rules are applied, is shown in (9). Given that the domains in which word order is determined can be larger than the local trees, we see crossing branches where discontinuous constituents are licensed.

(9)



To obtain a representation in which the order domains are represented as local trees again, we can draw a tree with the compacted domains forming the nodes, as shown in (10).

(10)



There are three non-lexical compacted domains in the tree in (9): the root **A**, the compacted **D**, and the partial compaction of **D** and **E** forming the domain **H** within **C**. In each domain, the global LP constraint **E** < **F** must be obeyed. Note that the string is licensed by this grammar even though the second occurrence of **E** does not precede the **F**. This **E** is inside a compacted domain and therefore is not in the same domain as the **F**, so that the LP constraint does not apply to those two elements. This illustrates the property of LP locality: domain compaction acts as a 'barrier' to LP application.

The second aspect of domain compaction, contiguity, is also illustrated by the example, in connection with the difference between total and partial compaction. The compaction of **D** specified in (8b) requires that the material it dominates be a contiguous segment of the input. In contrast, the partial compaction of the first two RHS categories in rule (8d) requires that the material dominated by **D** and **E**, taken together, be a continuous segment. This allows the second **e** to occur between the two categories dominated by **D**.

Finally, the two tree representations above illustrate the separation of the combinatorial potential of rules (9) from the flatter word order domains (10) that the GIDLP format achieves. It would, of course, be possible to write phrase structure rules that license the word order domain tree in (10) directly, but this would amount to replacing a set of

general rules with a much greater number of flatter rules corresponding to the set of all possible ways in which the original rules could be combined without introducing domain compaction. Müller (2004) discusses the combinatorial explosion of rules that results for an analysis of German if one wants to flatten the trees in this way. If recursive rules such as adjunction are included – which is necessary since adjuncts and complements can be freely intermixed in the German Mittelfeld – such flattening will not even lead to a finite number of rules. We will return to this issue in section 6.

5 A Parsing Algorithm for GIDLP

We have developed a GIDLP parser based on Earley's algorithm for context-free parsing (Earley, 1970). In Earley's original algorithm, each edge encodes the interval of the input string it covers. With discontinuous constituents, however, that is no longer an option. In the spirit of Johnson (1985) and Reape (1991), and following Ramsay (1999), we represent edge coverage with bitvectors, stored as integers. For instance, 00101 represents an edge covering words one and three of a five-word sentence.[4]

Our parsing algorithm begins by seeding the chart with passive edges corresponding to each word in the input and then predicting a compacted instance of the start symbol covering the entire input; each final completion of this edge will correspond to a successful parse.

As with Earley's algorithm, the bulk of the work performed by the algorithm is borne by two steps, *prediction* and *completion*. Unlike the context-free case, however, it is not possible to anchor these steps to string positions, proceeding from left to right. The strategy for prediction used by Suhre (1999) for his LSL parser is to predict every rule at every position. While this strategy ensures that no possibility is overlooked, it fails to integrate and use the information provided by the word order constraints attached to the rules – in other words, the parser receives no top-down guidance. Some of the edges generated by prediction therefore fall prey to the word order constraints later, in a generate-and-test fashion. This need not be the case. Once one daughter of an active edge has been found, the other daughters should only be predicted to occur in string positions that are compatible with the word order constraints of the active edge. For example, consider the edge in (11).

(11) $A \rightarrow B_1 \bullet C_2 \; ; \; 1 < 2$

[4]Note that the first word is the rightmost bit.

This notation represents the point in the parse during which the application of this rule has been predicted, and a **B** has already been located. Assuming that **B** has been found to cover the third position of a five-word string, two facts are known. From the LP constraint, **C** cannot precede **B**, and from the general principle that the RHS of a rule forms a partition of its LHS, **C** cannot overlap **B**. Thus **C** cannot cover positions one, two, or three.

5.1 Compiling LP Constraints into Bitmasks

We can now discuss the integration of GIDLP word order constraints into the parsing process. A central insight of our algorithm is that the same data structure used to describe the coverage of an edge can also encode restrictions on the parser's search space. This is done by adding two bitvectors to each edge, in addition to the coverage vector: a *negative mask* (n-mask) and a *positive mask* (p-mask). Efficient bitvector operations can then be used to compute, manipulate, and test the encoded constraints.

Negative Masks The n-mask constrains the set of possible coverage vectors that could complete the edge. The 1-positions in a masking vector represent the positions that are masked out: the positions that cannot be filled when completing this edge. The 0-positions in the negative mask represent positions that may potentially be part of the edge's coverage. For the example above, the coverage vector for the edge is 00100 since only the third word **B** has been found so far. Assuming no restrictions from a higher rule in the same domain, the n-mask for **C** is 00111, encoding the fact that the final coverage vector of the edge for **A** must be either 01000, 10000, or 11000 (that is, **C** must occupy position four, position five, or both of these positions). The negative mask in essence encodes information on where the active category cannot be found.

Positive Masks The p-mask encodes information about the positions the active category **must** occupy. This knowledge arises from immediate precedence constraints. For example, consider the edge in (12).

(12) $D \rightarrow E_1 \bullet F_2 \; ; \; 1 \ll 2$

If **E** occupies position one, then **F** must at least occupy position two; the second position in the positive mask would therefore be occupied.

Thus in the prediction step, the parser considers each rule in the grammar that provides the symbol being predicted, and for each rule, it generates bitmasks for the new edge, taking both rule-level and domain-level order constraints into account. The resulting masks are checked to ensure that there is

enough space in the resulting mask for the minimum number of categories required by the rule.[5]

Then, as part of each completion step, the parser must update the LP constraints of the active edge with the new information provided by the passive edge. As edges are initially constructed from grammar rules, all order constraints are initially expressed in terms of either descriptions or tokens. As the parse proceeds, these constraints are updated in terms of the actual locations where matching constituents have been found. For example, a constraint like $1 < 2$ (where 1 and 2 are tokens) can be updated with the information that the constituent corresponding to token 1 has been found as the first word, i.e. as position 00001.

In summary, compiling LP constraints into bitmasks in this way allows the LP constraints to be integrated directly into the parser at a fundamental level. Instead of weeding out inappropriate parses in a cleanup phase, LP constraints in this parser can immediately block an edge from being added to the chart.

6 Evaluation

As discussed at the end of section 4.2, it is possible to take a GIDLP grammar and write out the discontinuity. All non-domain introducing rules must be folded into the domain-introducing rules, and then each permitted permutation of a RHS must become a context-free rule on its own – generally, at the cost of a factorial increase in the number of rules.

This construction indicates the basis for a preliminary assessment of the GIDLP formalism and its parser. The grammar in (13) recognizes a very small fragment of German, focusing on the free word order of arguments and adjuncts in the so-called *Mittelfeld* that occurs to the right of either the finite verb in yes-no questions or the complementizer in complementized sentences.[6]

(13) a) root(s, [])

b) $s \rightarrow s(cmp)_1$

c) $s \rightarrow s(que)_1$

d) $s(cmp) \rightarrow cmp_1, clause_2$; $\langle [0], s(cmp), \langle cmp < \_, \_ < v(\_) \rangle \rangle$

e) $s(que) \rightarrow clause_1$; $\langle [0], s(que), \langle v(\_) < \_ \rangle \rangle$

f) $clause \rightarrow np(n)_1, vp_2$

g) $vp \rightarrow v(ditr)_1, np(a)_2, np(d)_3$

h) $vp \rightarrow adv_1, vp_2$

i) $vp \rightarrow v(cmp)_1, s(cmp)_2$

j) $[np(Case)] \rightarrow det(Case)_1, n(Case)_2$; $1 \ll 2$

k)	v(ditr) → *gab*	q)	v(cmp) → *denkt*
l)	comp → *dass*	r)	det(nom) → *der*
m)	det(dat) → *der*	s)	det(acc) → *das*
n)	n(nom) → *Mann*	t)	n(dat) → *Frau*
o)	n(acc) → *Buch*	u)	adv → *gestern*
p)	adv → dort		

The basic idea of this grammar is that domain compaction only occurs at the top of the head path, after all complements and adjuncts have been found. When the grammar is converted into a CFG, the effect of the larger domain can only be mimicked by eliminating the clause and vp constituents altogether.

As a result, while this GIDLP grammar has 10 syntactic rules, the corresponding flattened CFG (allowing for a maximum of two adverbs) has 201 rules. In an experiment, the four sample sentences in (14)[7] were parsed with both our prototype GIDLP parser (using the GIDLP grammar) as well as a vanilla Earley CFG parser (using the CFG); the results are shown in (15).

(14) a) *Gab der Mann der Frau das Buch?*

b) *dass das Buch der Mann der Frau gab.*

c) *dass das Buch gestern der Mann dort der Frau gab.*

d) *Denkt der Mann dass das Buch gestern der Mann dort der Frau gab?*

(15)

	Active Edges		Passive Edges	
Sentence	GIDLP	CFG	GIDLP	CFG
a)	18	327	16	15
b)	27	338	18	16
c)	46	345	27	27
d)	75	456	36	24

Averaging over the four sentences, the GIDLP grammar requires 89% fewer active edges. It also generates additional passive edges corresponding to the extra non-terminals vp and clause. It is important to keep in mind that the GIDLP grammar is more general than the CFG: in order to obtain a finite number of CFG rules, we had to limit the number of adverbs. When using a grammar capable of

[5] This optimization only applies to epsilon-free grammars. Further work in this regard can involve determining the minumum and maximum yields of each category; some optimizations involving this information can be found in (Haji-Abdolhosseini and Penn, 2003).

[6] The symbol _ is used to denote the set of all categories.

[7] The grammar and example sentences are intended as a formal illustration, not a linguistic theory; because of this and space limitations, we have not provided glosses.

handling longer sentences with more adverbs, the number of CFG rules (and active edges, as a consequence) increases factorially.

Timings have not been included in (15); it is generally the case that the GIDLP parser/grammar combination was slower than the CFG/Earley parser. This is an artifact of the use of atomic categories, however. For the large feature structures used as categories in HPSG, we expect the larger numbers of edges encountered while parsing with the CFG to have a greater impact on parsing time, to the point where the GIDLP grammar/parser is faster.

7 Summary

In this abstract, we have introduced a grammar format that can be used as a processing backbone for linearization-based HPSG grammars that supports the specification of discontinuous constituents and word order constraints on domains that extend beyond the local tree. We have presented a prototype parser for this format illustrating the use of order constraint compilation techniques to improve efficiency. Future work will concentrate on additional techniques for optimized parsing as well as the application of the parser to feature-based grammars. We hope that the GIDLP grammar format will encourage research on such optimizations in general, in support of efficient processing of relatively free constituent order phenomena using linearization-based HPSG.

References

Olivier Bonami, Danièle Godard, and Jean-Marie Marandin. 1999. Constituency and word order in French subject inversion. In Gosse Bouma et al., editor, *Constraints and Resources in Natural Language Syntax and Semantics*. CSLI.

Cathryn Donohue and Ivan A. Sag. 1999. Domains in Warlpiri. In *Abstracts of the Sixth Int. Conference on HPSG*, pages 101–106, Edinburgh.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2).

Gerald Gazdar, Ewan Klein, Geoffrey K. Pullum, and Ivan A. Sag. 1985. *Generalized Phrase Structure Grammar*. Harvard University Press.

Thilo Götz and W. Detmar Meurers. 1997. The ConTroll system as large grammar development platform. In *Proceedings of the EACL Workshop "Computational Environments for Grammar Development and Linguistic Engineering"*, Madrid.

Thilo Götz and Gerald Penn. 1997. A proposed linear specification language. Volume 134 in Arbeitspapiere des SFB 340, Tübingen.

Mohammad Haji-Abdolhosseini and Gerald Penn. 2003. ALE reference manual. Univ. Toronto.

Mark Johnson. 1985. Parsing with discontinuous constituents. In *Proceedings of ACL*, Chicago.

Andreas Kathol and Carl Pollard. 1995. Extraposition via complex domain formation. In *Proceedings of ACL*, pages 174–180, Boston.

Andreas Kathol. 1995. *Linearization-Based German Syntax*. Ph.D. thesis, Ohio State University.

Martin Kay. 1990. Head-driven parsing. In Masaru Tomita, editor, *Current Issues in Parsing Technology*. Kluwer, Dordrecht.

Stefan Müller. 1999. *Deutsche Syntax deklarativ*. Niemeyer, Tübingen.

Stefan Müller. 2004. Continuous or discontinuous constituents? A comparison between syntactic analyses for constituent order and their processing systems. *Research on Language and Computation*, 2(2):209–257.

Gerald Penn. 1999. Linearization and WH-extraction in HPSG: Evidence from Serbo-Croatian. In Robert D. Borsley and Adam Przepiórkowski, editors, *Slavic in HPSG*. CSLI.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago.

Allan M. Ramsay. 1999. Direct parsing with discontinuous phrases. *Natural Language Engineering*, 5(3):271–300.

Mike Reape. 1991. Parsing bounded discontinuous constituents: Generalisations of some common algorithms. In Mike Reape, editor, *Word Order in Germanic and Parsing.* DYANA R1.1.C.

Mike Reape. 1993. *A Formal Theory of Word Order: A Case Study in West Germanic*. Ph.D. thesis, University of Edinburgh.

Frank Richter and Manfred Sailer. 2001. On the left periphery of German finite sentences. In W. Detmar Meurers and Tibor Kiss, editors, *Constraint-Based Approaches to Germanic Syntax*. CSLI.

Stuart M. Shieber. 1984. Direct parsing of ID/LP grammars. *Linguistics & Philosophy*, 7:135–154.

Oliver Suhre. 1999. Computational aspects of a grammar formalism for languages with freer word order. Diplomarbeit. (= Volume 154 in Arbeitspapiere des SFB 340, 2000).

Gertjan van Noord. 1991. Head corner parsing for discontinuous constituency. In *ACL Proceedings*.

Shuichi Yatabe. 1996. Long-distance scrambling via partial compaction. In Masatoshi Koizumi, Masayuki Oishi, and Uli Sauerland, editors, *Formal Approaches to Japanese Linguistics 2*. MITWPL.