Categorial Unification Grammars*

Hans Uszkoreit
Artificial Intelligence Center
SRI International and
Center for the Study of Language
and Information at Stanford University

Abstract

Categorial unification grammars (CUGs) embody the essential properties of both unification and categorial grammar formalisms. Their efficient and uniform way of encoding linguistic knowledge in well-understood and widely used representations makes them attractive for computational applications and for linguistic research.

In this paper, the basic concepts of CUGs and simple examples of their application will be presented. It will be argued that the strategies and potentials of CUGs justify their further exploration in the wider context of research on unification grammars. Approaches to selected linguistic phenomena such as long-distance dependencies, adjuncts, word order, and extraposition are discussed.

0. Introduction

The work on merging strategies from unification grammars and categorial grammars has its origins in several research efforst that have been pursued in parallel. One of them is the grammar development on the PATR system (Shieber et al., 1983; Shieber, 1984) at SRI. For quite a while now I have been using the excellent facilities of PATR for the design and testing of experimental CUGs. Such grammars currently run on two PATR implementations: Stuart Shieber's Zetalisp version on the Symbolics 3600 and Lauri Karttunen's Interlisp-D version on the XEROX 1109. The work on CUGs has influenced our efforts to develop a larger PATR grammar, and will do so even more in the future. On the theoretical side, this work is part of ongoing research on such topics as word order variation, modification, and German syntax within projects at SRI and CSLI (Stanford University).

The structure of the paper reflects the diverse nature of the enterprise. In the first section, I will introduce the basic notions of CUGs and demonstrate them through examples in PATR notation. The second section discusses the motivation for this work and some of its theoretical implications. The third section sketches a linguistically motivated CUG framework with a strong lexical syntax that accomodates word order variation. The paper concludes with a brief discussion of possible CUG approaches to long-distance dependencies.

1. Basic Notions of Categorial Unification Grammars

1.2. Unification Grammars and Categorial Grammars

Both terms, unification grammar (UG) and categorial grammar (CG), stand for whole families of related grammar formalisms whose basic notions are widely known.[1] Yet, for the characterization of the class of formalisms I want to discuss, it will be useful to review the most central concepts of both UG and CG.

Unification grammar formalisms employ complex feature structures as their syntactic representations. These structures encode partial information about constituents. Either term or graph unification is utilized as the main operation for checking, propagating, and merging of the information in these complex representations. Most unification grammars also use the complex feature structures for the linking of syntactic and semantic information.

In traditional categorial grammars, all information about possible syntactic combinations of constituents is encoded in their categories. Those grammars allow only binary combinations. One of the two combined constituents, the functor, encodes the combination funtion, the other constituent serves as the argument to this function. Instead of phrase structure rules, the grammar contains one or, in some formalisms, two combination rules that combine a functor and an argument by applying the function encoded in the functor to the argument constituent. Most categorial grammars only combine constituents whose terminal strings concatenate in the input string, but this need not be so. In most categorial grammar formalisms, it is assumed that the syntactic functor-argument structure in the corresponding compositional semantics.

There are usually two types of grammatical categories in a categorial grammar, basic and derived ones. Basic categories are just category symbols, derived categories are functions from one (derived or basic) category to another. A derived category that encodes a function from category A to category B might be written B/A if the functor combines with an argument to its right or B\A, if it expects the argument to its left. Thus, if we assume just two basic categories, N and S, then N/S, S/N, N\S, S\N, (S\N)/N, (N/S\(S\(N/N)), etc. are also categories. Not all of these categories will ever occur in the derivation of sentences. The set of actually occurring categories depends on the lexical categories of the language.

Assume the following simple sample grammar:

(2) Basic categories: N, S

lexical categories: N *(Paul, Peter)*

(S\N)/N *(likes)*

The grammar is used for the sample derivation in (3):

(3)

Peter	likes	Paul
N	(S\N)/N	N
	S\N	
	S	

It should be clear from my brief description that the defining characteristics of unification grammar have nothing to do with the ones of categorial grammar. We will see that the properties of both grammar types actually complement each other quite well.

1.2. A Sample CUG in PATR Notation

Since the first categorial unification grammars were written in the PATR formalism and tested on the PATR systems implemented at SRI, and since PATR is especially well suited for the emulation of other grammar formalisms, I will use its notation.

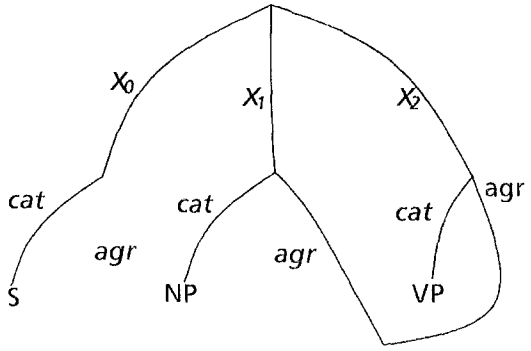
The representations in PATR are directed acyclic graphs (DAGs)[2] . Rules have two parts, a head and a body. The head is a context-free rewrite rule and the body is a DAG. Here is an example, a simple rule that forms a sentence by combining a noun phrase with a verb phrase.

(4) head XO → X1, X2

body in unification notation

<X0 cat> = S
<X1 cat> = NP
<X2 cat> = VP
<X1 agr> = <X2 agr>

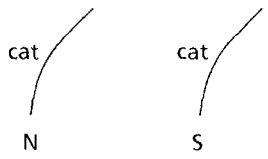
body in graph notation



The rule states that two constituents X1 and X2 can combine to form a constituent X0 if the terminal string covered by X1 immediately precedes the terminal string of X2 and if the DAGs of X0, X1, and X2 unify with the X0, X1, and X2 subgraphs of the rule body, respectively.

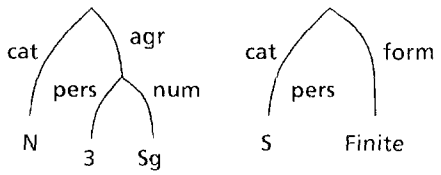
I will now show the most straight-forward encoding of a categorial grammar in this notation. There are two types of constituent graphs. Constituent graphs for basic categories are of the following form:

(5)



Of course, there might be more features associated with the constituent:

(6)

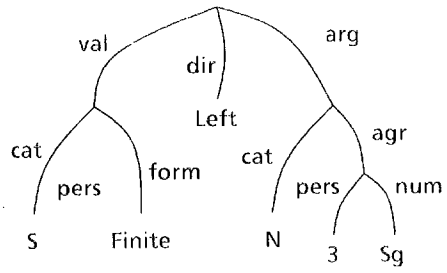


Derived constituents have graphs of the following form:

(7)

This is the graph associated with the VP *likes Paul*:

(8)



It corresponds to the derived-category symbol:

(9)

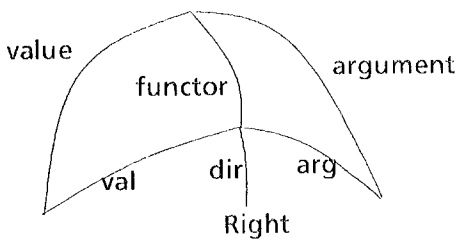
S	\	N
form : Finite		pers : 3
		num : Sg

(10a) and (10b) are the rules that combine constituents. As in traditional categorial grammars, two such rules sufice.

(10a) Forward Functional Application (FFA)

value → functor argument
<value> = <functor val>
<argument> = <functor arg>
<functor dir> = Right.

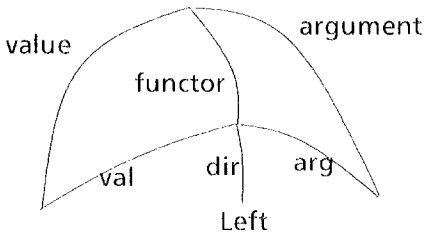
in graph notation:



(10b) Backward Functional Application (BFA)

value → functor argument
<value> = <functor val>
<argument> = <functor arg>
<functor dir> = Left.

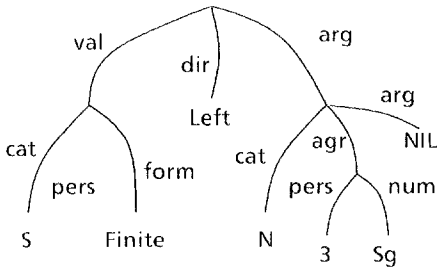
in graph notation:



If Backward Functional Application is used to combine the constituents *Peter* and *likes Paul*, the result is a finite sentence.

However, if the same rule is applied to the identical constituents *likes Paul* and *likes Paul*, again a finite sentence is obtained. This is so because the graph for *likes Paul* actually unifies with the value of arg in the same graph. This can be easily remedied by modifying the graph for the VP slightly. By stipulating that the argument must not have an unfilled argument position, one can rule out derived categories as subject arguments for the VP:

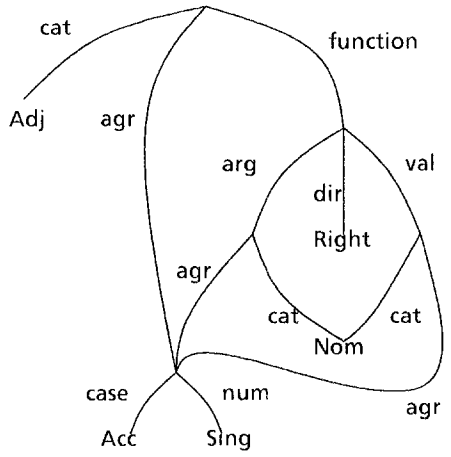
(11)



1.3. Extensions to the Basic Formalism

In this subsection I want to discuss very briefly a few extensions of the basic model that make it more suitable for the encoding of natural-language grammars. The first one is the sorting of functors according to their own syntactic category. This move might be described alternatively as defining the type of a constituent as being defined by both a set of syntactic (and semantic)

attributes and a function from categories to categories. This function is also expressed as the value of an attribute. For a basic category the value of the function attribute is NIL. The following graph is a simplified example of a functor category (prenominal adjective in a language with case and number agreement within the NP).



The combination rules need to be changed accordingly. This is the modified rule of forward functional application.

value → functor argument
<value> = <functor function val>
<argument> = <functor function arg>
<functor function dir> = Right.

In a traditional categorial grammar, a derived category is exhaustively described by the argument and value categories. But often, syntacticians want to make more fine grained distinctions. An example is VP modification. In a traditional categorial grammar, two different VP modifiers, lets say an adverb and an adverbial clause, would receive the same translation.

(12)	Peter	called	him	angrily
	N	(S\N)/N	N	(S\N)/(S\N)

(13)	Peter	called	him	at work
	N	(S\N)/N	N	(S\N)/(S\N)

But what should be the category for *very*? If it receives the category ((S\N)\(S\N))/((S\N)\(S\N)) to allow the derivation of (14), the ungrammatical sentence (15) is also permitted.

(14)	Peter	called	him	very	angrily
	N	(S\N)/N	N	((S\N)\(S\N))/ ((S\N)\(S\N))	(S\N)/(S\N)

(15)	*Peter	called	him	very	at work
	N	(S\N)/N	N	((S\N)\(S\N))/ ((S\N)\(S\N))	(S\N)/(S\N)

If functor categories are permitted to carry features of their own that are not necessarily bound to to any features of their argument and value categories, this problem disappears. Adverbs and adverbial clauses could receive different features even if their categories encode the same combination function.

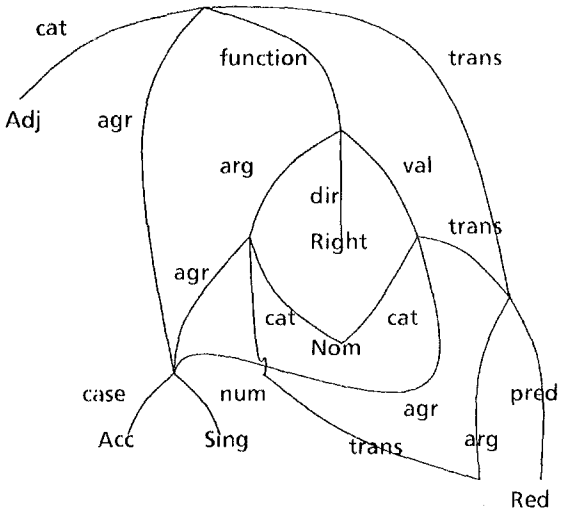
Another solution to the problem involves the encoding of the difference in the value part of the functor. Yet this solution is not only unintuitive but also contradicts a linguistic generalization. It is unintuitive because there is no difference in the distribution of the resulting VPs. The only difference holds between the modifiers themselves. The generalization that is violated by the encoding of the difference in the value subgraphs is the endocentricity of the VP. The modified VP shares all syntactic features with its head, the lower VP. Yet the feature that indicates the difference between adverbs and adverbial phrases could not be in both the argument and the value parts of the functor, otherwise iterations of the two types of modifiers as they occur in the following pair of sentences would be ruled out.

(16a) Peter called him very angrily at work.

(16b) Peter called him at work very angrily.

Another augmentation is based on the PATR strategy for linking syntax and semantics. Most grammars written in PATR use the constituent graphs also for encoding semantic information. Every constituent has an attribute called trans or semantics. The value of this attribute contains minimally the internal semantic function-argument structure of the

constituent, but may also encode additional semantic information. The separate encoding of the semantics allows for a compositional semantics even in construction in which syntactic and semantic structure divert as in certain raising constructions. The following graph for a ficticious prenominal adjective that was introduced earlier contains translation attributes for the functor, the argument and the value. The meaning of the adjective is indicated by the atom *Red*.



At first glance, the lexical graphs--even the ones that are used in the highly simplified examples--seem to exhibit an excessive degree of complexity and redundancy. However, the lexical approach to syntax is built on the assumption that the lexicon is structured. To create a lexicon that is structured according to linguistic generalizations, we introduced lexical templates early on in the development of PATR.

Templates are graphs that contain structure shared by a class of lexical entries. Lexical graphs can be partially or fully defined in terms of templates, which themselves can be defined in terms of templates. If a template name appears in the definition of some graph, the graph is simply unified with the graph denoted by the template.

The next augmentation is already built into the formalism. Categorial grammarians have recognized the limitations of functional application as the sole mode of combining constituents for a long time. One of the obvious extensions to classical categorial grammar was the utilization of functional composition as a further combination mode. A good example of a categorial grammar that employs both functional application and functional composition is Steedman (1985). Forward functional composition permits the following combination of categories:

(21) A/B + B/C = A/C

The resulting category inherits the argument place for C from the argument B/C.

Neither Steedman's nor any other CG I am aware of permits functional composition in its full generality. In order to prevent overgeneration, functional composition as well as other combination modes that are discussed by Steedman are restricted to apply to certain categories only. This somehow violates the spirit of a categorial grammar. Steedman's combination rules, for instance, are not universal.

In CUG, functional composition is subsumed under functional application. It is the functor category that determines whether simple functional application, or functional composition, or either one may take place. Conjunction is a good case for demonstrating the versatility.

Consider the following sentences:[3]

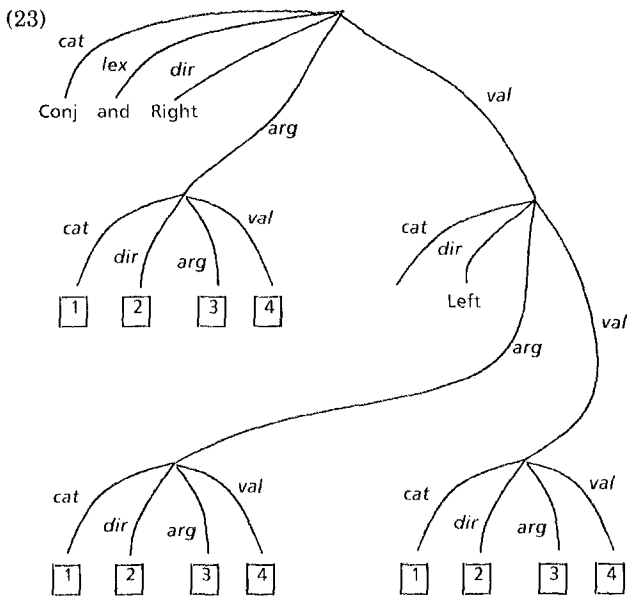
(22a) Peter and Paul like bananas.

(22b) Peter likes bananas and Paul likes oranges.

(22c) Peter likes and buys bananas.

The conjunction *and* may combine two simple argument categories (22a), two functors with one unfilled argument position (22b), or two functors with more than one unfilled argument position (22c). If the conjuncts have unfilled argument positions, the conjoined phrase needs to inherit them through functional composition. The simplified lexical graph for *and* is given under (23). In order to avoid a thicket of crossing edges, I have expressed some of the relevant bindings by indices.

(23)



The most appealing feature of this way of utilizing functional composition is that no additional combinators are required. No restriction on such a rule need to be formulated. It is only the lexical entries for functors that either demand, permit, or forbid functional composition.

Extensions to the formalism that I have experimented with that cannot be discussed in the frame of this paper are the use of multiple stacks for leftward and rightward arguments and the DCG-like encoding of the ordering positions in the graphs. In Sections 3. and 4., I will discuss further extensions of the formalism and specific linguistic analyses. The following section contains a summary of the motivations for working on and with CUG and the main objectives of this work.

2. Motivation and Theoretical Implications

Both terms, unification grammar and categorial grammar are used for classes of grammar formalisms, for individual grammar formalisms, and finally for grammars that are written in these formalisms. In addition, they might also be used by linguists to denote linguistic theories that are built around or on top of such a formalism. This is the type of terminological overloading that linguists have learned to live with--or at least gotten accustomed to.

As I indicated in the previous section, I consider CUG to stand for a family of grammar formalisms that might be described as the intersection of categorial and unification grammar formalisms. What has been proposed so far is therefore not a new grammar formalism and even less a linguistic framework.

The proposal is simply to further explore the usefulness and formal properties of subclasses of CUG. This proposal can be supported by a number of reasons.

Both types of formalisms have clear advantages. Categorial grammars have been hailed for their conceptual clarity and their potentials for linking syntax and semantics. The fact that they have been around for a long time and that they are currently enjoying a renaissance in the works of Steedman, Bach, Dowty, and many others demonstrates their virtues. Unification grammars are spreading fast and lend themselves to powerful but efficient computer implementations.

Traditionally, categorial grammars have been lacking syntactic sophistication. In a functor category such as A/B, only domain and range of the function are specified but nothing is said about how they are related; how, for instance, the features of the argument influence the features of the value. The graph notation expresses the relation between argument and value categories quite well; it is expressed in a set of bindings between subgraphs of the two categories.

In the context of this discussion, some remarks are in order on the specific role PATR has played for the experiments with CUGs. The philosophy behind the development of PATR has been to provide a tool for writing, testing, and comparing grammars of very different types in a powerful formalism with well-understood formal properties and a well-defined semantics (Shieber 1984).

Thus PATR could be useful for writing grammars, designing grammar formalisms, and for exploring classes of such formalisms. The work on exploring categorial unification formalisms has not only benefitted from the features of PATR but it has in a way also influenced the development of the PATR formalism. It was, for instance, essential for the writing of categorial grammars to allow category variables in the context-free phrase structure part of the rules. How elso could one formulate the rules of functional application. The implementation of this facility through Stuart Shieber, however, raised interesting problems in connection with the prediction aspect of the Earley-parser. Original Earley prediction works on category symbols. An answer to these problems was presented by Shieber (1985) who proposed to do Earley prediction on the basis of some finite quotient of all constituent DAGs which can be specified by the grammar writer.

Another example for the influence of the CUG efforts on the development of PATR is a new template notation introduced by Lauri Karttunen in his Interlisp-D version of PATR. Since categorial grammars exhibit an extensive embedding of categories within other categories, it is useful to unify templates not only with the whole lexical DAG but also with its categorial subgraphs. The @-notation permits this use of templates (Karttunen, 1986).[4]

3. A CUG Grammar Model that Accomodates Word Order Variation

Worder order variation has always been one of the hardest problems for categorial grammars. Functional composition together with type-raising can be used to obtain all permutations of the sentences that are generated by a traditional categorial grammar. Totally free word order does therefore not pose an unsurmountable problem to the categorial approach. As with other types of grammar formalisms, it is semi-free word order that is difficult to accommodate.

GPSG, LFG, and FUG all have mechanisms for encoding ordering regularities. Such a device does not exist in the categorial grammars that I am aware of. However, Uszkoreit (1985a,b) argues (on the basis of data from German) for an application of linear precedence rules to the valency list of syntactic functors. This approach presupposes that the valency list contains adjuncts as well as complements as the functor's syntactic arguments.[5]

The model can be summarized as follows. The lexicon lists uninstantiated entries. For functors, these entries contain a set of thematic roles. The uninstantiated lexical entry may also state whether thematic roles have to be filled, whether they may be filled more than once, and whether idiosyncratic properties of the functor predetermine the syntactic features of certain syntactic arguments.

There are three types of rules that instantiate lexical entries: feature instantiation rules, valency instantiation rules, and order instantiation rules.

An instantiated functor has an ordered valency list containing syntactic specifications of complements and adjuncts together with the appropriate semantic bindings. The model can account for the interspersing of complements and adjuncts as they occur in many languages including English. The model can also account for right-extraposition phenomena.

Therefore, the valency list may constain adjuncts that do not fill a thematic role of the functor but combine semantically with some constituent inside a linearily preceding member of the same valency list.[6]

In the proposed model, the dependency between the extraposed phrase and its antecendent is neither established by functional application/composition nor by feature passing. It is assumed that there is a different matching process that combines the noncontiguous phrases. A process of this kind is independently needed for the matching of adjuncts with thematic roles that are embedded in the meaning of the functor:

(26a) Tell me about French history.

(26b) Start in 1700.

The year 1700 is obviously not the start time for the telling.

(27a) His call was very urgent.

(27b) He tried desperately from every phone booth on campus.

It is not *try* that supplies here the source role but the implicit theme of *try*. If the theme role is filled, everybody would analyze the *from* PP as semantically belonging to the theme of *try*:

(28) He tried to call her desperately from every phone booth on campus.

I want to conclude this discussion with a remark on the parsing problem connected with the proposed model. In older PATR Phrase-Structure grammars as well as in the categorial PATR grammars, all graphs that may be connected with a word in the input string are either retrieved from the lexicon or from a cache of already built lexical graphs, or they are constructed on the spot from the lexical entries through the morphology and through lexical rules.

For obvious reasons, this approach cannot be used in conjunction with the categorial model just proposed. If all adjuncts are included in the valency list, and if moreover all acceptable linearizations are performed in the extended lexicon, there is no upper bound on the number of acceptable lexical graphs for functors. This means that lexical entries cannot be fully instantiated when the word is recognized. They need to be instantiated incrementally as potential arguments are encountered.

In Uszkoreit (1985b) it is argued that the ordered valency lists of a functor admitted by the lexical instantiation rules form a regular language. If further research confirms this hypothesis, the incremental instantiation of valency lists could be performed through sets of finite state machines.

4. A Note on Long-distance Dependencies in CUGs

In Steedman's (1985) categorial grammars, long-distance dependencies are endcoded in the function-argument structure of categories. The categories that form the path between filler and gap in a derivation tree all carry a valency slot for the filler. This uniform encoding of both subcategorization and long-distance dependencies in the argument structure of categories seems at first glance superior to the HPSG or PATR approaches to long-distance dependencies, in which the two types of information are marked in different feature sets. However, it turns out that the Steedman grammars have to mark the long-distance valency slots in order to distinguish them from other valency slots.

There could still be a justification for encoding the two types of dependencies in the same argument stack. One might loose important nesting information by separating the two types of slots. However, I have not yet seen a convincing example of nesting constraints among subcategorization and long-distance dependencies. Therefore, I consider the question of the appropriate place for encoding long-distance dependencies still open.

A last remark on long-distance dependencies. In a unification based system like PATR it is not trivial to ensure that gap information is passed up from one daughter constituent only when a rule is applied. There are two ways to enforce this constraint. The first one involves a multiplication of rules. For a binary rule A --> B C, for instance, one could introduce three new rules, one of which does not do any gap passing, another one the passing of a gap from B to A, and the third the passing of a gap from C to A.

PATR uses a little more elegant method which has been first suggested by Fernando Pereira. Two features are threaded through every tree, one of which carries a gap up a tree, passing through all the constituents to the left of the gap, and a second one that is set to NIL if a gap has been found and that is then sent through all the constituents to the right of the gap, unifying it on the way with potential gaps. It requires that information about the two special features be added to every rule. In PATR a preprocessor of rules adds this information for all rules in which the grammar writer did not include any gap threading information herself, e.g., for encoding island constraints.

In a CUG that only contains two (or at least very few) rules, the first method of duplicating rules appears preferrable over the gap threading approach. Rules that propagate gap information might also include rules that permit parasitic gaps along the lines of Steedman's rules of functional substitution.

References

Bar Hillel, Y. (1964) *Language and Information*. Wesley, Reading, Mass.

Johnson, M. (1986) "Fronting and the Internal structure of the VP in German." ms. Stanford University.

Karttunen, L. (1986) "D-PATR: A development environment for unification-based grammars." in this volume.

Shieber, S. M., H. Uszkoreit, F.C.N. Pereira, J.J. Robinson, and M. Tyson (1983) "The Formalism and Implementation of PATR-II." In: *Research on Interactive Acquisition and Use of Knowledge, Artificial Intelligence Center*, SRI International. Menlo Park, California.

Shieber, S. (1984) "The Design of a Computer Language for Linguistic Information." in Shieber, S., L. Karttunen, and F. Pereira (eds.), *Notes from the Unification Underground: A Compilation of Papers on Unification-based Grammar Formalisms*, Technical Note 327, SRI-International, Menlo Park, Cal.

Shieber, S. (1985) "Using Restriction to Extend Parsing Algorithms for Complex Feature-Based Formalisms," in: *Proceedings of the ACL 1985.*

Steedman, M. (1985) "Dependency and Coordination in the Grammar of Dutch and English." *Language* 61: 523-568.

Uszkoreit, H. (1982) "German Word Order in GPSG." In D. Flickinger, M. Macken, and N. Wiegand (Eds.), *Proceedings of the First West Coast Conference on Formal Linguistics*, Stanford University, Stanford, California.

Uszkoreit, H. (1985a) "Problematische Konstruktionen fü kontextfreie Phrasenstrukturgrammatiken des Deutschen." in Klenk, U. (Ed.) *Strukturen und Verfahren in der maschinellen Sprachverarbeitung*, AQ-Verlag, Dudweiler.

Uszkoreit, H. (1985b) "Linear Precedence in Discontinuous Constituents." paper presented at the Conference on Discontinuous Constituency, July 1985, Chicago, Illinois, (to appear in Syntax and Semantics 20).

Uszkoreit, H. (1986) "Constraints on Order." CSLI Report 46, Stanford University (also to appear in *Linguistics.*)

Notes

*The research for this paper was made possible through a gift by the System Development Foundation.

[1]For an introduction to the family of unification grammar models refer to Shieber (forthcoming). A good introduction to the basic notions of categorial grammar is Bar Hillel (1964).

[2]The PATR implementations that are currently used at SRI actually permit cyclic graphs.

[3]Right-Node-Raising (RNR) which leads to sentences as: *Peter likes and Paul buys bananas* will be neglected here (although RNR is an attractive topic for categorial grammarians and one of my grammars actually handles many cases of RNR.)

[4]An even more general notation can be used that does not distinguish between root templates and subgraph templates. As long as template names are marked by some typographic convention, could be freely used wherever a graph is described.

[5]The version of the linear precedence rule component proposed by Uszkoreit (1982, 1986) is fully compatible with this approach. The proposal permits the formalization of partially free word order as it results from the interaction of potentially conflicting ordering principles and as it probably occurs to some degree in all natural languages.

[6]Sag (1985) proposes a mechanism for HPSG that allows the syntactic binding of an extraposed phrase to a complement or adjunct slot of a complement or adjunct. However, this approach is too restricted. Although there is a strong tendency to only extrapose complements and adjuncts of top-level complements and adjuncts, there is certainly no such constraint in languages like English or German. The following sentence could not be handled since the extraposed relative clause modifies an adjunct of the subject. *Petitions from those people were considered who had not filed a complaint before.*

[7]Mark Johnson (1986) has worked out a quasi-categorial solution of this phenomenon in the framework of HPSG.