Appeared in Shuly Wintner (Ed.): Proceedings of NLULP'02: The 7th International Workshop on Natural Language Understanding and Logic Programming. Copenhagen. 2002. pp. 49–68.

Improving the Efficiency of Parsing with Discontinuous Constituents*

Mike Daniels and W. Detmar Meurers

Department of Linguistics, The Ohio State University
222 Oxley Hall, 1712 Neil Ave., Columbus, OH 43210
{daniels,dm}@ling.osu.edu

Abstract. We discuss a generalization of Earley's algorithm to grammars licensing discontinuous constituents of the kind proposed by the so-called linearization approaches in Head-Driven Phrase Structure Grammar. We show how to replace the standard indexing on the string position by bitmasks that act as constraints over possible coverage bitvectors. This improves efficiency of edge access and reduces the number of edges by constraining prediction to those grammar rules which are compatible with known word order properties. The resulting parsing algorithm does not have to process the righthand side categories in the order in which they cover the string, and so a head-driven strategy can be obtained simply by reordering the righthand side categories of the rules. The resulting strategy generalizes head-driven parsing in that it also permits the ordering of non-head categories.

1 Introduction

A prominent tradition within the framework of Head-Driven Phrase Structure Grammar (HPSG, Pollard and Sag 1994) has argued on linguistic grounds for analyses which license so-called discontinuous constituents (Reape 1993; Kathol 1995; Richter and Sailer 2001; Müller 1999a; Penn 1999; Donohue and Sag 1999; Bonami et al. 1999).[1] More recently, Müller (2002) argues that HPSG grammars for German which license discontinuous constituents should also be preferred on computational grounds. The idea underlying Müller's argument is that in order

* We would like to thank Chris Brew, Wesley Davidson, Paul Davis, Stefan Müller, Gerald Penn, and the Clippers for valuable discussion and the three anonymous reviewers for their helpful comments. The work was supported by an OSU College of Humanities Seed Grant and the German Federal Ministry for Research Technology (BMBF) as part of the MiLCA consortium (http://milca.sfs.uni-tuebingen.de/A4/).

[1] The concatenation of strings underlying ordinary phrase structure grammars has also been rejected by researchers in other linguistic frameworks, including Dependency Grammar (Bröker 1998; Plátek et al. 2001), Tree Adjoining grammar (Kroch and Joshi 1987; Rambow and Joshi 1994), Categorial Grammar (Dowty 1996; Hepple 1994; Morrill 1995), and those positing tangled trees (McCawley 1982; Huck 1985; Ojeda 1987; Blevins 1990). Interestingly, discontinuous constituents are also assumed in the two German treebanks (Skut et al. 1997; Hinrichs et al. 2000).

to license the many word order possibilities (as, for instance, are found in the so-called *Mittelfeld*), a large number of rules or equivalent specifications are needed, resulting in a large number of passive edges. Since there is no need to distinguish these different word orders in terms of the resulting semantics, positing different rules for each order results only in wasted computational effort.[2]

While Müller's argumentation seems sensible on a conceptual level, he also concedes that the parsing technology that has been developed to license discontinuous constituents (Johnson 1985; Reape 1991; van Noord 1991; Covington 1990, 1992; Müller 1996) is far less efficient than the standard parsing algorithm for context-free grammars (Earley 1970). Thus the cost of processing such kinds of grammars in practice seems to outweigh the significant reduction in passive edges that might theoretically result from the licensure of discontinuous constituents. The reason for this inefficiency is that while the parsing algorithms for context-free grammars index edges by their corresponding string position in the terminal yield, there is no such direct correspondence between edges and single string positions when parsing with discontinuous constituents.

The idea underlying this paper is that it should be possible to develop a general parsing algorithm which is as efficient as Earley's algorithm when enough word order information is available and degrades gradually in efficiency in relation to the number and kind of discontinuities permitted by a grammar. This idea is closely related to the proposal by Suhre (1999). However, while he focuses on formal language properties and provides valuable worst case complexity results, this paper focuses on the practical aspects of ensuring that the word order constraints are used for efficient lookup of edges in the chart during completion and to limit the number of rules considered for prediction. Our proposal eliminates instances of the generate-and-test paradigm through improved indexing of edges using two kinds of bitmasks encoding word order constraints and makes use of efficient bitvector operations. The bitmasks can be viewed as a compiled form of the word order constraints which allow the parsing algorithm to check both dominance and word order relations in a tightly integrated way.

The generalization of Earley's algorithm we propose also makes it possible to process the daughters in the order in which they provide information that can guide processing. As such, it extends the notion of a head-driven algorithm (Kay 1990; van Noord 1991) by additionally ordering the non-head daughters.

2 A format for linearization grammars

In an influential series of papers, Reape (1994, 1996) introduced into HPSG the idea of an *order domain* which can potentially span multiple local trees. Ordering across local trees becomes possible by unioning the order domains of

[2] It has been argued that such different word orders correspond to (subtle) semantic differences (see, for instance, Lenerz 2001). However, until a theory of these differences has been worked out, the only option is to license the indistinguishable word order variations as instances of the same semantic form. For most computational purposes this is also likely to be sufficient in general.

the daughters in a local tree. Daughters not unioned are sometimes referred to as *isolated* or *compacted* (Kathol 1995). Compaction fixes the word order in the compacted domain and inserts it as a unit into the higher domain. Linear precedence constraints therefore only apply within each compacted word order domain.

One can implement Kathol/Reape-style domains as part of the general linguistic data structure, with the general constraint language of HPSG expressing the word order requirements. A parser can be used to check that every part of the input string is actually part of the order domain of the root. Kasper et al. (1998) show how bitvectors can be used to improve the efficiency of such an approach, but they do not outsource the word order requirements and the domains they operate on to a dedicated parsing algorithm.

To make use of Reape's idea of word order domains in a way that allows the parser to also take word order constraints into account, constraints on immediate dominance and those on word order need to be separated from other linguistic constraints. The constraints on immediate dominance can be represented by standard ID rules, but there is less consensus in the literature on an appropriate language for word order constraints. Following Suhre (1999), we use a subset of the linear specification language (LSL) proposed by Götz and Penn (1997) to serve this purpose.

We assume *grammar rules* of the form $\mathbf{A} \rightarrow \boldsymbol{\alpha}$ **;** $\mathbf{L}$ expressing that the non-terminal A immediately dominates the non-terminals in the list α. In contrast to phrase structure rules, the order of the non-terminals in α is irrelevant for the interpretation of the rule; it only determines the order of processing as described in section 3. L is a set of word order constraints, each of which has one of the following three forms:

- **(Weak) precedence: $\mathbf{A} < \mathbf{B}$.** The rightmost terminal dominated by A occurs somewhere to the left of the leftmost terminal dominated by B.
- **Immediate precedence: $\mathbf{A} \ll \mathbf{B}$.** The rightmost terminal dominated by A occurs immediately to the left of the leftmost terminal dominated by B.
- **Isolation: [A].** A dominates an uninterrupted sequence of terminals in the terminal yield.

In addition to the grammar rules, the grammar includes *lexical entries* of the form $\mathbf{A} \rightarrow \mathbf{t}$ linking the *preterminal* A to the *terminal* t.

The notion of isolation deserves some extra attention here. First, while every non-terminal is associated with its possibly-discontinuous coverage of string positions, there are two special kinds of non-terminals: the complete phrase is by its nature an isolated and therefore continuous domain, and each terminal is also an isolated domain.[3]

[3] This direct correspondence between lexical entities and atomic domains means that our formalism cannot directly encode the idea of Kathol (1995, sec. 7.4) that the lexical entry of a particle verb in German contributes two independent domain elements, or the proposal of Richter and Sailer (2001) to insert the string of the topicalized constituent as part of the string covered by the lexical entry of the finite verb.

Second, it is important to realize that we interpret isolation statements as expressing two distinct notions: a) where the string contributed by a preterminal can surface; and b) which elements in what domains the precedence statements apply to. Aspect (a) expresses the fact that the terminal yield of an isolated nonterminal contains all and only the terminal yield of the nodes it dominates: there are no holes or additional strings. Aspect (b) encodes the fact that precedence statements constraint the order between all isolated domains within an isolated domain. This means that no precedence constraint can apply to an element that is inside an isolated domain.

Essentially this grammar formalism reintroduces into HPSG a relational backbone encoding dominance and precedence information. The idea is closely related to the ID/LP format of GPSG (Gazdar et al. 1985) in that both express dominance and precedence independently and separately from other grammatical constraints. However, it extends the ID/LP format, which does not license discontinuous constituents, by allowing the specification of domains which are larger than the local tree. We will therefore refer to grammars expressed in this format as Generalized ID/LP (GIDLP) grammars.[4] Our work can therefore be conceptualized as extending the tradition of direct processing regimes for ID/LP grammars from Shieber (1984) to Morawietz (1995), and references cited therein.

Finally, before turning to algorithmic issues, it is useful to note that every context-free grammar can be encoded in this GIDLP format. Take, for example, the context-free rule $S \rightarrow NP_N\ V\ NP_A$. This rule encodes the immediate dominance and precedence relations expressed by the GIDLP rule $S \rightarrow V\ NP_N\ NP_A$; $\{NP_N \ll V, V \ll NP_A, [V], [NP_A], [NP_N]\}$.

3 Earley's algorithm and edge coverage

To highlight those places where parsing with GIDLP grammars differs from context-free parsing, we start with a general characterization of Earley's algorithm for parsing context-free grammars.

Earley's algorithm is driven by two operations, *prediction* and *completion*. There are two kinds of edges these operations apply to: *passive edges*, which map found categories to the string they cover, and *active edges*, which represent partially-found categories. To predict a rule is to insert an active edge into the chart representing the hypothesis that this rule can be applied at a given position. To complete two edges, an active edge with a passive one, is to recognize that the passive edge provides a category (the *active element*) that the active edge is looking for and to add a new edge to the chart representing the combined coverage of the two edges.

The algorithm is realized with an *active chart*, a container that itself implements some of the parsing logic. That is, when the chart receives an active edge, it completes that edge with each compatible passive edge already in the chart

[4] Suhre (1999) refers to this grammar class as LSL grammars, which we do not follow here since it incorporates only a subset of the LSL of Götz and Penn (1997), and we want to emphasize the close relationship to ID/LP grammars.

and then predicts the application of all rules whose lefthand side is the active element of that edge. When a passive edge is added to the chart, it is used to complete every compatible active edge. When an attempt is made to add an edge that would duplicate one already in the chart, no action occurs.

We start the algorithm by seeding the active chart with the lexical edges: where x is a word in the input and A is a preterminal for x, we add a passive edge to the chart representing an A found at x's location. We then predict the root symbol of the grammar. Once the active chart has finished responding to this prediction and its consequences, every edge in the chart spanning the entire input string whose lefthand side is the root symbol indicates a successful recognition.

Let us mention in passing that the way it is presented here, the algorithm differs in one respect from Earley's original proposal: We start out by seeding the chart with all input words, whereas Earley interleaves scanning with prediction and completion. As a result, we have to do completion whenever an edge is entered, regardless of whether the entered edge is active or passive.[5] This is done to strengthen the bottom-up component, which is important considering the overall goal of parsing linearization-based HPSG grammars, where much of the information guiding parsing originates in the lexicon. Together with the ability (discussed in section 4.2) to determine the order in which the righthand side categories of a rule are predicted, this allows the grammar writer to, for example, specify a head-corner processing strategy.

In the abstract form presented above, the algorithm can be applied to both context-free and GIDLP grammars. To turn it into a concrete algorithm, we need to address both how the coverage of an edge is encoded as well as how this affects finding compatible edges for completion.

For the ordinary context-free grammar case, the answers were provided by Earley (1970). Edges have the form ${}_i[A \rightarrow \alpha \bullet_j \beta]$, which indicates a parse state in which the string from i to j is covered by α and we will have found an A if β is found immediately following j. In a *passive edge*, β is empty. An *active edge*, on the other hand, has a nonempty β, the initial element of which is the *active element*, with j the *active position*. Newly-predicted edges are of the form ${}_i[A \rightarrow \bullet_i \beta]$, where i is the active position of the edge triggering prediction. A passive edge ${}_k[C \rightarrow \gamma \bullet_l]$ is compatible with an active edge ${}_i[A \rightarrow \alpha \bullet_j B\beta]$ when $j = k$ and $C = B$; the new edge covers the string from i to l.

For GIDLP grammars, our discussion of edge coverage and edge compatibility must start by describing the data structure used to encode the coverage of an edge in light of the possibility of discontinuous constituents.

3.1 Efficient edge coverage encoding

The single interval (formed by i and j) that was used to encode the coverage of the edges in the context-free case is not sufficient to model edge coverage in a

[5] We also do completion before prediction, which in the revised setup reduces the number of attempts to add redundant edges to the chart.

grammar that licenses discontinuous constituents, as each edge may potentially be covered by a discontinuous subset of the string. Johnson (1985) showed that this issue can be addressed by generalizing from single intervals to *lists of intervals*: for example, $[[1,3],[5]]$ represents an edge that covers the first, second, third, and fifth words of the input. This representation permits constant-time checking for isolation: if the cardinality of the interval list is one, the edge is isolated. On the other hand, most of the other operations needed to retrieve and use edges in parsing, such as checking for overlap and computing the union of two interval lists, are linear in the length of the input string.

As Johnson (1985) pointed out, these interval lists are descriptionally equivalent to *bitvectors*: arrays of bits where element n corresponds to the inclusion of word n of the input. Since Reape (1991, 55ff.), many researchers have used bitvectors encoded as lists of zeroes and ones to represent potentially-discontinuous edge coverage; yet this representation also requires linear operations for most if not all bitvector manipulations.

As known from other applications, bitvectors can be encoded as integers by representing them as binary numbers, with the least-significant bit of the number corresponding to the leftmost word in the input string.[6] Based on this representation we have determined ways to compute all necessary bitvector operations shown in the table below in constant time.[7]

Function	Description	Defined as
$\textsc{overlap}(x,y)$	Position occupied in both x and y?	$\textsc{and}(x,y) \neq 0$
$\textsc{combine}(x,y)$	Union of x and y.	$\textsc{or}(x,y)$
$\textsc{singleton}(p)$	BV in which only p is occupied.	2^p
$\textsc{lbound}(x)$	Least-significant occupied bit in x.	$\textsc{rbound}(\textsc{xor}(x, x-1))$
$\textsc{rbound}(x)$	Most-significant occupied bit in x.	$\lfloor \log_2(x) \rfloor$
$\textsc{prefix}(p)$	BV covering all positions $\leq p$.	$2^{p+1}-1$
$\textsc{suffix}(p)$	BV covering all positions $\geq p$.	$\textsc{not}(2^x - 1)$
$\textsc{precede}(x,y)$	Does x completely precede y?	$x < y$
$\textsc{iprecede}(x,y)$	Does x immediately precede y?	$\textsc{rbound}(x) = \textsc{lbound}(y) + 1$
$\textsc{isolated}(x)$	Does x form a continuous unit?	$x = \textsc{and}(\textsc{prefix}(\textsc{rbound}(x)), \textsc{suffix}(\textsc{lbound}(x)))$

The use of bitvectors to encode edge coverage leads naturally to the use of bitvectors as coverage constraints (*bitmasks*) which we turn to in section 4.2.

[6] For example, Davis (2002) mentions the use of integer representations of bit-vectors in the context of machine translation, and some of our inspiration for the bitvector computations below stems from bitboard-based computer chess discussions on `rec.games.chess`. In the linearization parsing literature, Ramsay (1999) seems to be the only one to explore this possibility, giving definitions of OVERLAP and COMBINE and a more complex way for computing ISOLATED.

[7] In the table, BV abbreviates bitvector, x and y are bitvectors, p is a position index, and AND, OR, XOR, and NOT are the ordinary bitwise operators. We refer to a position set to 1 as *occupied* instead of the more common term *active*, which we avoid here in order to prevent confusion with the use of active in active category and active edge.

4 A parser for GIDLP grammars

Now that the general concepts have been introduced, we can discuss our parsing algorithm as an instance of the general characterization of the Earley algorithm provided in section 3. For space and presentation reasons we will not go through the entire algorithm as implemented in Prolog, but instead focus on the data structures and predicates particularly relevant from the perspective of efficient processing.

4.1 The Chart

The edges in the chart of our GIDLP parser are of the form `edge(Nr, Covers, NMask, PMask, LHS, Rest, WOCs)`. Here, `Nr` is the edge index, `Covers` is the coverage vector, `LHS` the lefthand category of the rule, `Rest` the list corresponding to the as-yet-unfound righthand categories of the rule, and `WOCs` the list of word order constraints. The remaining components are the negative masking vector `NMask` and the positive masking vector `PMask`. We turn to their meaning and use in the next section.

During parsing we often need to consider all edges in the chart that satisfy a given condition. For example, when a passive edge triggers completion, we need to find all non-overlapping active edges seeking the just-completed category. In completion triggered by an active edge, we need to find all non-overlapping passive edges that can provide the active element's category. To avoid generate-and-test in these lookups, we define parallel indices into the chart. Relying on Prolog's first-argument indexing, we define `vecchart(Mask, Nr)`, `rhschart(ActElem, Nr)`, and `lhschart(LHS, Nr)` as indices into the `edge/7` predicates representing the chart.

The use of such indexing is standard technology, and we turn now to a more interesting and novel aspect of our approach: the use of bitmasks to compile word order constraints for use in edge and rule access.

4.2 Prediction

The strategy for prediction used by Suhre (1999) 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. 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 a daughter of an active edge has been found, the other daughters should only be predicted to occur in string positions which are compatible with the word order constraints of the active edge. For example, consider the edge $A \rightarrow B \bullet C$; { $B < C$ }. Assuming B has been found to cover the third position of a five-word string, then C cannot cover positions one, two, or three.

To implement this intuition, every edge in the chart contains an additional bitvector (alongside the coverage vector): a *negative masking vector*. This mask

constrains the set of possible coverage vectors which could complete the edge. The ones in a masking vector represent the positions that are masked out: the positions that cannot be filled when completing this edge. The zeroes 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,[8] since only the third word (the B) has been found so far. Its masking vector is 00111. This encodes the fact that the final coverage vector of the edge must be either 01000, 10000, or 11000 (that is, C must occupy position four, position five, or both positions). The negative mask therefore encodes information on where the active category cannot be found.

We also have a *positive masking vector* to encode information about the positions the active category must occupy. This knowledge arises from two sources. First, immediate precedence constraints provide a fruitful source of positive information. For example, if in an edge D $\rightarrow$ E $\bullet$ F ; {E $\ll$ F}, we know that E occupies position one, we can conclude that F at least must occupy position two; the second position in the positive mask should therefore be occupied. Second, if we know from the prediction history of an edge and the word order constraints on that edge that the active category must occur at the left edge of the space covered by the edge, we can set the bit in the positive mask corresponding to the least-significant unoccupied bit.

Having introduced the use of the coverage and masking vectors, let us look at how prediction is implemented:

```
% predict(+RHS, +Coverage, +WOCs)
predict([],_,_).
predict([A:Id|_], Coverage, WOCs) :-
  ( rule(A, Alpha, AWOCs),
    calculate_masks(WOCs, Id, Coverage, NMask, PMask),
    check_space(NMask, Alpha),
    enter_edge(A, 0, NMask, PMask, Alpha, AWOCs),
    fail ; true ).
```

Here, `RHS` is a list of the as-yet-unfound categories, `Coverage` is the bitvector representing the edge's coverage, and `WOCs` is the list of word order constraints. The first clause represents the fact that prediction is a vacuous operation on passive edges. For active edges, on the other hand, we consider each rule in the grammar that could generate the active element.[9]

Note that we only have a single active element per edge; we do not introduce multiple dots on the righthand side as done by Suhre (1999), who essentially follows the direct ID/LP parsing tradition. Recall that the dot in Earley's original parser serves two purposes: First, it is the index to the next word of the input string that has to be found. Second, it marks the next active item to be predicted.

[8] Recall that the least-significant bit of the vector corresponds to the left-most word of the string. Thus the bitvector is iconically a mirror image of the string.

[9] Each category A in a rule has a unique identifier Id and the word order constraints are expressed in terms of these identifiers.

In our generalization of this algorithm, the first purpose is served by the coverage vector; thus the dot only has the second purpose of marking the active element. Conceptually, using a single dot is sufficient since we know that for an edge to be completed, every element on the righthand side has to be found at some point. We predict the righthand side categories in the order in which they are specified in the rule, so that the grammar writer can use the order to specify those daughters to be searched first which are most likely to cause an early failure. For example, a rule introducing a conjunction of sentences can be specified as S → Conj S1 S2 ; {S1 ≪ Conj, Conj ≪ S2, [S1], [S2]}. This causes the parser to look for the easy-to-identify conjunction before it tries to find the possibly quite complex conjunct sentences.

The call to `calculate_masks/4` in the second clause of the prediction predicate above calculates the two masks for each rule we consider. This calculation takes into account the RHS elements of the rule that have already been found as well as the word order constraints that refer to them. Each of the clauses of `calculate_masks` incorporates the effect of a particular type of word order constraint on the negative or the positive mask. For example, when predicting a component that must follow a component C already located, we generate a negative mask of PREFIX(RBOUND(C)). Every newly predicted edge thus comes with two mask vectors which provide immediate access to many of the constraints on its use in completion. Based on this encoding, one can efficiently test whether, for instance, the coverage of the active edge overlaps with the negative mask of the passive edge one wants to complete with.

On the basis of the negative mask, the predicate `check_space/2` ensures that an edge is only entered into the chart if the resulting mask has enough space for the rule being considered. For instance, a rule A → B C ; L cannot apply within a mask that has only one unoccupied position,[10] so such an edge would be blocked from entering the chart at this point. Edges that pass this check are then entered into the chart with an empty coverage vector; as usual, the chart will respond to this by applying completion and prediction to this edge.

4.3 Completion

Completion always involves an active and a passive edge. Upon entering a passive edge, we search the chart for all edges seeking the category provided by the passive edge. The completion predicate for this case looks as follows:[11]

```
% complete(+Rest, +Cat, +Vec, +WOCs, +M)
complete([], Cat, Vec, [], M) :-
  ( noverlap(Vec, Mask),
    vecchart(Mask, N),
    rhschart(Cat, N),
    edge(N, ActCat, ActVec, [Cat:Id|Cats], ActWOCs),
```

[10] When processing grammars with epsilon rules, this optimization cannot be used.

[11] When an active edge triggers completion, the process is virtually identical and will not be explicitly described here.

```
mergelp(ActWOCs, Id, Vec, NewWOCs, N, M),
NewVec is Vec \/ ActVec,
check_pmask(Cats, PMask, NewVec, PMaskOut),
enter_edge(ActCat, NewVec, NewVec, PMaskOut, Cats, NewWOCs),
fail; true ).
```

In section 4.1 we saw that the edges are indexed by their negative mask and active element, so we can efficiently retrieve only those edges (with index `N`) which both a) have a non-overlapping negative mask and b) are seeking the kind of active element (`Cat`) which the triggering passive edge has to offer.

For each edge we retrieve, we then call `mergelp/6` to merge the word order constraints of the active edge (`ActWOCs`) with the coverage of the passive edge (`Vec`). As mentioned in the previous section, this rules out completion when the negative mask of the active edge overlaps with the coverage vector of the passive edge that triggered completion. The second task which is part of the merging of word order constraints requires more computation than the overlap check (which is simple since the mask has been precomputed): we need to compute the new set of word order constraints for the edge resulting from completion. All word order constraints in a freshly predicted edge refer to (the unique indices of) the categories on the righthand side of the rule. As the parse proceeds, we can replace some of these categories with their location. For example, a constraint like A $<$ B can be combined with the information that A has been found at position p. We can eliminate an edge as soon as one of its attached constraints becomes impossible to fulfill. Each update step will take one of the following forms:

- When we find one of the categories mentioned in a (potentially immediate) precedence constraint, we update the constraint and test whether there is enough space for the other category. For example, if, given the constraint A $\ll$ B, we find a B as the first word of the string, A cannot occur within the string.
- When we find the second category of a (potentially immediate) precedence constraint or the only category of an isolation constraint, we check that the constraint actually holds; if it does, then that constraint will not appear as part of the word order constraint set of the resulting edge.

Once we have successfully merged the word order constraints, the rest of the new edge is easy to compute: the category of the edge is the category `ActCat` of the active edge, the missing righthand side is the tail `Cats` of the active edge's righthand side, and the coverage vector `NewVec` is the bitwise OR of the two edges' coverage vectors.

We then process the active edge's positive mask. If the tail `Cats` is empty (indicating the creation of a passive edge), then it must be the case that IMPLIES(`Pmask`, `NewVec`), or else completion fails. The positive mask of the new edge is either empty (if the new edge is passive) or identical to the active edge's positive mask.

Finally, the resulting edge is added to the chart and triggers another round of completion and prediction.

5 An Example

We illustrate the parsing algorithm with the following Sanskrit toy grammar:

1 | s → verb:1 nom:2 acc:3 ; {2 < 1, 3 < 1}
2 | s → verb:1 nom:2 ; {2 < 1}
3 | s → conj:2 s:1 s:3 ; {1 ≪ 2, 2 ≪ 3, [1], [3]}
4 | acc → adj:1 acc:2 ; {}
5 | नलस् → nom 'Nala' (a proper name)
6 | नगरम् → acc 'city'
7 | अगच्छत् → verb 'went'
8 | चैव → conj 'and then'
9 | अवदत् → verb 'spoke'
10 | रुचिरम् → adj 'shining'

The grammar can be summarized as follows: A sentence may consist of a verb and either one or two arguments preceding it. A sentence may also consist of a conjunction immediately between two (conjunct) sentences, each of which forms an isolated domain. Finally, accusatives may be modified by an adjective which may occur anywhere in a sentence, before or after the accusative it modifies.

We give the parse for the sentence (1), which contains the discontinuous constituent रुचिरम् नगरम् 'shining city'.[12]

(1) रुचिरम् नलस् नगरम् अगच्छत् चैव नलस् अवदत्
shining Nala city went and then Nala spoke
'Nala went to the shining city and then Nala spoke'

#	Description	E	Cover	NMask	PMask	Edge
1	SCAN रुचिरम्	1	0000001	0000001	0000000	adj, [], []
2	SCAN नलस्	2	0000010	0000010	0000000	nom, [], []
3	SCAN नगरम्	3	0000100	0000100	0000000	acc, [], []
4	SCAN अगच्छत्	4	0001000	0001000	0000000	verb, [], []
5	SCAN चैव	5	0010000	0010000	0000000	conj, [], []
6	SCAN नलस्	6	0100000	0100000	0000000	nom, [], []
7	SCAN अवदत्	7	1000000	1000000	0000000	verb, [], []
8	PRED s in 0	8	0000000	0000000	1111111	s, [verb:1, nom:2, acc:3], [c2 < c1, c3 < c1]
9	COMP 8 + 4	9	0001000	0001000	1111111	s, [nom:2, acc:3], [c2 < p8, c3 < p8]
10	COMP 9 + 2	10	0001010	0001010	1111111	s, [acc:3], [c3 < p8]
11	COMP 10 + 3 fails: pmask 1111111 ↛ 0001110.					
12	PRED acc in 10	11	0000000	1111010	0000000	acc, [adj:1, acc:2], []
13	COMP 11 + 1	12	0000001	1111011	0000000	acc, [acc:2], []
14	COMP 12 + 3	13	0000101	0000101	0000000	acc, [], []
15	COMP 10 + 13 fails: pmask 1111111 ↛ 0001111.					

[12] The example has been tokenized from रुचिरं नलो नगरमगच्छच्चैव नलो ऽवदत्.

16	PRED acc in 12 fails: no room for 2 elements in 1111011.					
17	COMP 9 + 6 fails: 0100000 $\not<$ 0001000					
18	COMP 8 + 7	14	1000000	1000000	1111111	s, [nom:2, acc:3], [c2 < p64, c3 < p64]
19	COMP 14 + 2	15	1000010	1000010	1111111	s, [acc:3], [c3 < p64]
20	COMP 15 + 13 fails: pmask 1111111 $\not\to$ 1000111.					
21	COMP 15 + 3 fails: pmask 1111111 $\not\to$ 1000110.					
22	PRED acc in 15	16	0000000	1000010	0000000	acc, [adj:1, acc:2], []
23	COMP 16 + 1	17	0000001	1000011	0000000	acc, [acc:2], []
24	COMP 17 + 3 = edge 13					
25	PRED acc in 17	18	0000000	1000011	0000000	acc, [adj:1, acc:2], []
26	COMP 14 + 6	19	1100000	1100000	1111111	s, [acc:3], [c3 < p64]
27	COMP 19 + 13 fails: pmask 1111111 $\not\to$ 1100101.					
28	COMP 19 + 3 fails: pmask 1111111 $\not\to$ 1100100.					
29	PRED acc in 19	20	0000000	1100000	0000000	acc, [adj:1, acc:2], []
30	COMP 20 + 1	21	0000001	1100001	0000000	acc, [acc:2], []
31	COMP 21 + 3 = edge 13					
32	PRED acc in 21	22	0000000	1100001	0000000	acc, [adj:1, acc:2], []
33	COMP 22 + 1 = edge 21					
34	PRED s in 0	23	0000000	0000000	1111111	s, [verb:1, nom:2], [c2 < c1]
35	COMP 23 + 4	24	0001000	0001000	1111111	s, [nom:2], [c2 < p8]
36	COMP 24 + 2 fails: pmask 1111111 $\not\to$ 0001010.					
37	COMP 24 + 6 fails: 0100000 $\not<$ 0001000					
38	COMP 23 + 7	25	1000000	1000000	1111111	s, [nom:2], [c2 < p64]
39	COMP 25 + 2 fails: pmask 1111111 $\not\to$ 1000010.					
40	COMP 25 + 6 fails: pmask 1111111 $\not\to$ 1100000.					
41	PRED s in 0	26	0000000	0000000	1111111	s, [conj:2, s:1, s:3], [c1 $\ll$ c2, c2 $\ll$ c3, [1], [3]]
42	COMP 26 + 5	27	0010000	0010000	1111111	s, [s:1, s:3], [c1 $\ll$ p16, p16 $\ll$ c3, [1], [3]]
43	PRED s in 27	28	0000000	1110000	0001000	s, [verb:1, nom:2, acc:3], [c2 < c1, c3 < c1]
44	COMP 28 + 4	29	0001000	1111000	0001000	s, [nom:2, acc:3], [c2 < p8, c3 < p8]
45	COMP 29 + 2	30	0001010	1111010	0001000	s, [acc:3], [c3 < p8]
46	COMP 30 + 13	31	0001111	0001111	0000000	s, [], []
47	COMP 27 + 31	32	0011111	0011111	1111111	s, [s:3], [p16 $\ll$ c3, [3]]
48	PRED s in 32 fails: no room for 3 elements in 0011111					
49	PRED s in 32	33	0000000	0011111	0100000	s, [verb:1, nom:2], [c2 < c1]
50	COMP 33 + 7	34	1000000	1011111	0100000	s, [nom:2], [c2 < p64]
51	COMP 34 + 6	35	1100000	1100000	0000000	s, [], []
52	COMP 32 + 35	36	1111111	1111111	0000000	s, [], []
53	COMP 27 + 35 fails: 1100000 $\not<$ 0010000					
54	PRED s in 32 fails: no room for 3 elements in 0011111					
55	COMP 30 + 3	37	0001110	0001110	0000000	s, [], []
56	COMP 27 + 37	38	0011110	0011110	1111111	s, [s:3], [p16 $\ll$ c3, [3]]
57	COMP 38 + 35 fails: pmask 1111111 $\not\to$ 1111110.					

58	PRED s in 38 fails: no room for 3 elements in 0011111					
59	PRED s in 38 = edge 33					
60	PRED s in 38 fails: no room for 3 elements in 0011111					
61	PRED acc in 30 = edge 11					
62	PRED s in 27	39	0000000	1110000	0001000	s, [verb:1, nom:2], [c2 < c1]
63	COMP 39 + 4	40	0001000	1111000	0001000	s, [nom:2], [c2 < p8]
64	COMP 40 + 2	41	0001010	0001010	0000000	s, [], []
65	COMP 27 + 41 fails: 0001010 is not contiguous					
66	PRED s in 27	42	0000000	1110000	0001000	s, [conj:2, s:1, s:3], [c1 ≪ c2, c2 ≪ c3, [1], [3]]
67	SUCCESS: 36					

As stated above in section 3, we begin the parse by seeding the chart with the lexical entries, each covering a singleton vector (lines 1–7). Then we predict the application of the first rule that can generate the root symbol of the grammar (line 8), which adds the first active edge to the chart. This addition then triggers completion. Since this edge's active category is `verb`, we look for passive edges whose lefthand side is `verb`. Here, edge 4 is the first edge in the chart for which this is the case, so edge 9 is generated by completing 8 with 4. Edge 9 is added to the chart, with an updated righthand side: `verb:1` has been removed from the list of daughters and the constraint `c2 < c1` has been updated to `c2 < p8` (representing that category 1 has been found at position 0001000).

Note that with this active chart parser, Prolog's call stack implicitly represents the parsing agenda. Once edge 9 is added to the chart, our agenda consists of completing with edge 9, predicting from edge 9, finding other passive edges to complete edge 8 with, predicting from edge 8, and finding other rules that generate the root symbol.

This sample parse illustrates many of the optimizations we have discussed in this paper: Line 11 depicts an attempt to recognize a sentence covering only part of the input (before application of the recursive rule 3 has been predicted); the positive mask is not satisfied and the attempted completion is rejected. Similarly, in line 65, an attempt to treat the second and fourth words as a conjunct fails from a lack of continuity. By line 47, one of the conjuncts has been found. Since only two positions are left uncovered, the remaining conjunct must fit in two positions. Hence the attempts to predict rules 1 and 3 from line 32 also fail (lines 48, 54). Note that in each of these cases, the effect of the optimization in question is only partially represented by the number of edges directly prevented from entering the chart. Had the potential edge described on line 11 been added to the chart, for instance, the parser would have tried to complete with the resulting edge and predict from it as well. The results of those steps would have led to more instances of prediction and completion, and so on. Each time we prevent one edge from entering the chart, we prune an entire branch of the parser's search tree.

6 Efficiency Considerations

Suhre (1999) shows that the membership problem for GIDLP grammars is NP-complete, both when considering the grammar plus the string as input (general membership problem) as well as when only the string is considered as input (fixed membership problem). It is known since Huynh (1983) that the general membership problem for unordered context-free grammars (ID/LP grammars without LP statements) is also NP-complete, so this result is not surprising. But what causes the NP-completeness of the fixed membership problem for GIDLP grammars? Suhre (1999, 61ff) demonstrates that it stems from the potential for recursive growth of discontinuities; when the parser can assume an upper bound on the number of discontinuities in any given constituent, the parsing problem becomes polynomial. Formally, this can be achieved by requiring that the number of discontinuities introduced by a recursive non-terminal is bounded by some constant.

Interestingly, a related practical proposal based on a linguistic argumentation is discussed by Müller (1999b). He proposes a continuity constraint for linearization-based HPSG which requires saturated phrasal elements (that is, maximal projections) to be continuous.[13] Müller shows that adding his continuity constraint results in a significant reduction in the number of passive edges and thereby significant improvements in parsing performance. This continuity constraint is weaker than Suhre's condition in that recursion on the $\overline{\text{X}}$ level (adjunction) is not restricted. It is, however, interesting to note in this context that a grammar incorporating the $\overline{\text{X}}$-schema will require all non-head constituents to be maximal projections. In sum, Müller's result strongly suggests that further research on linguistically-motivated continuity constraints can result in efficient parsing of those GIDLP grammars which include such constraints.

This raises the question of how the parsing algorithm that we have proposed in this paper performs when used to parse grammars incorporating linear precedence and isolation constraints (since the worst-case performance results are based on the absence of such constraints). As we mentioned earlier, the GIDLP grammars form a superset of the context-free grammars. Thus it would be desirable for a GIDLP parser to be just as efficient as a context-free parser when presented with a context-free grammar encoded in the GIDLP format.[14] To investigate this, we have tested the performance with the three types of context-free grammars discussed in Earley (1970) – those that require linear, quadratic, and cubic time for strings to be recognized. We found that when given these context-free grammars encoded as GIDLP grammars, the increase in

[13] If extraposition is handled via discontinuous constituents, a more complex constraint is required.

[14] Another interesting point of reference is the ID/LP parsing literature. Volk (1996) showed that in terms of efficiency and expressivity, it is advantageous to be able to combine ID/LP and ordinary context-free rules in one grammar. We agree with his argumentation and while we have focused on the issue of discontinuity in this paper, our algorithm can be seen to seamlessly integrate context-free and (G)ID/LP rules.

the number of edges produced by our parser when the string length is increased is comparable to Earley's original algorithm.

7 Outlook

In order to focus on the fundamental aspects of efficient parsing with discontinuous constituents, a number of orthogonal aspects were kept as simple as possible in this paper. To advance towards the general goal of efficient parsing with linearization-based HPSG grammars, the next step is to introduce global linear precedence and isolation constraints and to replace atomic categories with complex categories, encoded by typed feature structures.

Global linear precedence and isolation statements. The basic GIDLP grammar format we described in section 2 extends the ID/LP paradigm in that it does not require each local tree to cover a continuous string. The precedence constraints are attached to individual rules, though, and thus they only express order constraints on the daughters of the rule they are attached to. When a discontinuity arises, the grammar writer thus cannot constrain the order of elements which have "escaped" out of their local tree in relation to other elements introduced in another local tree. The next step towards making GIDLP a useful grammar formalism thus consists of adding word order constraints which express immediate or weak linear precedence over any pair of elements within the same order domain. For example, the statement NP $<$ VP states that the NP has to precede the VP in every domain containing both. The challenge in adding such global precedence constraints consists in obtaining and storing for each domain the minimal representation of the domain elements needed to determine whether a global precedence constraint has been violated.

Adding global isolation statements, on the other hand, is less important; since every node that should be isolated has to be introduced in some rule, the isolation constraint can be stated on that rule. We nevertheless will introduce global isolation statements as a shorthand to avoid having to express the fact on every rule in which a particular kind of element can occur that it is always isolated.

From atomic to complex categories. The move to typed feature structures brings with it a number of complications. Apart from the well-known issues generally involved in adding complex categories to a chart-parser (for example, subsumption checking or the use of a restrictor), the most interesting challenge in our context is the observation by Seiffert (1991) that word order constraints cannot always be verified when a local domain is constructed. While Seiffert addresses the issue by checking word order constraints in a second pass once the entire tree has been constructed, Morawietz (1995) shows that by keeping the relevant information around, this problem can be avoided. In a similar vein, we intend to keep the relevant information around implicitly by making use of the co-routining capabilities of SICStus Prolog.

Another important effect of the move to complex categories is that the ability of the grammar writer to specify the prediction order for the righthand side of each GIDLP rule makes it possible to ensure that they are processed in the order in which they convey information. This, for example, is relevant for subject-to-object-raising constructions (for instance, *I expect him to leave*), where the verbal complement (*to leave*) determines what can occur as the object (*him*) of the verbal head (*see*).[15]

Other frameworks. In this paper we have motivated processing with the GIDLP formalism based on the HPSG linearization tradition. But as discussed in the introduction and footnote 1, analyses involving discontinuous constituents have been argued for in a variety of frameworks and also occur in the two treebanks that have been produced for German. Given that the formalism we deal with in this paper is an extension of context-free grammars, it, for instance, makes a natural candidate for extending LFG to license c-structures with discontinuities.

8 Summary

This paper has described a number of optimizations for parsing with a formalism licensing discontinuous constituents. Using bitvectors encoded as integers to model subsets of the terminal yield, we showed how the required bitvector operations can be computed in constant time. To efficiently access edges and rules in a way that makes use of word order information, we use two kinds of bitmasks to constrain possible coverage vectors, specifying the positions that are possible, impossible, and required to be covered by an edge. The algorithm can thereby take word order constraints into account in a more interleaved fashion, restricting the search space of the parser.

In the GIDLP grammar format we define, the order of the righthand side of a grammar rule does not encode the word order of the daughters. Instead, it expresses the order in which these elements will be processed; the grammar writer can order them according to the degree in which they constrain the search space of a parse. This can be used to encode a head-driven strategy by making the heads the first righthand side elements in their rules; it also becomes possible for the grammar writer to rank the non-head daughters in a rule.

As described in the outlook, we plan to extend the GIDLP grammar format and our parser to include global word order constraints and complex categories. This brings with it the opportunity for more practical evaluation metrics. It is generally accepted, for instance, that the dominating factor in feature structure-based algorithms is the number of unifications that must be performed; such a metric is easily calculated once one has a grammar which can be used for testing. We intend to recode the linearization-based Babel grammar (Müller 1996), one of the most comprehensive grammars of German, as a GIDLP grammar and use it as a test case for the extended parser. This should allow us to substantiate (or refute) the claim that processing comprehensive linearization grammars of

[15] Thanks to Wesley Davidson for this example.

natural languages is efficient once all available word order constraints are actually used to guide processing in a well-engineered way.

Bibliography

Blevins, James (1990). Syntactic Complexity: Evidence for Discontinuity and Multidomination. Ph.D. thesis, University of Massachusetts, Amherst, MA.

Bonami, Olivier, Danièle Godard and Jean-Marie Marandin (1999). Constituency and word order in French subject inversion. In Gosse Bouma, Erhard W. Hinrichs, Geert-Jan M. Kruijff and Richard T. Oehrle (eds.), *Constraints and Resources in Natural Language Syntax and Semantics*, Stanford, CA: CSLI Publications, pp. 21–40.

Bröker, Norbert (1998). Separating Surface Order and Syntactic Relations in a Dependency Grammar. In COLING-ACL (1998), pp. 174–180.

Bunt, Harry and Arthur van Horck (eds.) (1996). *Discontinuous Constituency*. Berlin and New York, NY: Mouton de Gruyter.

COLING-ACL (1998). *Proceedings of the 17th International Conference on Computational Linguistics (COLING) and the 36th Annual meeting of the ACL (ACL)*, Montreal.

Covington, Michael A. (1990). Parsing discontinuous constituents in dependency grammar. *Computational Linguistics* 16(4), 234–236.

Covington, Michael A. (1992). A dependency parser for variable-word-order languages. In K. R. Billingsley, Hilton U. Brown III and Ed Derohanes (eds.), *Computer assisted modeling on the IBM 3090: Papers from the 1989 IBM Supercomputing Competition*, Athens, GA: Baldwin Press, vol. 2, pp. 799–845.

Davis, Paul C. (2002). Stone Soup Translation: The Linked Automata Model. Ph.D. thesis, Ohio State University, Columbus, OH.

Donohue, Cathryn and Ivan A. Sag (1999). Domains in Warlpiri. In *Abstracts of the Sixth Int. Conference on HPSG*. Edinburgh: University of Edinburgh, pp. 101–106. `http://www-csli.stanford.edu/~sag/papers/warlpiri.ps`.

Dowty, David R. (1996). Towards a Minimalist Theory of Syntactic Structure. In Bunt and van Horck (1996).

Earley, Jay (1970). An Efficient Context-Free Parsing Algorithm. *Communications of the ACM* 13(2), 94–102. Also in Grosz et al. (1986).

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

Götz, Thilo and Gerald Penn (1997). *A Proposed Linear Specification Language*. Arbeitspapiere des SFB 340. Nr. 134, Universität Tübingen. `http://www.sfs.uni-tuebingen.de/sfb/reports/berichte/134/134abs.html`.

Grosz, Barbara, Karen Sparck Jones and Bonnie Lynn Webber (eds.) (1986). *Readings in Natural Language Processing*. Los Altos, CA: Morgan Kaufmann.

Hepple, Mark (1994). Discontinuity and the Lambek Calculus. In *Proceedings of the 15th Conference on Computational Linguistics (COLING-94)*. Kyoto. `ftp://ftp.dcs.shef.ac.uk/home/hepple/papers/coling94.ps`.

Hinrichs, Erhard, Julia Bartels, Yasuhiro Kawata, Valia Kordoni and Heike Telljohann (2000). The Tübingen Treebanks for Spoken German, English, and Japanese. In Wolfgang Wahlster (ed.), *Verbmobil: Foundations of Speech-to-Speech Translation*, Berlin: Springer, pp. 552–576.

Huck, Geoffrey (1985). Exclusivity and discontinuity in phrase structure grammar. In *West Coast Conference on Formal Linguistics (WCCFL)*. Stanford University, CSLI Publications, pp. 92–98.

Huck, Geoffrey and Almerindo Ojeda (eds.) (1987). *Discontinuous Constituency*. New York: Academic Press.

Huynh, Dung T. (1983). Commutative Grammars: The Complexity of Uniform Word Problems. *Information and Control* 57(1), 21–39.

Johnson, Mark (1985). Parsing with discontinuous constituents. In *Proceedings of the 23rd Annual Meeting of the ACL*. Chicago, IL, pp. 127–132.

Kasper, Robert T., Mike Calcagno and Paul C. Davis (1998). Know When to Hold 'Em: Shuffling Deterministically in a Parser for Nonconcatenative Grammars. In COLING-ACL (1998), pp. 663–669.

Kathol, Andreas (1995). Linearization-Based German Syntax. Ph.D. thesis, Ohio State University, Columbus, OH. Revised version published by Oxford University Press, 2000.

Kay, Martin (1990). Head-Driven Parsing. In Masaru Tomita (ed.), *Current Issues in Parsing Technology*, Dordrecht: Kluwer Academic Publishers.

Kroch, Anthony S. and Aravind K. Joshi (1987). Analyzing Extraposition in a Tree Adjoining Grammar. In Huck and Ojeda (1987).

Lenerz, Jürgen (2001). Word Order Variation: Competition or Co-Operation. In Gereon Müller and Wolfgang Sternefeld (eds.), *Competition in Syntax*, Berlin and New York, NY: Mouton de Gruyter, pp. 249–281.

McCawley, James D. (1982). Parentheticals and discontinuous constituent structure. *Linguistic Inquiry* 13(1), 91–106.

Morawietz, Frank (1995). *Formalization and Parsing of Unification–Based ID/LP Grammars*. Arbeitspapiere des SFB 340. Nr. 68, Universität Tübingen. `http://www.sfs.uni-tuebingen.de/sfb/reports/berichte/68/68abs.html`.

Morrill, Glynn V. (1995). Discontinuity in categorial grammar. *Linguistics and Philosophy* 18, 175–219.

Müller, Stefan (1996). The Babel-System—An HPSG Prolog Implementation. In *Proceedings of the Fourth International Conference on the Practical Application of Prolog*. London, pp. 263–277. Revised version available from `http://www.dfki.de/~stefan/Pub/babel.html`.

Müller, Stefan (1999a). *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche*. Tübingen: Max Niemeyer Verlag.

Müller, Stefan (1999b). *Restricting Discontinuity*. Verbmobil Report 237, DFKI, Saarbrücken. Also published in the Proceedings of GLDV 99 (Frank-

furt/Main). http://www.dfki.de/~stefan/Pub/e_restricting.html.

Müller, Stefan (2002). Continuous or Discontinuous Constituents? A Comparison between Syntactic Analyses for Constituent Order and Their Processing Systems. *Language and Computation* To appear. http://www.dfki.de/~stefan/Pub/discont.html.

Ojeda, Almerindo (1987). Discontinuity, multidominances and unbounded dependency in Generalized Phrase Structure Grammar. In Huck and Ojeda (1987).

Penn, Gerald (1999). Linearization and WH-extraction in HPSG: Evidence from Serbo-Croatian. In Robert D. Borsley and Adam Przepiórkowski (eds.), *Slavic in HPSG*, Stanford, CA: CSLI Publications, pp. 149–182.

Plátek, Martin, Tomáš Holan, Vladimir Kuboň and Karel Oliva (2001). Word-Order Relaxations and Restrictions within a Dependency Grammar. In G. Satta (ed.), *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT)*. Beijing: Tsinghua University Press, pp. 237–240.

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

Rambow, Owen and Aravind Joshi (1994). 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. http://arxiv.org/abs/cmp-lg/9410007.

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

Reape, Mike (1991). Parsing Bounded Discontinuous Constituents: Generalisations of some common algorithms. In Mike Reape (ed.), *Word Order in Germanic and Parsing*, Edinburgh: Centre for Cognitive Science, University of Edinburgh, pp. 41–70.

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

Reape, Mike (1994). Domain Union and Word Order Variation in German. In John Nerbonne, Klaus Netter and Carl Pollard (eds.), *German in Head-Driven Phrase Structure Grammar*, Stanford, CA: CSLI Publications, pp. 151–197.

Reape, Mike (1996). Getting things in order. In Bunt and van Horck (1996), pp. 209–253. Published version of a Ms. from 1990.

Richter, Frank and Manfred Sailer (2001). On the Left Periphery of German Finite Sentences. In W. Detmar Meurers and Tibor Kiss (eds.), *Constraint-Based Approaches to Germanic Syntax*, Stanford, CA: CSLI Publications, pp. 257–300.

Seiffert, Roland (1991). Unification–ID/LP Grammars: Formalization and Parsing. In Otthein Herzog and Claus-Rolf Rollinger (eds.), *Text Understanding in LILOG*, Berlin: Springer Verlag, pp. 63–73.

Shieber, Stuart M. (1984). Direct Parsing of ID/LP Grammars. *Linguistics and Philosophy* 7, 135–154.

Skut, Wojciech, Brigitte Krenn, Thorsten Brants and Hans Uszkoreit (1997). An Annotation Scheme for Free Word Order Languages. In *Proceedings of the*

5th Conference on Applied Natural Language Processing (ANLP). Washington, D.C. `http://www.coli.uni-sb.de/~thorsten/publications/Skut-ea-ANLP97.ps.gz`.

Suhre, Oliver (1999). *Computational Aspects of a Grammar Formalism for Languages with Freer Word Order*. Arbeitspapiere des SFB 340. Nr. 154, Universität Tübingen. (= Diploma Thesis). `http://www.sfs.uni-tuebingen.de/sfb/reports/berichte/154/154abs.html`.

van Noord, Gertjan (1991). Head Corner Parsing for Discontinuous Constituency. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*. pp. 114–121.

Volk, Martin (1996). Parsing with ID/LP and PS rules. In *Natural Language Processing and Speech Technology. Results of the 3rd KONVENS Conference (Bielefeld)*, Berlin: Mouton de Gruyter, pp. 342–353.