Towards Uniform Processing of Constraint-based Categorial Grammars

Gertjan van Noord

Lehrstuhl für Computerlinguistik
Universität des Saarlandes
Im Stadtwald 15
D-6600 Saarbrücken 11, FRG
vannoord@coli.uni-sb.de

Abstract

A class of constraint-based categorial grammars is proposed in which the construction of both logical forms and strings is specified completely lexically. Such grammars allow the construction of a uniform algorithm for both parsing and generation. Termination of the algorithm can be guaranteed if lexical entries adhere to a constraint, that can be seen as a computationally motivated version of GB's projection principle.

1 Motivations

In constraint-based approaches to grammar the *semantic interpretation* of phrases is often defined in the lexical entries. These lexical entries specify their semantic interpretation, taking into account the semantics of the arguments they subcategorize for (specified in their *subcat list*). The grammar rules simply percolate the semantics upwards; by the selection of the arguments, this semantic formula then gets further instantiated (Moore, 1989). Hence in such approaches it can be said that all semantic formulas are 'projected from the lexicon' (Zeevat *et al.*, 1987). Such an organization of a grammar is the starting point of a class of generation algorithms that have become popular recently (Calder *et al.*, 1989; Shieber *et al.*, 1989; Shieber *et al.*, 1990). These semantic-head-driven algorithms are both geared towards the input semantic representation and the information contained in lexical entries. If the above sketched approach to semantic interpretation is followed systematically, it is possible to show that such a semantic-head-driven generation algorithm terminates (Dymetman *et al.*, 1990).

In van Noord (1991) I define a head-driven parser (based on Kay (1989)) for a class of constraint-based grammars in which the construction of strings may use more complex operations that simple context-free concatenation. Again, this algorithm is geared towards the input (string) and the information found in lexical entries. In this paper I investigate an approach where the construction of strings is defined lexically. Grammar rules simply percolate strings upwards. Such an approach seems feasible if we allow for powerful constraints to be defined. The head-corner parser knows about strings and performs operations on them; in the types of grammars defined here these operations are replaced by general constraint-solving techniques (Höhfeld and Smolka, 1988; Tuda *et al.*, 1989; Damas *et al.*, 1991). Therefore, it becomes possible to view both the head-driven generator and the head-driven parser as one and the same algorithm.

For this uniform algorithm to terminate, we generalize the constraint proposed by Dymetman *et al.* (1990) to both semantic interpretations and strings. That is, for each lexical entry we require that its string and its semantics is larger than the string and the semantics associated with each of its arguments. The following picture then emerges. The depth of a derivation tree is determined by the subcat list of the ultimate head of the tree. Furthermore, the string and the semantic representation of each of the non heads in the derivation tree is determined by the subcat list as well. A specific condition on the relation between elements in the subcat list and their semantics and string representation ensures termination. This condition on lexical entries can be seen as a lexicalized and computationally motivated version of GB's projection principle.

Word-order domains. The string associated with a linguistic object (*sign*) is defined in terms of its *word-order domain* (Reape, 1989; Reape, 1990a). I take a word-order domain as a sequence of signs. Each of these signs is associated with a word-order domain recursively, or with a sequence of words. A word-order domain is thus a tree. Linear precedence rules are defined that constrain possible orderings of signs in such a word-order domain. Surface strings are a direct function of word-order domains. In the lexicon, the word-order domain of a lexical entry is defined by sharing parts of this domain with the arguments it subcategorizes for. Word-order domains are percolated upward. Hence word-order domains are constructed in a derivation by gradual instantiations (hence strings are constructued in a derivation by gradual instantiation as well). Note that this implies that an unsaturated sign is not associated with one string, but merely with a set of possible strings (this is similar to the *semantic* interpretation of unsaturated signs (Moore, 1989)). In lexical entries, word order domains are defined using Reape's *sequence union* operation (Reape, 1990a). Hence the grammars are not only based on context-free string concatenation.

2 Constraint-based versions of categorial grammar

The formalism I assume consists of definite clauses over constraint languages in the manner of Höhfeld and Smolka (1988). The constraint language at least consists of the path equations known from PATR II (Shieber, 1989), augmented with variables. I write such a definite clause as:

$$p :- q_1 \dots q_n, \phi.$$

where p, q_i are atoms and ϕ is a (conjunction of) constraint(s). The path equations are written as in PATR II, but each path starts with a variable:

$$\langle X_i \; l_1 \dots l_n \rangle \doteq c$$

or

$$\langle X_i \; l_1 \dots l_n \rangle \doteq \langle X_j \; l'_1 \dots l'_m \rangle$$

where X_k are variables, c is a constant, l, l' are attributes. I also use some more powerful constraints that are written as atoms.

This formalism is used to define what possible 'signs' are, by the definition of the unary predicate **sign/1**. There is only one nonunit clause for this predicate. The idea is that unit clauses for **sign/1** are lexical entries, and the one nonunit clause defines the (binary) application rule. I assume that lexical entries are specified for their arguments in their 'subcat list' (**sc**). In the application rule a head selects the first (**f**) element from its subcat list, and the tail (**r**) of the subcat list is the subcat list of the mother; the semantics (**sem**) and strings (**phon**) are shared between the head and the mother.

$$\begin{aligned} sign(X_0) :- \; & sign(X_1),\; sign(X_2), \\ & \langle X_0 \; synsem \; sem \rangle \doteq \langle X_1 \; synsem \; sem \rangle, \\ & \langle X_0 \; phon \rangle \doteq \langle X_1 \; phon \rangle, \\ & \langle X_0 \; synsem \; sc \rangle \doteq \langle X_1 \; synsem \; sc \; r \rangle, \\ & \langle X_1 \; synsem \; sc \; f \rangle \doteq \langle X_2 \rangle. \end{aligned}$$

I write such rules using matrix notation as follows; **string(X)** represents the value **Y**, where **string(X,Y)**.

$$\begin{bmatrix} X_0 : \begin{bmatrix} synsem : \begin{bmatrix} sem : \boxed{2} \\ sc : \boxed{4} \end{bmatrix} \\ phon : \boxed{3} \end{bmatrix} \\ X_1 : \begin{bmatrix} synsem : \begin{bmatrix} sem : \boxed{2} \\ sc : \langle \boxed{1} \cdot \boxed{4} \rangle \end{bmatrix} \\ phon : \boxed{3} \end{bmatrix} \\ X_2 : \boxed{1} \end{bmatrix}$$

The grammar also consists of a number of lexical entries. Each of these lexical entries is specified for its subcat list, and for each subcat element the semantics and word-order domain is specified, such that they satisfy a termination condition to be defined in the following section. For example, this condition is satisfied if the semantics of each element in the subcat list is a proper subpart of the semantics of the entry, and each element of the subcat list is a proper subpart of the word-order domain of the entry. The phonology of a sign is defined with respect to the word-order domain with the predicate 'string'. This predicate simply defines a left-to-right depth-first traversel of a word-order domain and picks up all the strings at the terminals. It should be noted that the way strings are computed from the word-order domains implies that the string of a node

$$\begin{bmatrix} synsem : \boxed{3}\begin{bmatrix} syn : vp \\ sc : \langle \boxed{2}\begin{bmatrix} synsem : \begin{bmatrix} syn : np \\ sc : \langle\rangle \\ sem : \boxed{4}\end{bmatrix}\end{bmatrix}\rangle \\ sem : schlafen(\boxed{4})\end{bmatrix} \\ dom : \boxed{1}\langle \boxed{2}, \begin{bmatrix} synsem : \boxed{3} \\ dom : \langle\rangle \\ phon : \langle \textit{schläft}\rangle\end{bmatrix}\rangle \\ phon : string(\boxed{1})\end{bmatrix}$$

Figure 1: The German verb 'schläft'

not necessarily is the concatenation of the strings of its daughter nodes. In fact, the relation between the strings of nodes is defined indirectly via the word-order domains.

The word-order domains are sequences of signs. One of these signs is the sign corresponding to the lexical entry itself. However, the domain of this sign is empty, but other values can be shared. Hence the entry for an intransitive German verb such as 'schläft' (sleeps) is defined as in figure 1.

I introduce some syntactic sugaring to make such entries readable. Firstly, XP_i will stand for

$$\begin{bmatrix} synsem : \begin{bmatrix} syn : xp \\ sem : \boxed{i} \\ sc : \langle\rangle \end{bmatrix}\end{bmatrix}$$

Furthermore, in lexical entries the **synsem** part is shared with the **synsem** part of an element of the word order domain, that is furthermore specified for the empty domain and some string. I will write: $\ll$ *string* $\gg$ in a lexical entry to stand for the sign whose **synsem** value is shared with the **synsem** of the lexical entry itself; its **dom** value is $\langle\rangle$ and its **phon** value is *string*. The foregoing entry is abreviated as:

$$\begin{bmatrix} synsem : \begin{bmatrix} syn : vp \\ sem : schlafen(\boxed{1}) \\ sc : \langle \boxed{2} NP_1\rangle \end{bmatrix} \\ dom : \boxed{3}\langle \boxed{2}, \ll \textit{schläft} \gg\rangle \\ phon : string(\boxed{3})\end{bmatrix}$$

Note that in this entry we merely stipulate that the verb preceded by the subject constitutes the word-order domain of the entire phrase. However, we may also use more complex constraints to define word-order constraints. In particular, as already stated above, LP constraints are defined which holds for word-order domains. I use the sequence-union predicate (abbreviated **su**) defined by Reape as a possible constraint as well. This predicate is motivated by clause union and scrambling phenomena in German. A linguistically motivated example of the use of this constraint can be found in section 4. The predicate $su(A, B, C)$ is true in case the elements of the list C is the multi set union of the elements of the lists A and B; moreover, $a < b$ in either A or B iff $a < b$ in C. I also use the notation $X \cup_{\langle\rangle} Y$ to denote the value `Seq`, where `su(X,Y,Seq)`. For example, $su([a,d,e],[b,c,f],[a,b,c,d,e,f])$; $[a,c] \cup_{\langle\rangle} [b]$ stands for $[a,c,b]$,$[a,b,c]$ or $[b,a,c]$. In fact, I assume that this predicate is also used in the simple cases, in order to be able to spel out generalizations in the linear precedence constraints. Hence the entry for 'schlafen' is defined as follows, where I write $lp(X)$ to indicate that the lp constraints should be satisfied for X. I have nothing to say about the definition of these constraints.

$$\begin{bmatrix} synsem : \begin{bmatrix} syn : vp \\ sem : schlafen(\boxed{1}) \\ sc : \langle \boxed{2} NP_1\rangle \end{bmatrix} \\ \boxed{3} dom : \langle \boxed{2}\rangle \cup_{\langle\rangle} \langle \ll \textit{schläft} \gg\rangle \\ phon : string(lp(\boxed{3}))\end{bmatrix}$$

In the following I (implicitly) assume that for each lexical entry the following holds:

$$\begin{bmatrix} dom : \boxed{1} \\ phon : string(lp(\boxed{1}))\end{bmatrix}$$

3 Uniform Processing

In van Noord (1991) I define a parsing strategy, called 'head-corner parsing' for a class of grammars allowing more complex constraints on strings than context-free concatenation. Reape defines generalizations of the shift-reduce parser and the CYK parser (Reape, 1990b), for the same class of grammars. For generation head-driven generators can be used (van Noord, 1989; Calder *et al.*, 1989; Shieber *et al.*, 1990). Alternatively I propose a generalization of these head-driven parsing- and generation algorithms. The generalized algorithm can be used both for parsing and generation. Hence we obtain a ***uniform*** algorithm for both processes. Shieber (1988) argues for a uniform architecture for parsing in generation. In his proposal, both processes are (different) instantiations of a parameterized algorithm. The algorithm I define is not parameterized in this sense, but really uses the same code in both directions. Some of the specific properties of the head-driven generator on the one hand, and the head-driven parser on the other hand, follow from general constraint-solving techniques. We thus obtain a uniform algorithm that is suitable for linguistic processing. This result should be compared with other uniform scheme's such as SLD-resolution or some implementations of type inference (Zajac, 1991, this volume) which clearly are also uniform but faces severe problems in the case of lexicalist grammars, as such scheme's do not take into account the specific nature of lexicalist grammars (Shieber *et al.*, 1990).

Algorithm. The algorithm is written in the same formalism as the grammar and thus constitutes a meta-interpreter. The definite clauses of the object-grammar are represented as

$$lexical\_entry(X) :- \phi.$$

for the unit clauses

$$sign(X) :- \phi.$$

and

$$rule(H, M, A) :- \phi.$$

for the rule

$$sign(M) :- sign(H), sign(A), \phi.$$

The associated interpreter is a Prolog like top-down backtrack interpreter where term unification is replaced by more general constraint-solving techniques, (Höhfeld and Smolka, 1988; Tuda *et al.*, 1989; Damas *et al.*, 1991). The meta-interpreter defines a head-driven bottom-up strategy with top-down prediction (figure 2), and is a generalization of the head-driven generator (van Noord, 1989; Calder *et al.*, 1989; van Noord, 1990a) and the head-corner parser (Kay, 1989; van Noord, 1991).

```
prove(T) :-
    lexical_entry(L), connect(L,T),
    ⟨T phon⟩ ≐ ⟨L phon⟩,
    ⟨T synsem sem⟩ ≐ ⟨L synsem sem⟩.

connect(T,T).
connect(S,T) :-
    rule(S,M,A), prove(A),
    connect(M,T).
```

Figure 2: The uniform algorithm

In the formalism defined in the preceding section there are two possible ways where non-termination may come in, in the constraints or in the definite relations over these constraints. In this paper I am only concerned with the second type of non-termination, that is, I simply assume that the constraint language is decidable (Höhfeld and Smolka, 1988). [1] For the grammar sketched in the foregoing section we can define a very natural condition on lexical entries that guarantees us termination of both parsing and generation, provided the constraint language we use is decidable.

The basic idea is that for a given semantic representation or (string constraining a) word-order domain, the derivation tree that derives these representations has a finite depth. Lexical entries are specified for (at least) **sc**, **phon** and **sem**. The constraint merely states that the values of these attributes are dependent. It is not possible for one value to 'grow' unless the values of the other attributes grow as well. Therefore the constraint we propose can be compared with GB's projection principle if we regard each of the attributes to define a 'level of description'. Termination can then be guaranteed because derivation trees are restricted in depth by the value of the **sc** attribute.

In order to define a condition to guarantee termination we need to be specific about the inter-

[1] This is the case if we only have PATR equations; but probably not if we use $\cup_{\langle\rangle}$, `string/2`and *lp/2* unlimited.

pretation of a lexical entry. Following Shieber (1989) I assume that the interpretation of a set of path equations is defined in terms of directed graphs; the interpretation of a lexical entry is a set of such graphs. The 'size' of a graph simply is defined as the number of nodes the graph consists of. We require that for each graph in the interpretation of a lexical entry, the size of the subgraph at **sem** is strictly larger than each of the sizes of the **sem** part of the (subgraphs corresponding to the) elements of the subcat list. I require that for each graph in the interpretation of a lexical entry, the size of **phon** is strictly larger than each of the sizes of (subgraphs corresponding to) the **phon** parts of the elements of the subcat lists. Summarizing, all lexical entries should satisfy the following condition:

Termination condition. For each interpretation L of a lexical entry, if E is an element of L's subcat list (i.e. $\langle L\ synsem\ sc\ r^*\ f\rangle \doteq E$), then:

$$size[\langle E\ phon\rangle] < size[\langle L\ phon\rangle]$$
$$size[\langle E\ synsem\ sem\rangle] < size[\langle L\ synsem\ sem\rangle]$$

The most straightforward way to satisfy this condition is for an element of a subcat list to share its semantics with a proper part of the semantics of the lexical entry, and to include the elements of the subcat list in its word-order domain.

Possible inputs. In order to prove termination of the algorithm we need to make some assumptions about possible inputs. For a discussion cf. van Noord (1990b) and also Thompson (1991, this volume). The input to parsing and generation is specified as the goal

$?\!-\ sign(X_0), \phi.$

where ϕ restricts the variable X_0. We require that for each interpretation of X_0 there is a maximum for parsing of $size[\langle X_0\ phon\rangle]$, and that there is a maximum for generation of $size[\langle X_0\ synsem\ sem\rangle]$.

If the input has a maximum size for either semantics or phonology, then the uniform algorithm terminates (assuming the constraint language is decidable), because each recursive call to 'prove' will necessarily be a 'smaller' problem, and as the order on semantics and word-order domains is well-founded, there is a 'smallest' problem. As a lexical entry specifies the length of its subcat list, there is only a finite number of embeddings of the 'connect' clause possible.

4 Some examples

Verb raising. First I show how Reape's analysis of Dutch and German verb raising constructions can be incorporated in the current grammar (Reape, 1989; Reape, 1990a). For a linguistic discussion of verb-raising constructions the reader is referred to Reape's papers. A verb raiser such as the German verb 'versprechen' (promise) selects three arguments, a **vp**, an object **np** and a subject **np**. The word-order domain of the **vp** is unioned into the word order domain of **versprechen**. This is necessary because in German the arguments of the embedded **vp** can in fact occur left from the other arguments of **versprechen**, as in:

es$_i$ ihm$_j$ jemand$_k$ zu lesen$_i$ versprochen$_j$ hat$_k$
(it him someone to read promised had
i.e. someome had promised him to read it.

Hence, the lexical entry for the raising verb 'versprechen' is defined as in figure 3. The word-order domain of 'versprechen' simply is the sequence union of the word-order domain of its **vp** object, with the **np** object, the subject, and **versprechen** itself. This allows any of the permuations (allowed by the LP constraints) of the **np** object, **versprechen**, the subject, and the elements of the domain of the **vp** object (which may contain signs that have been unioned in recursively).

Seperable prefixes. The current framework offers an interesting account of seperable prefix verbs in German and Dutch. For an overview of alternative accounts of such verbs, see Uszkoreit (1987)[chapter 4]. At first sight, such verbs may seem problematic for the current approach because their prefixes seem not to have any semantic content. However, in my analysis a seperable prefix is lexically specified as part of the word-order domain of the verb. Hence a particle is not identified as an element of the subcat list. Figure 4 might be the encoding of the German verb 'anrufen' (call up). Note that this analysis conforms to the condition of the foregoing section, because the particle is not on the subcat list. The advantages of this analysis can be summarized as follows.

Firstly, there is no need for a feature system to link verbs with the correct prefixes, as eg. in Uszkoreit's proposal. Instead, the correspondence is directly stated in the lexical entry of the particle verb which seems to me a very desirable

$$\begin{bmatrix} \mathit{synsem}: \begin{bmatrix} \mathit{sem}: \mathit{versprechen}(\boxed{4},\boxed{5},\boxed{6}) \\ \mathit{sc}: \langle \begin{bmatrix} \mathit{synsem}: \begin{bmatrix} \mathit{syn}: \mathit{vp} \\ \mathit{sem}: \boxed{6} \\ \mathit{sc}: \langle NP_4 \rangle \end{bmatrix} \\ \mathit{dom}: \boxed{1} \end{bmatrix}, \boxed{2} NP_5, \boxed{3} NP_4 \rangle \end{bmatrix} \\ \mathit{dom}: \langle \ll \mathit{versprechen} \gg \rangle \cup_{()} \boxed{1} \cup_{()} \langle \boxed{2} \rangle \cup_{()} \langle \boxed{3} \rangle \end{bmatrix}$$

Figure 3: The German verb 'versprechen'

result.

Secondly, the analysis predicts that particles can 'move away' from the verb in case the verb is sequence-unioned into a larger word-order domain. This prediction is correct. The clearest examples are possibly from Dutch. In Dutch, the particle of a verb can be placed (nearly) anywhere in the verb cluster, as long as it precedes its matrix verb:

*dat jan marie piet heeft willen zien bellen op
dat jan marie piet heeft willen zien op bellen
dat jan marie piet heeft willen op zien bellen
dat jan marie piet heeft op willen zien bellen
dat jan marie piet op heeft willen zien bellen

that john mary pete up has want see call
(i.e. john wanted to see mary call up pete)

The fact that the particle is not allowed to follow its head word is easily explained by the (independently motivated) LP constraint that arguments of a verb precede the verb. Hence these curious facts follow immediately in our analysis (the analysis makes the same prediction for German, but because of the different order of German verbs, this prediction can not be tested).

Thirdly, Uszkoreit argues that a theory of seperable prefixes should also account for the 'systematic orthographic insecurity felt by native speakers' i.e. whether or not they should write the prefix and the verb as one word. The current approach can be seen as one such explanation: in the lexical entry for a seperable prefix verb the verb and prefix are already there, on the other hand each of the words is in a different part of the word-order domain.

5 HPSG Markers

In newer versions of HPSG (Pollard and Sag, 1991) a special 'marker' category is assumed for which our projection principle does not seem to work. For example, complementizers are analyzed as markers. They are not taken to be the head of a phrase, but merely 'mark' a sentence for some features. On the other hand, a special principle is assumed such that markers do in fact select for certain type of constituents. In the present framework a simple approach would be to analyze such markers as functors, i.e. heads, that have one element in their subcat list:

$$\begin{bmatrix} \mathit{synsem}: \begin{bmatrix} \mathit{syn}: \mathit{vp\_sub} \\ \mathit{sem}: \boxed{1} \\ \mathit{sc}: \langle \boxed{2}\ VP\_FIN_1 \rangle \end{bmatrix} \\ \mathit{dom}: \langle \ll \mathit{dass} \gg, \boxed{2} \rangle \end{bmatrix}$$

However, the termination condition defined in the third section can not always be satisfied because these markers usually do not have much semantic content (as in the preceding example). Furthermore these markers may also be phonetically empty, for example in the HPSG-2 analysis of infinite vp's that occur independently such an empty marker is assumed. Such an entry would look presumably as follows, where it is assumed that the empty marker constitutes no element of its own domain:

$$\begin{bmatrix} \mathit{synsem}: \begin{bmatrix} \mathit{syn}: \mathit{vp\_inf\_sub} \\ \mathit{sem}: \boxed{1} \\ \mathit{sc}: \langle \boxed{2}\ VP\_INF_1 \rangle \end{bmatrix} \\ \mathit{dom}: \langle \boxed{2} \rangle \end{bmatrix}$$

It seems, then, that analyses that rely on such marker categories can not be defined in the current framework. On the other hand, however, such markers have a very restricted distribution, and are never recursive. Therefore, a slight mod-

$$\left[\begin{array}{l} synsem : \left[\begin{array}{l} syn : vp \\ sem : anrufen(\boxed{1}) \\ sc : \langle \boxed{2} NP_1 \rangle \end{array}\right] \\ dom : \langle \ll ruft \gg \rangle \cup_{()} \langle \boxed{2} \rangle \cup_{()} \langle \left[\begin{array}{l} synsem : syn : part \\ dom : \langle\rangle \\ phon : \langle an \rangle \end{array}\right] \rangle \end{array}\right]$$

Figure 4: The verb 'anrufen'

ification of the termination condition can be defined that take into account such marker categories. To make this feasible we need a constraint that markers can not apply arbitrarily. In HPSG-2 the distribution of the English complementizer 'that' is limited by the introduction of a special binary feature whose single purpose is to disallow sentences such as 'john said that that that mary loves pete'. It is possible to generalize this to disallow any marker to be repeatedly applied in some domain. The 'seed' of a lexical entry is this entry itself; the seed of a rule is the seed of the head of this rule unless this head is a marker in which case the seed is defined as the seed of the argument. In a derivation tree, no marker may be applied more than once to the same seed. This 'don't stutter' principle then subsumes the feature machinery introduced in HPSG-2, and parsing and generation terminates for the resulting system.

Given such a system for marker categories, we need to adapt our algorithm. I assume lexical entries are divided (eg. using some user-defined predicate) into markers and not markers; markers are defined with the predicate *marker(Sign, Name)* where *Name* is a unique identifier. Other lexical entries are encoded as before. *marktypes(L)* is the list of all marker identifiers. The idea simply is that markers are applied top-down, keeping track of the markers that have already been used. The revised algorithm is given in figure 5.

```
prove(T) : -
    marktypes(M), prove(T, M).

prove(T, M) : -
    marker(L, Name), del(Name, M, M2),
    rule(L, T, A), prove(A, M2).

prove(T, M) : -
    lexical_entry(L), connect(L, T),
    ⟨T phon⟩ ≐ ⟨L phon⟩,
    ⟨T synsem sem⟩ ≐ ⟨L synsem sem⟩.

connect(T, T).
connect(S, T) :-
    rule(S, M, A), prove(A),
    connect(M, T).
```

Figure 5: The algorithm including markers

Acknowledgements

I am supported by SFB 314, Project N3 BiLD.

References

Jonathan Calder, Mike Reape, and Henk Zeevat. An algorithm for generation in unification categorial grammar. In *Fourth Conference of the European Chapter of the Association for Computational Linguistics*, pages 233–240, Manchester, 1989.

Luis Damas, Nelma Moreira, and Giovanni B. Varile. The formal and processing models of CLG. In *Fifth Conference of the European Chapter of the Association for Computational Linguistics*, Berlin, 1991.

Marc Dymetman, Pierre Isabelle, and Francois Perrault. A symmetrical approach to parsing and generation. In *Proceedings of the 13th In-*

ternational Conference on Computational Linguistics (COLING), Helsinki, 1990.

Markus Höhfeld and Gert Smolka. Definite relations over constraint languages. Technical report, 1988. LILOG Report 53; to appear in Journal of Logic Programming.

Martin Kay. Head driven parsing. In *Proceedings of Workshop on Parsing Technologies*, Pittsburgh, 1989.

Robert C. Moore. Unification-based semantic interpretation. In *27th Annual Meeting of the Association for Computational Linguistics*, Vancouver, 1989.

Carl Pollard and Ivan Sag. *Information Based Syntax and Semantics, Volume 2.* Center for the Study of Language and Information Stanford, 1991. to appear.

Mike Reape. A logical treatment of semi-free word order and bounded discontinuous constituency. In *Fourth Conference of the European Chapter of the Association for Computational Linguistics*, UMIST Manchester, 1989.

Mike Reape. Getting things in order. In *Proceedings of the Symposium on Discontinuous Constituency*, ITK Tilburg, 1990.

Mike Reape. Parsing bounded discontinous constituents: Generalisations of the shift-reduce and CKY algorithms, 1990. Paper presented at the first CLIN meeting, October 26, OTS Utrecht.

Stuart M. Shieber, Gertjan van Noord, Robert C. Moore, and Fernando C.N. Pereira. A semantic-head-driven generation algorithm for unification based formalisms. In *27th Annual Meeting of the Association for Computational Linguistics*, Vancouver, 1989.

Stuart M. Shieber, Gertjan van Noord, Robert C. Moore, and Fernando C.N. Pereira. Semantic-head-driven generation. *Computational Linguistics*, 16(1), 1990.

Stuart M. Shieber. A uniform architecture for parsing and generation. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, Budapest, 1988.

Stuart M. Shieber. *Parsing and Type Inference for Natural and Computer Languages.* PhD thesis, Menlo Park, 1989. Technical note 460.

Henry S. Thompson. Generation and translation - towards a formalism-independent characterization. In *Proceedings of ACL workshop Reversible Grammar in Natural Language Processing*, Berkeley, 1991.

Hirosi Tuda, Kôiti Hasida, and Hidetosi Sirai. JPSG parser on constraint logic programming. In *Fourth Conference of the European Chapter of the Association for Computational Linguistics*, Manchester, 1989.

Hans Uszkoreit. *Word Order and Constituent Structure in German.* CSLI Stanford, 1987.

Gertjan van Noord. BUG: A directed bottom-up generator for unification based formalisms. *Working Papers in Natural Language Processing, Katholieke Universiteit Leuven, Stichting Taaltechnologie Utrecht*, 4, 1989.

Gertjan van Noord. An overview of head-driven bottom-up generation. In Robert Dale, Chris Mellish, and Michael Zock, editors, *Current Research in Natural Language Generation.* Academic Press, 1990.

Gertjan van Noord. Reversible unification-based machine translation. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING)*, Helsinki, 1990.

Gertjan van Noord. Head corner parsing for discontinuous constituency. In *29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, 1991.

Rémi Zajac. A uniform architecture for parsing, generation and transfer. In *Proceedings of ACL workshop Reversible Grammar in Natural Language Processing*, Berkeley, 1991.

Henk Zeevat, Ewan Klein, and Jo Calder. Unification categorial grammar. In Nicholas Haddock, Ewan Klein, and Glyn Morrill, editors, *Categorial Grammar, Unification Grammar and Parsing.* Centre for Cognitive Science, University of Edinburgh, 1987. Volume 1 of Working Papers in Cognitive Science.