Asymmetry in Parsing and Generating with Unification Grammars: Case Studies From ELU

Graham Russell,* Susan Warwick,* and John Carroll†

*ISSCO, 54 rte. des Acacias
1227 Geneva, Switzerland
russell@divsun.unige.ch

† Cambridge University Computer Laboratory
New Museums Site, Pembroke Street
Cambridge CB2 3QG

Abstract

Recent developments in generation algorithms have enabled work in unification-based computational linguistics to approach more closely the ideal of grammars as declarative statements of linguistic facts, neutral between analysis and synthesis. From this perspective, however, the situation is still far from perfect; all known methods of generation impose constraints on the grammars they assume.

We briefly consider a number of proposals for generation, outlining their consequences for the form of grammars, and then report on experience arising from the addition of a generator to an existing unification environment. The algorithm in question (based on that of Shieber et al. (1989)), though among the most permissive currently available, excludes certain classes of parsable analyses.

1. Introduction

Parsing and generation are both concerned with the relation between texts and representations, and in so far as a grammar defines this relation without reference to direction, it may be regarded as reversible. Yet, in practice, the program which 'applies' a grammar for the purpose of parsing is quite distinct from the one which performs generation.[1]

The essential difference between parsing and generating lies in the nature of the input. The text, as a string of words, traditionally establishes the starting point of parsing; whether the processing is top-down or bottom-up, the basis for selecting grammar rules is information associated with words in the lexicon. In the case of generation, there is in general no guarantee that the constituents of an input representation correspond to words; a portion of the input may be related directly to a given word, or it may be the result of combining representations associated to some sequence of rules, portions of which are ultimately related to lexical items. For example, if the sentence *John kicked the bucket* receives the semantic representation **die(John)**, it is relatively easy to see how during parsing the recognition of *kicked* and *the bucket* will provide the necessary information (from the lexical entry for *kick*) to build that representation. The representation and the lexical items are in general related not directly, but rather via intermediate syntactic rules, any of which is able to manipulate the representation in arbitrary ways; in generation, it is not possible to identify the correct lexical item without considering the syntactic rules which may intervene.

The generation problem, then, consists in how to build a syntactic structure from an initial representation, taking it as the root, and extending the structure 'downward' to the lexicon by selecting rules from the grammar and attaching them at the appropriate points.

Though unification based systems have been in use for parsing for a number of years, generation has until recently not attracted comparable attention; Wedekind (1988), Dymetmann & Isabelle (1988) and Shieber (1988) describe three systems of note. Not surprisingly, given the relative infancy of these explorations, none of these systems is without problems. The most permissive of the current proposals appears to be Shieber et al.'s (1989) revision of the Shieber (1988) algorithm, yet several plausible grammatical analyses handled by the parser are beyond the capacity of even this approach.

This paper reports on experience arising from the addition of a generator component to the ELU[2] environment; the algorithm is a variant of that proposed in Shieber et al. (1989). We first consider general aspects of adapting unification grammars initially developed for parsing to their use in generation. A brief description of the generator in ELU highlights the differences and improvements we have adopted. We then demonstrate shortcomings

[1] Parsing and generation need not employ different algorithms or control strategies; see Shieber (1988) for discussion. However, a truly reversible program would be an entirely different undertaking from what is described here. One such project is currently under way at New Mexico State University (Yorick Wilks, p.c.).

[2] "Environnement Linguistique d'Unification". Cf. Johnson & Rosner (1989) for a description of UD (Unification Device) which includes the parser and facilities such as procedural abstractions and extended data types (lists and trees) and Estival et al. (1989) for a description of the extended ELU system which incorporates the original UD plus a generation and translation component.

of this class of generation algorithms on the basis of two case studies.

2. Generating with Unification Grammars

The goal of employing a single, minimally augmented, grammar for both parsing and generation has become more accessible with the introduction of declarative grammar formalisms (cf. Kay, 1985). In the context of machine translation, for which the ELU system has been developed, the use of the same grammar for both tasks is highly desirable; indeed much of the work on bidirectional grammars has been carried out in centres working on MT (cf. Busemann, 1987; van Noord, to appear; Dymetmann & Isabelle, 1988; and Wedekind, 1988). Regardless of the application, however, the ability to generate with a grammar is extremely useful as a method of checking its adequacy.

Despite the objective of reversibility, all of the systems mentioned here impose generation-specific restrictions on their grammars, either by limiting the form of possible rules or by augmenting them with annotations. Dymetmann & Isabelle (1988) require the grammar writer to specify for each rule the order in which daughters should be generated; however, an order that might be correct when generating from one structure can lead to non-terminating search with another. Busemann (1987) and Saint-Dizier (1989) describe methods of generation which rely on the parsing of a control structure using a specialized grammar to build the syntax of a sentence; it is questionable to what extent the latter two systems can be considered to operate with bidirectional grammars.

Constraints imposed by Wedekind (1988) and van Noord (to appear) exclude certain linguistic analyses from generation. In order to overcome the high degree of non-determinism inherent in the top-down approach, Wedekind stipulates that a daughter of a rule must be 'connected' (i.e. that its semantics must be instantiated) before it can be generated from. Less restrictively, van Noord stipulates similar constraints on rules, i.e. that if the semantics of the mother node is known, then the semantics of the head daughter is instantiated, and additionally that if the syntax of the semantic head is known, then the semantics of each daughter is known. These restrictions limit the class of possible analyses, excluding accounts appropriate to LFG (Kaplan and Bresnan, 1982), HPSG (Pollard & Sag, 1987) and UCG (Zeevat et al., 1987).

The disparate state of progress in parsing and generation raises important issues concerning the adequacy of grammatical descriptions and the computational tools that interpret them. A situation exists in which a grammar may be 'correct' for analysis, but 'incorrect' for generation. Significantly, this may be the case even when the restrictions and annotations mentioned above are taken into account. Grammatical analyses developed in a purely parsing environment cannot always be transferred straightforwardly into a format suitable for generation. Two types of conclusion may be drawn from this: failures may be ascribed to inadequacies of current generator technology, or the grammatical analyses in question may be re-evaluated. Practical remedies will involve two related strands of research; improving methods of generation so as to minimize restrictions on the form of grammars that can be generated from, and identifying problematic properties of grammars. It is the second of these which the present paper chiefly addresses, though we also remark, in the next section, on some enhancements to the Shieber et al. (1989) algorithm that have been incorporated in the ELU generator.

3. The Generator in ELU

In this section we describe the generation algorithm in ELU, and discuss in what respects it differs from that described by Shieber et al. (1989).[3] Two notions central to this method of generation are that of the 'pivot', and that of partitioning the grammar into 'chaining' and 'non-chaining' rules. Loosely, the 'pivot' of a structure to be generated from is the lowest node in a path down semantic heads of rules at which the semantics of the current generation root structure remains unchanged. A chaining rule is one in which the semantics of the object associated with the right-hand side category that has been declared as the head unifies with that of the left-hand side category. Other rules are non-chaining rules. Rules that apply between the root and the pivot are, by definition, chaining rules; further, any rule which can be attached below the pivot is, by definition, a non-chaining rule. Rules are partitioned into these two groups during grammar compilation.

Once the chaining rules have been identifed, the grammar compiler computes the possible sequences of such rules along a path through their mothers and semantic heads. The result is a 'reachability table', each of whose elements is a pair of restrictor value sets[4] representing classes of FSs which can occur at the top and bottom of such a path; in each case, the 'bottom' restrictor set characterizes a pivot. A restrictor set is also computed for each lexical stem, in order to retrieve words efficiently during generation.

The generation algorithm uses the distinction between chaining and non-chaining rules as well as

[3] Our discussion will therefore assume familiarity with this paper.

[4] Restrictors are attributes selected by the writer of a grammar as being maximally distinctive; when two FSs are to be unified, their respective restrictor values are first checked for compatibility, so as to eliminate the cost of an attempted unification which is bound to fail. See Shieber (1985).

that between head and non-head daughters, the reachability table for chaining rules, the semantic portion of the FS to be generated from[5], and the restrictors for lexicon stems. The algorithm is:

1. Take all grammar rules declared as 'initial' (or all rules in the grammar if no such declaration has been made); for each of these rules whose mother unifies with the input FS, apply the rule top-down, building FSs for each of the daughters, and, starting with the head daughter, execute step 2 for each one. If generation from the daughters is successful, compute all possible word-forms (as constrained by the locally available syntactic information) for each lexical stem generated.
2. Create a pivot consisting of just the semantic portion of the current FS. Non-deterministically perform steps 2a and 2b:
   a. Find a lexical stem which unifies with the pivot, making sure (by checking with the reachability table) that the FS resulting from the unification can be linked through semantic heads of just chaining rules up to the current FS.
   b. Find a non-chaining rule which can have the pivot as mother, similarly making sure that the FS resulting from the unification of the pivot and the mother can be linked up to the current FS. Recursively (through 2) generate the rule's daughters, starting with the head daughter.
3. Link the pivot up to the current FS through semantic heads of just chaining rules (at each stage, before adding a new rule in the chain, checking with the reachability table that further linking will be possible) and then recursively (through 2) generate the non-head daughters of these rules.

In this algorithm non-chaining rules are used top-down, while chaining rules are used bottom-up. Linking information is used both to check the applicability of a lexical stem or a non-chaining rule when generating top-down from a pivot, and also to control search when generating bottom-up, by ensuring that the left-hand side of any rule considered still lies on a possible path through chaining rules to the current FS.

One innovation of the ELU generator is that the notion 'semantic head' is interpreted rather differently; whereas the earlier work simply defines the semantic head of a rule as the daughter whose semantics unifies with that of the left-hand side, and thus leaves the notion undefined for non-chaining rules, that described here permits the grammar writer to identify one daughter in each rule as the semantic head. A rule in which a daughter shares the semantics of the mother can thus be made into a chaining rule or a non-chaining rule, according to whether that daughter is identified as the semantic head, and a rule that would otherwise have multiple semantic heads can be assigned just one.[6] A rule in which there is no such daughter will remain a non-chaining rule, but may nevertheless be annotated with a similar specification. The rationale is two-fold: the ability to coerce what would otherwise be a chaining rule to a non-chaining rule grants the grammar writer more control over generation, and the ability to specify one daughter as semantically more significant than the others may be exploited in order to direct the attention of the generator towards that daughter.

A second difference is the order of events in bottom-up generation. Instead of generating from the non-head daughters of each chaining rule as it is attached, the pivot is first linked to the root, so that, if backtracking is forced, effort will not have been spent on processing structure that must be discarded.

Finally, on each occasion that top-down generation is initiated, an attempt is made to add a lexical item below the current root, rather than extending the path by application of non-chaining rules until no such rule is applicable. Here, the motivation is that lexical information may be made available as soon as possible without forcing the grammar writer to adopt analyses that will produce bottom-up generation. This is important because global syntactic properties of a sentence are often determined by lexical information.

4. Grammars for Generation

4.1. Introduction

In this section we examine more closely interactions between generator and grammar. These fall under two headings: (i) the presence of non-determinism in the grammar, and (ii) the role of lexicalism.

One aspect of non-determinism in generation, that of the ordering of rule application, is partially overcome in ELU by the user specification of the head daughter. Non-determinism with respect to the order of solving constraint equations is less well understood. The use of restrictors helps to reduce the number of feature structures to be considered.

[5] The relevant paths being determined by the user's declaration

[6] Thus circumventing a problem noted by Shieber et al. (1989, fn.4) in connection with such rules. Van Noord (p.c.) stipulates that any daughter which has the same semantics as the mother, but is not the semantic head, may not branch: this constraint is clearly too strong, precluding, among other things, linguistically motivated accounts of coordination.

However, in ELU, the use of relational abstractions as a generalization of template facilities increases the problem considerably.[7] Relational abstractions permit the grammar writer to augment the phrase structure rules with statements which may receive multiple definitions in terms of constraint equations; the 'Linear Precedence' definition in (2) below is an example. This facility is a standard ELU device for collapsing what would in an unextended PATR-like formalism be several distinct rules, thereby capturing linguistic generalizations that would otherwise go unexpressed.

It is particularly important to control nondeterminism in generation, since, at least when processing is initiated, there is relatively little information available to direct the search. Expanding multiple definitions as they are encountered would give rise to an unacceptable number of alternatives, many of which might be identical, and often the information from the abstraction is not required until all but one of the alternatives have been excluded by other factors. This is not always the case, however, and when exceptions occur their effect may be drastic. We now describe one such exception to demonstrate how an elegant analysis for parsing is unsuitable for generation.

4.2. A grammar for French clitics

A common technique in modern lexically-oriented grammars, and one which reflects and extends the traditional notion of 'valency', is to encode information about the various phrases with which a verb combines in items on a subcategorization list. The grammar then enforces a match between a member of the list and a phrase which is to combine with some projection of the verb and removes the item from the list. When a sentence is complete, i.e. the verb has 'found' all necessary phrases, a grammar may require that the list be empty, or perhaps that any remaining item is in some way specified as optional. See e.g. Shieber (1986) and Pollard and Sag (1987) for applications of this method.

A complete grammar of French must account for the position and ordering of clitic pronouns. These precede the verb, while other complement phrases follow. Moreover, they appear in a fixed order, as shown in (1):

(1)

me	le	lui	y	en
te	la	leur		
se	les			
nous				
vous				

Up to three clitics may occur, but for the sake of this discussion, we consider only the simpler case of two clitics as complement phrases to the verb.[8] There are of course many ways of accounting for their distribution;[9] the subcategorization list device seems a natural solution, since any complement phrase may be realized as a clitic. The grammar rule in (2) introduces up to two clitics before the verb, their relative order determined by a relational abstraction which is defined by a number of clauses, each clause licensing one of the possible clitic sequences.

(2)
```
vplus -> Cl1 Cl2 HV
         !Precede(Cl1,Cl2)
         List = <HV subcat> -- Cl1
         <vplus subcat> = List -- Cl2
Precede(X,Y)
         <X person> = first/second
         <Y person> = third
Precede(X,Y)
         <X case> = accusative
         <Y case> = dative
...
```

Some remarks on notation will be helpful: calls to relational abstractions are indicated by the exclamation mark, feature-value disjunction is indicated by the slash, and an equation of the form 'X = Y -- Z', where X and Y are lists, unifies X non-deterministically with the result of extracting one instance of Z from Y.

The effect of this rule, then, is to associate a pair of clitics with a verb, checking that they are correctly ordered, and unifying the subcategorization list of the left-hand side category with a copy of that of the head verb from which objects unifying with each of the clitics have been removed.

The problem emerges when information assumed to be held in the subcategorization list of 'vplus' is required in order to control further generation. For example, if 'vplus' appears as sister to another complement phrase, and the same procedure of unifying the latter with an item on the list takes place, then because the generator has suspended expansion of non-deterministic abstractions, the subcategorization list itself will be uninstantiated, and therefore no information regarding the semantics of the complement phrase will be available to restrict top-down generation.

[7] Cf. Johnson & Rosner (1989) for a fuller description of relational abstractions.

[8] This is something of an oversimplification, as not only complement phrases, but also adverbials and parts of complement phrases are realized as clitics. See Grimshaw (1982) for a partial LFG account of these phenomena. We also ignore the issue of negation, which considerably complicates the clitic-aux-verb structure.

[9] The categorial treatment proposed in Baschung et al. (1987) not only makes use of order of arguments, but also codes each clitic for all possible combinations.

Modifications to the syntactic constituency assumed here do not affect the principle; as long as the instantiation of so central an element of the grammar as the subcategorization list is delayed, the problem will remain. An alternative type of analysis would remove the non-determinism from the grammar by factoring it out into a larger number of rules. This solution is not without its own disadvantages; the number of distinct rules needed by a full treatment of French clitics, integrated with the placement of the various negative particles and auxiliaries, should not be underestimated. We postpone further discussion of non-determinism and delay until the conclusion and turn now to the problem of empty semantic heads, an important problem for head-driven generation algorithms.[10]

4.3. Empty Semantic Heads

In German and Dutch, there are two positions in a sentence where tensed verbs may appear: in second position of a main clause, and in final position of a subordinate clause. Once again, a multitude of analyses are possible within ELU grammars. One approach is to control the distribution of verbs with grammar rules specific to clause-type; this solution gives rise to what might be felt to be an unacceptable degree of duplication in the grammar. A more elegant approach, successful for parsing, exploits the possibility of associating a word or phrase appearing in one position within a sentence with a 'gap' elsewhere.

The latter analysis will be recognized as a variant of a standard Government-Binding treatment, in which a tensed verb in a main clause is 'raised' from an 'underlying' sentence-final position to a 'surface' second position (see e.g. Haider (1985), Platzack (1985) for discussion of this class of analyses). The dependency may be implemented by the use of a feature, say 'v2', whose value in a verb-second construction is a feature structure representing the verb to be raised, and in other constructions an atomic constant such as 'none', which serves to block the dependency. At the extraction site, any value of 'v2' other than 'none' may be cashed out as an empty production. Information regarding the various syntactic properties of the raised verb is passed in the normal fashion between the verb's true position and the extraction site, where it is able to exert the same constraints upon complement phrases that a lexically-realized verb would.

The simplified rule set given in (3) will serve as a basis for discussion. Recall that the generator operates by partitioning the rules of the grammar into classes to be applied top-down (non-chaining rules – here 'S-gap' and 'V2') and bottom-up (chaining rules – here 'TOP', 'S' and 'V'). Bottom-up generation is only practical if the input structure to that phase of generation contains sufficient information, e.g. the verb with its subcategorization list.

(3)
```
# Rule TOP
TOP -> XP H_S
 <* cat> = top      <* head> = <H_S head>
 <XP cat> = np      <H_S subcat> = [XP]
 <H_S cat> = sbar
# Rule V2
Sbar -> H_V2 S
 <* cat> = sbar     <H_V2 cat> = v
 <S cat> = s        <* subcat> = <S subcat>
 <S v2> = H_V2      <* head> = <S head>
 <H_V2 head syn vform> = finite
# Rule S
S -> XP H_S
 <S cat> = s        <XP cat> = np
 <H_S cat> = s      <* v2> = <H_S v2>
 <* subcat> = <H_S subcat> -- XP
 <* head> = <H_S head>
# Rule V
S -> H_V
 <S cat> = s        <* head> = <H_V head>
 <H_V cat> = v      <* subcat> = <H_V subcat>
# Rule S-gap
S -> -
 <S cat> = s        <S head> = <V2 head>
 <S v2> = V2        <S subcat> = <V2 subcat>
```

The verb-raising analysis sketched here has the unfortunate property of supplying the generator with a semantic head (the verb gap) about which nothing is known. At the stage when top-down processing has identified the verb gap as the starting point for bottom-up generation, the input feature structure is underspecified. In particular, the subcategorization list of the missing verb is uninstantiated, and in the grammar in question, it is the length of this list which controls invocation of the recursive rule 'S'. No bindings can be found, and the generator suspends evaluation of that equation in the hope, ill-founded on this occasion, that information not yet present will later allow its solution. The result is that 'S' is repeatedly added above 'S-gap', in a non-terminating attempt to ensure completeness of the search.

Van Noord (1989) describes two solutions to this problem, both of which are additions to the original program, and whose only motivation (so far) is to overcome this specific problem. The first, somewhat ad-hoc, solution allows the verb to have as one of its morphological realizations the empty string. Since word forms are generated at the end of processing by a morphological front-end, the generator can posit the same word in both positions

[10] This problem is alluded to in Shieber et al. (1989, fn.4) and is discussed in a draft of an expanded version of the paper.

(for the purpose of retrieving its subcategorization behaviour from the lexicon, for example). The morphological component then generates one empty string and one full word according to the position of the verb (i.e. in a main or subordinate clause). This mechanism is not available in ELU. The second solution adds an additional 'connect' clause in the Prolog program, specific to gaps, in order to assure that the gap is first instantiated before further processing; this solution raises the issue of tuning programs to treat specific problems as they are encountered.

There are other constructions which raise the same kind of problem; the fronting of apparently non-constituent verbal sequences in German (Nerbonne, 1986) introduces more complex dependencies, while in English the phenomena of Gapping and Verb-Phrase Ellipsis both manifest themselves syntactically in the absence from a sentence of a verb and possibly other material. Here, the difficulty is, if anything, greater, as the dependencies in question are anaphoric in nature, rather than syntactic.

5. Conclusion

We have seen, in the preceding section, how in order to write grammars suitable for use with the generator, one must either modify the technical aspects of the grammar or dispense with certain classes of grammatical analysis (losing the benefits of relational abstraction on one hand, and lexicalism on the other, for example). Both of these may be interpreted as restricting the freedom of the grammar writer. The problematic case illustrated in section 4.2 raises the issue of non-determinism, a potential pitfall for all unification-based systems. In parsing, the result may be long processing times, but when generating with algorithms of this class, the consequence is often non-termination. As Shieber et al. (1989, fn.4) observe, failure to choose the right daughter as the starting point for recursive generation may prevent termination.

The desire to exploit the power of unification by using the lexicon as a repository of essentially syntactic (beyond pure semantic) information is natural, and has been encouraged by the success in theoretical linguistics of grammatical formalisms which employ such techniques. Yet the use of these techniques in grammar writing, which are highly attractive from the point of view of economy and expressive power, deprives the generator of information that is, strictly speaking, syntactic. Semantic heads alone are not sufficient to drive the generation process, if syntactic information cannot also be made available. Our interim conclusion is that strong versions of the lexicalist position do not appear to be compatible with our current generator, at least for a number of cases. This is not to say that it should be abandoned – the benefits in terms of clarity and economy are probably too great – but some care is needed if it is to be exploited effectively.

Given that work on this type of generation is in its early stages, it is to be hoped that continuing research will enable less restricted grammars to be written. Nevertheless, the currently available facilities have been employed successfully in general, making it possible to envisage defining the 'adequacy' of a grammar in terms of its behavior both in parsing and in generation.

References

Baschung, K, G.G. Bes, A. Corluy, and T. Guillotin (1987) "Auxiliaries and Clitics in French UCG Grammar". *Proceedings of the Third Conference of the European Chapter of the Association for Computational Linguistics*, Copenhagen, Denmark, April 1st–3rd 1987: 173–178.

Bresnan, J. (ed.) (1982) *The Mental Representation of Grammatical Relations*. Cambridge, MA: MIT Press.

Busemann, S. (1987) "Generierung mit GPSG". KIT-Report 49, Technische Universität Berlin.

Dymetman, M. & P. Isabelle (1988) "Reversible Logic Grammars for Machine Translation". *Proceedings of the 2nd International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages*, Carnegie-Mellon University, Pittsburgh, USA.

Estival, D., A. Ballim, G. Russell, and S. Warwick (1989) "A Syntax and Semantics for Feature-Structure Transfer". MS, ISSCO.

Grimshaw, J. (1982) "On the Lexical Representation of Romance Reflexive Clitics", in Bresnan (ed.): 87 – 148.

Haider, H. (1985) "V-Second in German", in H. Haider and M. Prinzhorn (eds.) *Verb Second Phenomena in Germanic Languages*: 49 – 75. Dordrecht: Foris.

Johnson, R. and M. Rosner (1989) "A Rich Environment for Experimentation with Unification Grammars". *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, Manchester, UK, April 10th–12th 1989: 182–189.

Kaplan, R.M. and J. Bresnan (1982) "Lexical-Functional Grammar: A Formal System for Grammatical Representation", in Bresnan (ed.): 173–281.

Kay, M. (1985) "Parsing in Functional Unification Grammar", in D. Dowty, L. Karttunen, and A. Zwicky (eds.) *Natural Language Parsing*. Cambridge: Cambridge University Press: 251–278.

Nerbonne, J. (1986) "'Phantoms' and German Fronting: Poltergeist Constituents?". *Linguistics 24–5*, 857–870.

van Noord, G. (to appear) "Bottom Up Generation in Unification-based Formalisms", in C. Mellish, R. Dale, and M. Zock (eds.) *Proceedings of the Second European Workshop on Natural Language Generation*.

Platzack, C. (1985) "A Survey of Generative Analyses of the Verb Second Phenomenon in Germanic". *Nordic Journal of Linguistics 8*: 49–73.

Pollard, C. and I.A. Sag (1987) *Information-Based Syntax and Semantics, Volume 1: Fundamentals*. CSLI Lecture Notes no. 13

Saint-Dizier, P. (1989) "A Generation Method Based on Principles of Government-Binding Theory". Paper presented at the Second European Natural Language Generation Workshop, Edinburgh, April 1989.

Shieber, S.M. (1985) "Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms". *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*: 145–152.

Shieber, S.M. (1986) *An Introduction to Unification-Based Approaches to Grammar*. CSLI Lecture Notes no. 4.

Shieber, S.M. (1988) "A Uniform Architecture for Parsing and Generation". *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest, Hungary: 614–619.

Shieber, S.M., van Noord, G., R.C. Moore, and F.C.N. Pereira (1989) "A Semantic-Head-Driven Algorithm for Unification-Based Formalisms". *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*: 7–17.

Wedekind, J. (1988) "Generation as Structure-Driven Derivation". *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest, Hungary: 732–737.

Zeevat, H., E. Klein, and J. Calder (1987) "Unification Categorial Grammar". *Categorial Grammar, Unification Grammar, and Parsing*, Edinburgh Working Papers in Cognitive Science, Volume 1. Centre for Cognitive Science, University of Edinburgh: 195–222