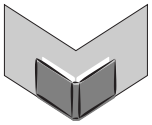
An Introduction to Unification-Based Approaches to Grammar

An Introduction to Unification-Based Approaches to Grammar

An Introduction to Unification-Based Approaches to Grammar

Stuart M. Shieber



Microtome Publishing
Brookline, Massachusetts

Preface

These notes were originally developed as a supplement to a tutorial on unification-based approaches to grammar presented at the 23rd Annual Meeting of the Association for Computational Linguistics on 7 July, 1985 at the University of Chicago.

My intention was to present a set of formalisms and theories from a single organizing perspective, that of their reliance on notions of identity of complex feature structures and on unification in the underlying algebras. Of course, as discussed in the first chapter, such an expositional method precludes covering the formalisms from the perspective put forward by particular researchers in the field. Fortunately, I can now recommend an introduction to the two main linguistic theories discussed in these notes which does present their own view in an introductory way yet with surprisingly broad coverage; Peter Sells' *Lectures on Contemporary Syntactic Theories*, published in this same CSLI Lecture Note Series, serves this purpose admirably.

Because of the brevity of the present work, and its intended (though not completely realized) nonparochial nature, I have purposefully left out any linguistic analyses from a unification-based perspective of any but the most cursory sort. My reasoning here was that the vast linguistics literature in the area provides ample examples of such analyses. (For interested readers, Chapter 5 contains references to some of this research.) A future edition, however, may be extended with examples of analyses of control, long-distance dependencies and various of the other canonical linguistic phenomena described using the tools of unification-based formalisms.

Much of my thinking on these issues has been influenced by researchers at the Center for the Study of Language and Information at Stanford University, and especially those in the Foundations of Grammar project. CSLI is a unique environment for research, and has done more to shape this work and the opinions presented herein than any other single force. I am also indebted to Martin Kay, Fernando Pereira, Ivan Sag, Susan Stucky, Hans Uszkoreit, Thomas Wasow and Annie Zaenen for their comments on earlier drafts of these notes. However, the views and conclusions expressed here are not necessarily shared by them, nor are they accountable for any errors remaining.

The exposition in these notes, such as it is, was considerably improved, as usual, by my editor at SRI, Savel Kliachko. Any unfortunate idiosyncrasies in the text are undoubtedly lapses of the author; Savel never errs. I also extend

heartfelt thanks to the CSLI editor, Dikran Karagueuzian, for his patience and support of this work. Finally, the preparation of these notes was considerably simplified by the efforts of Emma Pease in the formatting and indexing of the final version.

The preparation of the published version of these notes was made possible by a gift from the System Development Foundation.

Menlo Park, California
13 February, 1986

This second printing corrects some typographical and bibliographical errors pointed out by readers of the first printing. Special thanks go to Geoffrey K. Pullum for his assiduous comments on the citations.

Menlo Park, California
2 February, 1988

This reissue of "An Introduction to Unification-Based Approaches to Grammar Formalisms" is only lightly edited from the previous 1988 printing. By now, the content is considered quite elementary and the presentation is grossly out of date, but it may have some remaining utility by virtue of the former property at least. More complete and modern presentations of the formal basis for the type of formalisms described in this book can be found in the books by Carpenter (1992) and Johnson (1988). From a linguistic perspective, a good starting point is the textbook by Sag and Wasow (1999).

Cambridge, Massachusetts
20 August, 2003

Contents

CHAPTER 1

Introduction and History

These notes discuss a particular approach to encoding linguistic information, both syntactic and semantic, which has been called "unification-based" or "complex-feature-based." The term "unification-based grammar formalism" covers quite a variety of formal devices, which have been developed along historically different paths. The fact that they are regarded as a group is due perhaps partly to accidental and historical circumstances, partly to a shared underlying view of grammar, but most importantly to their reliance in some way on one particular device—unification.

Historically, these grammar formalisms are the result of separate research in computational linguistics, formal linguistics, and natural-language processing; related techniques can be found in theorem proving, knowledge representation research, and theory of data types. Several independently initiated strains of research have converged on the idea of unification to control the flow of information.

Beginning with the augmented-transition-network (ATN) concept (like so much of the research in modern computational linguistics) and inspired by Bresnan's work on lexically oriented nontransformational linguistics, the lexical-functional grammar (LFG) framework of Bresnan and Kaplan was evolved. Simultaneously, Kay devised the functional grammar (later unification grammar, now functional unification grammar [FUG]) formalism.

Independently, Colmerauer had produced the Q-system and metamorphosis grammar formalisms as tools for natural-language processing. The logic-programming community, specifically Pereira and Warren, created definite-clause grammars (DCG) on the basis of Colmerauer's earlier work on these formalisms and on the programming language Prolog. Independent work in logic programming has used DCG as the foundation of many unification-based formalisms, such as extraposition, slot, and gapping grammars.

Another strain of parallel research grew out of the work on nontransformational linguistic analyses, from which Gazdar developed generalized phrase structure grammar (GPSG). In its later formalization by Gazdar and Pullum, GPSG imported a unification relation. Pollard defined head grammars in his dissertation, basing them on GPSG. Implementation of GPSG at Hewlett-Packard led Pollard and his colleagues to design their head-driven phrase-structure grammar (HPSG) as a successor to GPSG and head grammars.

Most recently, influenced by early papers on GPSG, Rosenschein and the author devised PATR as a successor to the DIAGRAM grammar formalism at SRI International. Although PATR did not use unification, it developed (under the influence of FUG, later work in GPSG, and DCG) into PATR-II, perhaps the simplest of the unification-based formalisms.

One thing has become clear through this morass of historical fortuities: unification is a powerful tool for grammar formalisms. All these formalisms make crucial use of unification in one manner or another, supplementing it, for various linguistic or computational reasons, with other mechanisms.

A Caveat. Unification formalisms, as we have seen, can be traced to a diversity of sources. Consequently, proponents of each formalism have their own ideas on how their formalisms are appropriately viewed. To present each formalism from the unique perspective of its practitioners would be difficult, confusing, and would ultimately prove a disservice to the formalisms viewed in toto as a coherent group. Instead, we will offer an underlying framework for these formalisms intended to provide a coherent basis for their comparison and evaluation. We hope that none of them will be unduly perverted by this rational reconstruction of the field. Nonetheless, the reader should keep in mind that this approach to the unification-based formalisms is peculiar to the author and may not necessarily be subscribed to by all researchers.

CHAPTER 2

The Underlying Framework

2.1. The Role of Grammar Formalisms

Grammar formalisms are, first and foremost, languages whose intended usage is to describe languages themselves—to describe the set of sentences the language encompasses (the string set), the structural properties of such sentences (their syntax), and the meanings of such sentences (their semantics). Each individual grammar written in a grammar formalism (the *metalanguage*) encodes an analysis of an *object language*. There are several reasons we might want such a metalanguage:

- To provide a precise tool for the description of natural languages.
- To delimit the class of possible natural languages.
- To provide a computer-interpretable characterization of natural languages.

The choice of this metalanguage in which the analyses are encoded is critical, since it determines the following three parameters which serve as important criteria of grammar formalisms:

- *Linguistic felicity:* The degree to which descriptions of linguistic phenomena can be stated directly (or indirectly) as linguists would wish to state them.
- *Expressiveness:* Which class of analyses can be stated at all.
- *Computational effectiveness:* Whether there exist computational devices for interpreting the grammars expressed in the formalism and, if they do exist, what computational limitations inhere in them.

Toward the first goal of a grammar formalism as a descriptive tool, linguistic felicity and expressiveness are most important. Toward the second goal of universal linguistic delimitation, linguistic felicity and *lack* of expressiveness are of foremost importance. Finally, toward the final goal of computer-interpretable linguistic characterization, all three criteria are vital.

The criterion of expressive power merits special consideration. While power is considered an advantage for a descriptive or computational tool, it is detrimental to a universal linguistic theory as commonly construed. Thus, different motivating forces can lead us to different preferences in designing a formalism. As our concern in these notes is primarily with the computational interpretation of grammar formalisms, we will not be overly worried about unconstrained expressive power. Nevertheless, certain of these theories, most notably GPSG and LFG, are especially concerned with linguistic universals. For such theories, expressive power is to be constrained, not promoted.

2.2. Some Particular Design Choices

These very general criteria certainly do not delineate an approach to grammar formalisms very precisely, but they can be used to guide us in choosing a more particular approach. Unification-based formalisms tend to make certain specific assumptions about what a grammar formalism should do. In general, they require that grammar formalisms be

- *surface-based:* providing a *direct* characterization of the *actual surface order* of string elements in a sentence,
- *informational:* associating with the strings information from some *informational domain*,
- *inductive:* defining the association of strings and informational elements *recursively*, with new pairings being derived by merging substrings according to prescribed string-combining operations, and merging the associated informational elements according to prescribed information-combining operations, and
- *declarative:* defining the association between strings and informational elements in terms of *what* associations are permissible, not *how* they are computed.

More specifically, these informational elements are characterizable as

- *complex-feature-based:* as associations between *features* and *values* taken from some well-defined, possibly structured set.

These complex-feature-based informational elements are given various names in the literature; we will uniformly refer to them as *feature structures*. Looking ahead to the more mathematical view of feature structures that we will develop shortly, we can take this domain to be a set of graphs over a finite set of arc labels and a finite set of atomic values. This will provide a useful

mathematical abstraction of the notion of informational element which admits of several combinatorial operations currently used in linguistics. For example, consider the combination of two sets of feature structures that involves taking the union of the feature/value pairs (as long as they are consistent) and, in case both sets have values for the same feature, combining these values recursively. This mode of combination, which can be defined formally as a graph-combining process to reflect this informal description, is exactly the notion of *unification* we have been alluding to, a primary operation of functional unification grammar, lexical-functional grammar, generalized phrase-structure grammar, and definite-clause grammar. Other operations (e.g., generalization, disjunction, and overwriting) can be similarly defined, but unification plays a central role in all of the theories discussed in this paper, thus the term *unification-based grammar formalism.*

2.3. Coverage of the Framework

In Section 5.1 we will discuss mathematical measures of the expressiveness of particular formalisms falling within this methodological class. The following list is intended to help the reader develop an intuitive appreciation of the breadth and diversity of linguistic formalisms that express analyses in this manner. Not all of these formalisms are based on unification, or even on complex feature structures, but they can all be modeled to a great extent in this way; more importantly, they are all surface-based, informational, inductive, and declarative in the broad senses outlined above.

Categorial grammar: A pure categorial grammar, allowing functional application, uses string concatenation to form constituents. The informational elements are complex categories that may be regarded as having a category-valued *functor* feature and an *argument* feature. For instance, a category (S/NP)/NP (e.g., for the verb "loves") might be encoded in a feature/value system with a functor feature whose value is the recursive encoding of S/NP into functor and argument features, and with an argument feature whose value is the final argument NP. Variations on this technique are widely used in PATR-II grammars and grammars based on the head grammar and HPSG formalisms.

The categorial system of Ades and Steedman (1982), a related formalism, still fits within this class although it includes both functional application and composition.

Similarly, Montague grammars (e.g., Montague (1974)) are directly stated as pairings of string-combining and denotation-combining rules. The informational elements can be thought of as being comprised of a complex category feature (as described above for categorial grammar) and a feature whose value is the denotation of the expression. Although certain variants of Montague grammar would be unhappily characterized as *directly* characterizing surface order, much of the literature falls within this broad methodology.

GPSG: GPSG (Gazdar et al., 1985), since it uses a context-free base, involves only concatenation to build up the surface string (as do LFG and DCG). Its informational domain is a restricted feature/value system, involving both simply-valued features (e.g., *number, case*) and complex-valued features (e.g., *slash, refl*).[1] (See Section 4.6.)

Head grammars: Head grammars (Pollard, 1984) and HPSG (Pollard, 1985a) extend GPSG by respectively introducing head-wrapping string operations and removing the restrictions on the feature system that yield GPSGs context-freeness. Nonetheless, such grammars belong to a surface-based feature/value methodology. (See Section 4.7.)

LFG: LFG's (Bresnan, 1982) informational structures, called f-structures, are a recursive feature/value system with certain specialized types of features (e.g., *pred*) and information (e.g., concerning bounding and constraints). (See Section 4.5.)

FUG: Through FUG's (Kay, 1983) patterns, concatenation is mandated as the constituent-forming operation.[2] Functional structures, the informational entities, are a generalized feature/value system. (See Section 4.3.)

[1]The use of metarules requires some flexibility in interpreting GPSG in this class. We merely disregard them and view GPSGs as already being closed under metarules. Note that recent versions of GPSG have made less and less use of metarules, preferring to establish generalizations in the lexicon.

[2]Recent work extending the expressivity of the pattern language allows for more flexibility in combining strings.

DCG: Terms are the basic information-bearing structures in DCG (Pereira and Warren, 1980). They can be thought of as a degenerate case of a feature/value system in which the features correspond to argument positions. In particular, a term $f(a, b, c)$ may be thought of as having a *functor* feature whose *name* is f and whose *arity* is 3, plus three argument features with respective values a, b, and c. (See Section 4.4.)

In fact, viewed from a computational perspective, it is not surprising that so many paradigms of linguistic description can be encoded directly with generalized feature/value structures of this sort. Similar structures have been put forward by various computer scientists as general mechanisms for knowledge representation (Ait-Kaci, 1985) and data types (Cardelli, 1984). Thus, we have hardly constrained ourselves at all even though limited to this methodology.

In summary, the methodological class outlined above involves

- The association of strings with elements in a system of features and (possibly structured) values.
- The inductive building of such associations by the simultaneous rule-based combination of substrings as well as of the associated informational elements.

CHAPTER 3

The Simplest Unification-Based Formalism

3.1. Overview of the Formalism

As an illustrative example, we will fix upon a choice of string- and information-combining operations so as to yield a first, simple, unification-based formalism, the PATR-II formalism developed at SRI International. The constraints we impose are the following:

- *Concatenation:* Concatenation is prescribed as the sole string-combining operation. This causes our formalism to be context-free-based (though certainly not context-free in formal power, as discussed in Section 5.1).

  This first constraint eliminates the possibility of directly stating head-grammar analyses (which use an operation of head-wrapping) and those Montagovian analyses that use such string operations as wrapping (Bach, 1980) and substitution (Montague, 1974). However, analyses within these systems can often be modeled indirectly.
- *Unification:* Unification is prescribed as the sole information-combining operation. This causes our formalism to be completely declarative (see the discussion of Section 5.1) and its interpretation order-independent.

  Reliance on unification is in happy concurrence with linguistic practice, since unification is a primary operation in many current linguistic grammar formalisms; moreover, its typical applications—pattern-matching, equality testing, and feature passing—are found in an even wider range of linguistic analyses. Unification can also be used to model analyses with many other combining operations and can sometimes even substitute for string operations other than concatenation.

We now move beyond general comments to the task of making precise the formalism just described. First, we define the domain of informational

elements and concomitant notions of subsumption and unification. Then we describe how the inductive pairing rules can be notated. This will lead us to our first very simple unification-based formalism.

3.2. The Informational Domain

Unification-based formalisms use as their informational domain a system based on *features* and their *values*. We will refer to elements of this domain as *feature structures*; other works call them f-structures (in LFG), feature bundles, feature matrices or categories (in GPSG), functional structures (in FUG), terms (in DCG), or dags (in PATR-II, as an acronym for directed acyclic graph). A feature structure is a partial function from features to their values. For instance, we might have a function mapping the feature *number* onto the value *singular* and mapping *person* onto *third*. The common notation for such a feature structure is

$$\begin{bmatrix} \textit{number}\colon \textit{singular} \\ \textit{person}\colon \textit{third} \end{bmatrix} \qquad (D_{3sg})$$

Let us call this structure D_{3sg}.[1]

One of the distinguishing characteristics of unification-based formalisms is that the feature values may themselves be structured. For example, D_{3sg} might be just one component (the *agreement* component) of a larger structure associated with, say, a third-person singular noun phrase.

$$\begin{bmatrix} \textit{cat}\colon & \textit{NP} \\ \textit{agreement}\colon & \begin{bmatrix} \textit{number}\colon \textit{singular} \\ \textit{person}\colon \textit{third} \end{bmatrix} \end{bmatrix} \qquad (D_{NP3sg})$$

The final important characteristic of feature structures is that they can be *reentrant*. A reentrant feature structure is one in which two features in the structure share *one common value*. We must carefully distinguish between two features with one value and the weaker notion of two features with two different but *similar* values. Another way of viewing the distinction in terms of *types* and *tokens* of feature structures is that, in the first case, we have one token, whereas in the second, we have two tokens both of the same (one) type.

[1] We will usually use subscripted D to denote feature structures. Also, as done above, we will put such mnemonics parenthetically adjacent to the feature structure.

Of course, since feature structures can be viewed as both types and tokens, this distinction is critical.

For instance, in D_{sim} below, we have features f and g with two distinct feature structure values of the same type.

$$\begin{bmatrix} f\colon \begin{bmatrix} h\colon a \end{bmatrix} \\ g\colon \begin{bmatrix} h\colon a \end{bmatrix} \end{bmatrix} \qquad (D_{sim})$$

In D_{id}, f and g share one value, also of the same type as the value of f or g in D_{sim}.

$$\begin{bmatrix} f\colon \boxed{1} \begin{bmatrix} h\colon a \end{bmatrix} \\ g\colon \boxed{1} \end{bmatrix} \qquad (D_{id})$$

A shared value is notated by using coindexing boxes on the values. The single value is written only once, with a coindexing box labeling it. The other features that share the same value just put the coindexing box as their value. An alternative notation, found in the LFG literature, uses lines from one value to another in the following manner

$$\begin{bmatrix} f\colon \begin{bmatrix} h\colon a \end{bmatrix} \\ g\colon \end{bmatrix} \qquad (D_{id})$$

Other notations could be imagined. Note that the equals symbol (=) is always used for token, not type, identity in this paper.

We now characterize feature structures a little more rigorously in order to introduce some new concepts.

3.2.1. Basic Concepts. Feature structures come in two varieties: *complex* (like D_{3sg}) and *atomic* (like the values *third* and *singular* in D_{3sg}). As mentioned before, complex feature structures can be viewed as partial functions from features to values (which are themselves feature structures). The notation $D(f)$ will therefore denote the value associated with the feature f in the feature structure D. For instance, $D_{3sg}(number) = third$. In the same vein, we can refer to the *domain* of a feature structure D as $dom(D)$. For example, $Dom(D_{3sg}) = \{number, person\}$. A feature structure with an empty domain is often called an *empty* feature structure or a *variable*. Variables are notated (in the obvious way) as [ ]. A *path* in a feature structure is a sequence of features (notated, e.g., $\langle agreement\ number \rangle$), which can be used to pick out

a particular subpart of a feature structure by repeated application, so, by extending the notation $D(p)$ in the obvious way to include the substructure of D picked out by the path p, we conclude that $D_{NP3sg}(\langle agreement\ number \rangle) = third$.

3.2.2. Subsumption. There is a natural lattice structure for feature structures that is based on *subsumption*—an ordering on feature structures that roughly corresponds to the compatibility and relative specificity of information contained in them. Recall that feature structures are serving an informational function; by associating a particular feature structure with a phrase, we are making a claim about that phrase. For example, by associating the feature structure D_{NP3sg} with the phrase "Arthur" we are claiming that the phrase has a certain category and agreement properties, i.e., that it is a third-person singular noun phrase. The simpler feature structure D_{NP}

$$\left[cat\colon NP\right] \qquad (D_{NP})$$

makes the compatible but more general claim about a phrase that it is a noun phrase, but leaves open the question of what its agreement properties are. Thus, D_{NP} is said to carry *less information* than, to be *more general* than, or to *subsume* D_{NP3sg}.

Viewed intuitively, then, a feature structure D subsumes a feature structure D' (notated $D \sqsubseteq D'$) if D contains a subset of the information in D'. More precisely, a complex feature structure D subsumes a complex feature structure D' if and only if $D(l) \sqsubseteq D'(l)$ for all $l \in dom(D)$ and $D'(p) = D'(q)$ for all paths p and q such that $D(p) = D(q)$.[2] An atomic feature structure neither subsumes nor is subsumed by a different atomic feature structure. Variables subsume all other feature structures, atomic or complex, because, as the trivial case, they contain no information at all.

Despite the relatively complex definition, subsumption is a quite intuitive notion thought of from its informational perspective. We list here some examples to justify this intuition. Consider the following feature structures:

$$[\ ] \qquad (D_{var})$$

$$\left[cat\colon NP\right] \qquad (D_{NP})$$

[2]Recall that by "=" here and elsewhere we mean token identity, i.e., that the paths share a common value.

$$\begin{bmatrix} cat: & NP \\ agreement: & \begin{bmatrix} number: & singular \end{bmatrix} \end{bmatrix} \qquad (D_{NPsg})$$

$$\begin{bmatrix} cat: & NP \\ agreement: & \begin{bmatrix} number: & singular \\ person: & third \end{bmatrix} \end{bmatrix} \qquad (D_{NP3sg})$$

$$\begin{bmatrix} cat: & NP \\ agreement: & \begin{bmatrix} number: & singular \\ person: & third \end{bmatrix} \\ subject: & \begin{bmatrix} number: & singular \\ person: & third \end{bmatrix} \end{bmatrix} \qquad (D_{NP3sgSubj})$$

$$\begin{bmatrix} cat: & NP \\ agreement: & \boxed{1} \begin{bmatrix} number: & singular \\ person: & third \end{bmatrix} \\ subject: & \boxed{1} \end{bmatrix} \qquad (D'_{NP3sgSubj})$$

The following subsumption relations hold:

$$D_{var} \sqsubseteq D_{NP} \sqsubseteq D_{NPsg} \sqsubseteq D_{NP3sg} \sqsubseteq D_{NP3sgSubj} \sqsubseteq D'_{NP3sgSubj}$$

3.2.3. Unification. Subsumption is only a partial order—that is, not every two feature structures are in a subsumption relation with each other. This can come about because the feature structures have differing but compatible information, as in

$$\begin{bmatrix} cat: & NP \\ agreement: & \begin{bmatrix} number: & singular \end{bmatrix} \end{bmatrix} \qquad (D_{NPsg})$$

$$\begin{bmatrix} cat: & NP \\ agreement: & \begin{bmatrix} person: & third \end{bmatrix} \end{bmatrix} \qquad (D_{NP3})$$

or because they have conflicting information, e.g.,

$$\begin{bmatrix} cat: & NP \\ agreement: & \begin{bmatrix} number: & singular \end{bmatrix} \end{bmatrix} \qquad (D_{NPsg})$$

$$\begin{bmatrix} cat: & NP \\ agreement: & \begin{bmatrix} number: & plural \end{bmatrix} \end{bmatrix} \qquad (D_{NPpl})$$

The difference between the two cases is that in the first case, there exists a more specific feature structure that is subsumed by both feature structures, namely,

$$\begin{bmatrix} cat: & NP \\ agreement: & \begin{bmatrix} number: & singular \\ person: & third \end{bmatrix} \end{bmatrix} \qquad (D_{NP3sg})$$

whereas in the second case, no such feature structure exists. This notion of combining the information from two feature structures to obtain a feature structure that includes all the information of both is central to unification-based formalisms, for it is the notion of *unification* itself.

Of course, there are many feature structures that are subsumed by D_{NP3} and D_{NPsg}. For instance, the following would have done just as well:

$$\begin{bmatrix} cat: & NP \\ agreement: & \begin{bmatrix} number: & singular \\ person: & third \\ gender: & masculine \end{bmatrix} \end{bmatrix} \qquad (D_{NP3sgM})$$

In general, though, we are interested in the most general feature structure of this type—the one that contains all the information from the unified feature structures but no additional information. In formal terms, we define the *unification* of two feature structures D' and D'' as the most general feature structure D, such that $D' \sqsubseteq D$ and $D'' \sqsubseteq D$. We notate this $D = D' \sqcup D''$.

As we have seen, not all pairs of feature structures can be unified in this way; they may contain conflicting information. In this case, unification is said to *fail*.

The following examples may facilitate the intuition of unification as an information-combining function:

(Unification adds information.)

$$\begin{bmatrix} cat: & np \end{bmatrix}$$
$$\sqcup \begin{bmatrix} agreement: & \begin{bmatrix} number: & singular \end{bmatrix} \end{bmatrix}$$
$$= \begin{bmatrix} cat: & np \\ agreement: & \begin{bmatrix} number: & singular \end{bmatrix} \end{bmatrix}$$

(Unification is idempotent.)

$$\begin{bmatrix} cat: & np \end{bmatrix}$$
$$\sqcup \begin{bmatrix} cat: & np \\ agreement: & \begin{bmatrix} number: & singular \end{bmatrix} \end{bmatrix}$$
$$= \begin{bmatrix} cat: & np \\ agreement: & \begin{bmatrix} number: & singular \end{bmatrix} \end{bmatrix}$$

(Variables are unification identity elements.)

$$[\ ]$$
$$\sqcup \begin{bmatrix} cat: & np \\ agreement: & \begin{bmatrix} number: & singular \end{bmatrix} \end{bmatrix}$$
$$= \begin{bmatrix} cat: & np \\ agreement: & \begin{bmatrix} number: & singular \end{bmatrix} \end{bmatrix}$$

(Unification acts differently depending on whether values are similar or identical.)

$$\begin{bmatrix} agreement: & \begin{bmatrix} number: & singular \end{bmatrix} \\ subject: & \begin{bmatrix} agreement: & \begin{bmatrix} number: & singular \end{bmatrix} \end{bmatrix} \end{bmatrix}$$
$$\sqcup \begin{bmatrix} subject: & \begin{bmatrix} agreement: & \begin{bmatrix} person: & third \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} agreement: \begin{bmatrix} number: singular \end{bmatrix} \\ subject: \quad \begin{bmatrix} agreement: \begin{bmatrix} number: singular \\ person: \ third \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} agreement: \boxed{1} \begin{bmatrix} number: singular \end{bmatrix} \\ subject: \quad \begin{bmatrix} agreement: \boxed{1} \end{bmatrix} \end{bmatrix}$$

$$\sqcup \begin{bmatrix} subject: \begin{bmatrix} agreement: \begin{bmatrix} person: third \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} agreement: \boxed{1} \begin{bmatrix} number: singular \\ person: \ third \end{bmatrix} \\ subject: \quad \begin{bmatrix} agreement: \boxed{1} \end{bmatrix} \end{bmatrix}$$

This last example is crucial in illustrating the important role of reentrancy in unification. In the example, we have used unification to add information about the agreement features of the subject[3] of a phrase, and in so doing, because of the reentry in the feature structure, we have concluded information about the agreement features of the phrase itself. Such examples will play an increasingly significant role in the sections to come.

3.2.4. Feature Structures as Graphs. Feature structures can be viewed as rooted, directed, acyclic[4] graph structures (from which the term "dag" is derived as an acronym) whose arcs are labeled with feature names. Each arc points to another such dag or an atomic symbol.

The feature structure $D'_{NP3sgSubj}$ would be expressed in a graph-structural notation as in Figure 1. Reentry in the graph corresponds to coindexing in

[3]We use the linguistic term *subject* here and later in an informal manner. This usage is important to distinguish from its technical use in such theories as LFG and relational grammar. It should be clear when we are using the word informally (as here) or in its technical application (as in discussions of LFG, Section 4.5).

[4]Note that certain implementations allow cyclic graph structures, i.e., directed graphs (dgs) in which a descendant dg has a feature whose value is the dg itself. These can be useful for modeling the *variable labels* of LFG, as in equations of the form (↑ (↓ *pcase*)) =↓.

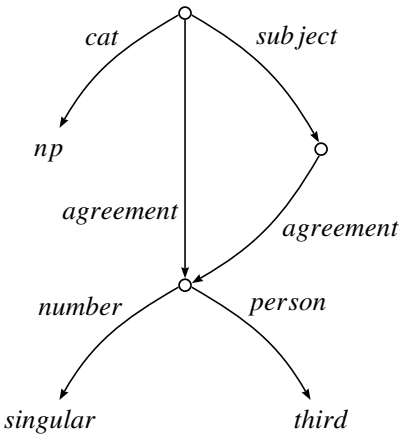


FIGURE 1

feature matrix notation. Underlying the graph-theoretic view is a twofold rationale. First, graph theory provides a simple and mathematically well-defined vocabulary with which to model the various feature systems of linguistic theories. Second, it leads to a coherent framework for investigating potential structure-combining operations.

Such operations on graph structures abound. Notions of unification, generalization, disjunction, negation, overwriting, and other more idiosyncratic operations can all be formally defined.

3.3. Combinatory Rules

Having characterized the informational domain, elements of which are associated with strings, we now need to describe how the inductive rules for building up the association can be represented. Rules must describe two things.

- How strings are concatenated to form larger strings.
- How the associated feature structures are related.

The former can be described with an abstraction of a context-free rule. The latter is cast in terms of identities among subparts of the associated feature structures. For instance, consider the following grammar rule:[5]

$$
\begin{aligned}
&X_0 \rightarrow X_1\ X_2 \\
&\quad \langle X_0\ cat \rangle\ = s \\
&\quad \langle X_1\ cat \rangle\ = np \\
&\quad \langle X_2\ cat \rangle\ = vp \\
&\quad \langle X_0\ head \rangle\ = \langle X_2\ head \rangle \\
&\quad \langle X_0\ head\ subject \rangle\ = \langle X_1\ head \rangle
\end{aligned}
\qquad (R_1)
$$

The context-free portion states that the constraint applies among three constituents—the string associated with the first being the concatenation of that associated with the second and third, in that order. In addition, it requires that the values for the *cat* features of the constituents be S, NP, and VP, respectively. The next identity requires that the value of the *head* feature associated with the VP be identical to the *head* of the S. Finally, the *subject* of the S is identical to the *head* of the NP.

For these identities to hold, the *head* value associated with the NP would have to be compatible with the VP's *subject* feature. In other words, the NP fills the role of the VP's subject.

As an example of string/feature-structure pairs admitted by this rule, consider the following pairings:[6]

$$
\textit{Uther sleeps} \longmapsto \begin{bmatrix} cat: & S \\ head: \boxed{1} & \begin{bmatrix} form: & finite \\ subject: \boxed{2} & \begin{bmatrix} agreement: & \begin{bmatrix} number: & singular \\ person: & third \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}
$$

$$
\textit{Uther} \longmapsto \begin{bmatrix} cat: & NP \\ head: \boxed{2} & \end{bmatrix}
$$

[5]The reader will notice an inconsistency of notation here in the paths used in rules. A less abusive notation, more consistent with Section 3.2.1, would use $X_0(\langle head\ subj \rangle)$ rather than $\langle X_0\ head\ subj \rangle$. The notation employed here is simpler, however, and has historical precedent.

[6]We use the symbol $\longmapsto$ to notate pairings between strings and their associated feature structures. For instance, lexical pairings will later be thus notated.

$$sleeps \longmapsto \begin{bmatrix} cat\colon & VP \\ head\colon & \boxed{1} \end{bmatrix}$$

3.3.1. Identity and Unification Revisited. Note that the pairings shown above are precisely those one would get by starting with the following simpler feature structures for X_1 and X_2

$$\begin{bmatrix} cat\colon & NP \\ head\colon & \begin{bmatrix} agreement\colon & \begin{bmatrix} number\colon singular \\ person\colon\ third \end{bmatrix} \end{bmatrix} \end{bmatrix} \qquad (D_{X_1})$$

$$\begin{bmatrix} cat\colon & VP \\ head\colon & \begin{bmatrix} form\colon & finite \\ subject\colon & \begin{bmatrix} agreement\colon & \begin{bmatrix} number\colon singular \\ person\colon\ third \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \qquad (D_{X_2})$$

and then interpreting the identity statements in the rule as *instructions to replace the substructures with their unifications* in the three feature structures associated with the three phrases. This replacement process is conventionally (and ambiguously) referred to as *unifying* the substructures.[7] Note that after two substructures have been unified in this sense, a common value is thereby introduced, because the process involves replacement by the same, not merely similar, feature structures. In fact, for this reason, reentrant feature structures and unifications can be thought of as duals of one another.

These simpler feature structures can in turn be derived from smaller phrases by performing destructive unifications associated with other combinatory rules. The basis step for this recursive process rests, of course, in the lexicon, which holds the primitive string/feature-structure pairings.

This systematic relationship between static identity and dynamic unification is taken advantage of extensively. It allows a declarative formalism to have a procedural interpretation. This interpretation is pervasive in unification-based formalisms. Often we can think of proceeding bottom up, combining

[7]Sometimes the process is referred to as *destructive unification* to differentiate it from the algebraic relation on feature structures that was first introduced.

subphrases by concatenation *and their informational elements by destructive unification* thereby yielding the minimal feature structures that conform to all the identities. Because this type of interpretation exists, such identities in rules are often referred to as unifications. We will see further examples of this interpretation in Section 3.4.

However, it is important to keep in mind that the bottom-up interpretation is not the only, or even the principal, way of viewing unification-based grammars. As discussed in Section 5.1, there is nothing inherently directional (bottom-up, top-down, left-to-right, or otherwise) about this or most unification-based formalisms. Indeed, this property of order-independence, declarativeness, or nonprocedurality, when taken together with the existence of procedural, computational interpretations such as the bottom-up view just sketched, is regarded as a prime advantage of the unification-based formalisms. It is only with considerable trepidation that inherently procedural mechanisms (such as some of the devices discussed in Section 4) are added to the formalisms.

3.3.2. Notational Sugar. Before going on, we introduce one useful notational convention. We can conventionally eliminate unifications for the special feature *cat* (the major category feature), instead recording this information implicitly by using it in the "name" of the constituent, e.g.,

$$\begin{aligned} &S \rightarrow NP\ VP \\ &\quad \langle S\ head \rangle = \langle VP\ head \rangle \\ &\quad \langle S\ head\ subject \rangle = \langle NP\ head \rangle \end{aligned} \qquad (R_1)$$

Whenever a constituent name is other than a (possibly subscripted) X, it is to be conventionally interpreted as representing the value of the *cat* feature for that constituent.

3.4. Some PATR-II Grammars

The unification-based grammar formalism just described is the PATR-II formalism, developed as a powerful, simple, least common denominator of the various unification-based formalisms. In Chapter 0 we will use PATR-II as a starting point for describing these other formalisms. First, however, we describe how some simple natural-language constructions can be encoded in

PATR-II grammars. The examples will perforce be oversimplified. However, the bibliography includes references to papers with more extensive analyses.

We will introduce three increasingly complex sample grammars to handle the following three constructs:

- *Agreement* of subject and verb for person and number.
- *Subcategorization* of verbs for particular postverbal complements.
- *Semantics* of sentences, expressed as encodings of logical forms.

As a side note, all three of the grammars developed here are listed in their entirety in Appendix 0 in the machine-interpretable form used by the PATR-II software.

3.4.1. Sample Grammar One: Agreement. The grammar rule R_1, presented earlier and repeated here, forms the basis of our first grammar.

$$\begin{aligned} &S \rightarrow NP\ VP \\ &\quad \langle S\ head \rangle = \langle VP\ head \rangle \\ &\quad \langle S\ head\ subject \rangle = \langle NP\ head \rangle \end{aligned} \tag{R_1}$$

When this rule is combined with one for identifying the head features of a verb phrase with its head verb, i.e.,

$$\begin{aligned} &VP \rightarrow V \\ &\quad \langle VP\ head \rangle = \langle V\ head \rangle \end{aligned} \tag{R_2}$$

and with a lexicon associating some words with corresponding feature structures, such as

$$Uther \longmapsto \begin{bmatrix} cat: & NP \\ head: & \begin{bmatrix} agreement: & \begin{bmatrix} number: & singular \\ person: & third \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$sleeps \longmapsto \begin{bmatrix} cat: & V \\ head: & \begin{bmatrix} form: & finite \\ subject: & \begin{bmatrix} agreement: & \begin{bmatrix} number: & singular \\ person: & third \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$sleep \longmapsto \begin{bmatrix} cat: & V \\ head: & \begin{bmatrix} form: & finite \\ subject: & \left[agreement: \left[number: plural \right] \right] \end{bmatrix} \end{bmatrix}$$

and so on, the grammar thus formed admits the sentence "Uther sleeps" but fails (properly) to admit "Uther sleep." We can see this failure by attempting to build the phrase bottom-up. The verb "sleep" participates in R_2, giving us a VP with the identical (because unified) *head* features; thus

$$sleep \longmapsto \begin{bmatrix} cat: & VP \\ head: & \begin{bmatrix} form: & finite \\ subject: & \left[agreement: \left[number: plural \right] \right] \end{bmatrix} \end{bmatrix}$$

This phrase/feature-structure pair plus the lexical pair for "Uther" participate potentially in R_1. Unifying the *cat* features of the three constituents mutually being constructed with S, NP, and VP, respectively, succeeds yielding

$$Uther\ sleep \longmapsto \left[cat: S \right]$$

$$Uther \longmapsto as\ before$$

$$sleep \longmapsto as\ before$$

Unifying the head features of the S and VP results in

$$Uther\ sleep \longmapsto \begin{bmatrix} cat: & S \\ head: & \boxed{1} \begin{bmatrix} form: & finite \\ subject: & \left[agreement: \left[number: plural \right] \right] \end{bmatrix} \end{bmatrix}$$

$$Uther \longmapsto as\ before$$

$$sleep \longmapsto \begin{bmatrix} cat: & VP \\ head: & \boxed{1} \end{bmatrix}$$

But now the final unification fails. There is no feature structure that unifies $\langle S\ head\ subject \rangle$ and $\langle NP\ head \rangle$. Since the unification fails, so does the rule application, and the string "Uther sleep" is not admitted. Note that the cause of

the failure, the clash of number features on subject and verb, corresponds intuitively to the number disagreement in the sentence. Also, this failure would have occurred regardless of the order in which the unifications had been applied. It is in this sense that the formalism is order-independent.

In future examples, we will not go into such great detail in charting derivations and our terminology will become considerably looser. *Caveat lector*.

3.4.2. Sample Grammar Two: Subcategorization. The astute reader will notice that the first sample grammar allowed no postverbal complements of verbs, a considerable limitation. Our second grammar deals with the problem of lexical selection of postverbal "subcategorization frames"—the manner in which, for example, the verb "storm" (as in "Uther storms Cornwall") lexically selects (*subcategorizes for*) a single postverbal NP, whereas "persuade" subcategorizes for an NP and an infinitival VP as complements.

A simple solution would be to add rules of the form

$$\begin{aligned} &VP \rightarrow V\ NP \\ &\quad \langle VP\ head \rangle\ = \langle V\ head \rangle \end{aligned} \tag{R_3}$$

and

$$\begin{aligned} &VP_1 \rightarrow V\ NP\ VP_2 \\ &\quad \langle VP_1\ head \rangle\ = \langle V\ head \rangle \\ &\quad \langle VP_2\ head\ form \rangle\ = \textit{infinitival} \end{aligned} \tag{R_4}$$

and so on, one for each subcategorization frame. The problem of matching up verbs with a VP rule could be achieved (as usual) with unification. A feature *subcat* in the verb's feature structure would be forced to unify with an arbitrary value specified in the rule. The following rules and lexical entries achieve such a matching.

$$\begin{aligned} &VP \rightarrow V\ NP \\ &\quad \langle VP\ head \rangle\ = \langle V\ head \rangle \\ &\quad \langle V\ subcat \rangle\ = np \end{aligned} \tag{R'_3}$$

$$\begin{aligned} &VP_1 \rightarrow V\ NP\ VP_2 \\ &\quad \langle VP_1\ head \rangle\ = \langle V\ head \rangle \\ &\quad \langle VP_2\ head\ form \rangle\ = \textit{infinitival} \\ &\quad \langle V\ subcat \rangle\ = npinf \end{aligned} \tag{R'_4}$$

$$storms \longmapsto \begin{bmatrix} cat: & V \\ head: & \begin{bmatrix} form: & finite \\ subject: & \begin{bmatrix} agreement: & \begin{bmatrix} number: & singular \\ person: & third \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ subcat: & np \end{bmatrix}$$

$$persuades \longmapsto \begin{bmatrix} cat: & V \\ head: & \begin{bmatrix} form: & finite \\ subject: & \begin{bmatrix} agreement: & \begin{bmatrix} number: & singular \\ person: & third \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ subcat: & npinf \end{bmatrix}$$

Early GPSG used this type of analysis with some forty basic verb phrase rules.

In the second grammar, we adopt a more radical approach that takes fuller advantage of the power of unification. Just as the first grammar had a "slot" for the subject NP complement, the second grammar uses slots for all the complements, both pre- and postverbal. This is achieved through the feature structure encoding of a list with features *first* and *rest* and end marker value *end*. The slots in the list correspond to the complements in the following order: postverbal complements from left to right, followed by the preverbal subject. For instance, the lexical entry for the verb "storms" would be given by the pairing

$$storms \longmapsto \begin{bmatrix} cat: & V \\ head: & \begin{bmatrix} form: & finite \end{bmatrix} \\ subcat: & \begin{bmatrix} first: & \begin{bmatrix} cat: & NP \end{bmatrix} \\ rest: & \begin{bmatrix} first: & \begin{bmatrix} cat: & NP \\ head: & \begin{bmatrix} agreement: & \begin{bmatrix} number: & singular \\ person: & third \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ rest: & end \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

while for the verb "persuades" we would have the even more complex pairing

$$
persuades \longmapsto \left[\begin{array}{l}
cat: \quad V \\
head: \left[form: finite\right] \\
subcat: \left[\begin{array}{l}
first: \left[cat: NP\right] \\
rest: \left[\begin{array}{l}
first: \left[\begin{array}{l} cat: \quad VP \\ head: \left[form: infinitival\right] \\ subcat: \left[rest: end\right] \end{array}\right] \\
rest: \left[\begin{array}{l} first: \left[\begin{array}{l} cat: \quad NP \\ head: \left[\begin{array}{l} agreement: \\ \quad \left[\begin{array}{l} number: singular \\ person: \ third \end{array}\right] \end{array}\right] \end{array}\right] \\ rest: end \end{array}\right]
\end{array}\right]
\end{array}\right]
$$

The convoluted *subcat* value here lists the complements of "persuades" as, in order, an NP (the object), a VP whose form is infinitival and whose subcategorization requirement is a single element list (i.e., only the subject is missing), and the subject NP itself (marked as third-person singular, to fold in the agreement conditions).

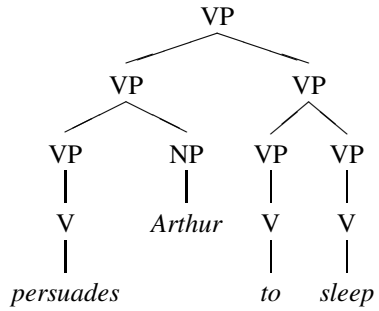
As each postverbal complement is concatenated onto the VP, its feature structure is unified with the next slot in the list. Thus, the verb can impose requirements on its complements—e.g., category requirements, or requiring a VP complement to be infinitival—by adding the appropriate features to the slot, as was done to ensure agreement with the subject in the previous grammar. Let us look at one such VP-forming rule that adds an NP complement to the VP.

$$
\begin{aligned}
&VP_1 \rightarrow VP_2\ NP \\
&\quad \langle VP_1\ head\rangle = \langle VP_2\ head\rangle \\
&\quad \langle VP_2\ subcat\ first\rangle = \langle NP\rangle \\
&\quad \langle VP_2\ subcat\ rest\rangle = \langle VP_1\ subcat\rangle
\end{aligned}
\qquad (R_5)
$$

The unifications in R_5 require, respectively, that

- Head features are shared by the VPs.
- The NP is unified with the first remaining slot in the subcategorization frame for the VP.
- The subcategorization frame for the newly formed VP is that of the shorter VP minus the first element just found.

The grammar therefore builds a left-recursive structure for verb phrases, so that, for instance, the phrases "persuades," "persuades Arthur," and "persuades Arthur to sleep" will all be VPs—the first subcategorizing an NP, a VP and a subject NP (as just seen in the foregoing lexical entry), the second a VP and a subject NP, and the third just the subject NP. The phrase structure for this final VP in accordance with this grammar would be



Of course, a similar rule would be required for VP complements.

$$
\begin{aligned}
&VP_1 \rightarrow VP_2\ VP_3 \\
&\quad \langle VP_1\ head \rangle = \langle VP_2\ head \rangle \\
&\quad \langle VP_2\ subcat\ first \rangle = \langle VP_3 \rangle \\
&\quad \langle VP_2\ subcat\ rest \rangle = \langle VP_1\ subcat \rangle
\end{aligned}
\qquad (R_6)
$$

How does this left recursion bottom out? We add a rule for just this purpose.

$$
\begin{aligned}
&VP \rightarrow V \\
&\quad \langle VP\ head \rangle = \langle V\ head \rangle \\
&\quad \langle VP\ subcat \rangle = \langle V\ subcat \rangle
\end{aligned}
\qquad (R_7)
$$

Finally, we need a rule to form sentences from NPs and *VPs whose subcategorization frame requires no more postverbal complements.* This latter condition is verified by unifying the *rest* of the frame with the end marker value *end*. We also unify the subject NP with the last remaining element in the frame.

$$
\begin{aligned}
&S \rightarrow NP\ VP \\
&\quad \langle S\ head \rangle\ = \langle VP\ head \rangle \\
&\quad \langle S\ head\ form \rangle\ = finite \\
&\quad \langle VP\ subcat\ first \rangle\ = \langle NP \rangle \\
&\quad \langle VP\ subcat\ rest \rangle\ = end
\end{aligned}
\qquad (R_8)
$$

A final optimization can be made. Rather than having separate rules for each possible category of postverbal complement, we can substitute the following single general rule:[8]

$$
\begin{aligned}
&VP_1 \rightarrow VP_2\ X \\
&\quad \langle VP_1\ head \rangle\ = \langle VP_2\ head \rangle \\
&\quad \langle VP_2\ subcat\ first \rangle\ = \langle X \rangle \\
&\quad \langle VP_2\ subcat\ rest \rangle\ = \langle VP_1\ subcat \rangle
\end{aligned}
\qquad (R_9)
$$

Thus instead of the forty basic rules (and the additional rules derived by metarule) that the early GPSG analysis postulated, we need just this one.

3.4.3. Sample Grammar Three: Logical Form. The simplicity and generality of the second sample grammar is such that the addition of a "semantics"—in the form of construction of logical forms—requires *no* changes in the grammar itself. Thus our third sample grammar differs only in the lexical pairings.

We add semantics by encoding logical forms using the features *pred* (for the predicate) and *arg-i* (for the *ith* argument). For instance, the logical expression we want to construct for the sentence "Uther storms Cornwall" might be

$$
\begin{bmatrix}
pred\colon storm \\
arg1\colon uther \\
arg2\colon cornwall
\end{bmatrix}
\qquad (D_{LF1})
$$

for "Uther persuades Arthur to sleep" we might have

[8]Note that this rule does not uniformly encode category information in the name of the nonterminal. For a discussion of the notational convention being assumed here refer to Section 3.3.2.

$$\begin{bmatrix} pred\text{: } persuade \\ arg1\text{: } uther \\ arg2\text{: } arthur \\ arg3\text{: } \begin{bmatrix} pred\text{: } sleep \\ arg1\text{: } arthur \end{bmatrix} \end{bmatrix} \qquad (D_{LF2})$$

and so on.

In this grammar, the encoded logical form will be the value of the feature structure's ⟨*head trans*⟩. For instance, the lexical entry for Uther will be

$$Uther \longmapsto \begin{bmatrix} cat\text{: } & NP \\ head\text{: } & \begin{bmatrix} agreement\text{: } & \begin{bmatrix} number\text{: } singular \\ person\text{: } third \end{bmatrix} \\ trans\text{: } & uther \end{bmatrix} \end{bmatrix}$$

Now, what will the lexical entry for a verb like "storms" look like—in particular, its semantics? Part of it is clear: the *pred* feature value should be *storm*. But what of the arguments? All that is known of the argument values is that they should be the respective translations of the various complements of the verb, whatever those might be. We can state this directly by identifying (unifying) the value for *arg1* with the translation of the subject slot in the verb's *subcat* frame, and the *arg2* with the translation of the object slot.

$$
storms \longmapsto \begin{bmatrix}
cat : & V \\
head : & \begin{bmatrix} form : finite \\ trans : \begin{bmatrix} pred : storm \\ arg1 : \boxed{2}\,[\ ] \\ arg2 : \boxed{1}\,[\ ] \end{bmatrix} \end{bmatrix} \\
subcat : & \begin{bmatrix} first : \begin{bmatrix} cat : & NP \\ head : & \begin{bmatrix} trans : & \boxed{1} \end{bmatrix} \end{bmatrix} \\ rest : \begin{bmatrix} first : \begin{bmatrix} cat : & NP \\ head : & \begin{bmatrix} agreement : & \begin{bmatrix} number : singular \\ person : & third \end{bmatrix} \\ trans : & \boxed{2} \end{bmatrix} \end{bmatrix} \\ rest : end \end{bmatrix} \end{bmatrix}
\end{bmatrix}
$$

Readers can convince themselves that the third grammar, which includes these types of lexical entries, derives appropriate logical forms for various sentences.

Note the importance of the *trans* feature's being under *head*. Because of the sharing of *head* features, the *trans* of the S will be identified with the *trans* of the head verb of the sentence, which is exactly what we want. Along the way of course, unifications of complements with the elements of the subcategorization frame will cause the arguments of the verb's translation to be filled in incrementally.

3.4.4. Coverage of the Sample Grammars. By suitable lexical definition, the grammar just developed has been shown to cover

- *agreement* of subject and verb,
- *subcategorization* for NPs and VPs of all types, and
- *logical form* construction.

Furthermore, the same grammar could be used for

- *auxiliaries*,
- *subcategorization* for Ss,

- *control*, and
- *equi* and *raising* verbs.

This might give the reader some awareness of the power of lexically oriented systems, as well as for the conciseness that unification makes possible. Most of the unification-based formalisms—LFG, PATR-II, and HPSG especially—tend to use this lexically oriented style of analysis, proposed originally by LFG.

3.4.5. Encoding Lexical Generalizations. The simplicity of these grammars stems from their use of complex lexical structures.[9] Although the grammars are thus kept simple, we now have the problem of dealing with unwieldy lexical feature structures, such as those encountered in the previous section. Clearly, no one is willing to write such complex and redundant feature structures for each lexical entry. There are two solutions to this problem.

First, we can come up with general techniques for expressing lexical generalizations so as to allow lexical entries to be written in a compact notation. We will defer further discussion of such techniques for lexical organization to the end of the following chapter, Section 4.8. Suffice it to say that such devices as templates and lexical rules in PATR-II, lexical-redundancy rules in LFG, and default inheritance in HPSG serve just this purpose: to express lexical generalizations. As grammars rely more and more on complex lexical encodings (as is the current trend in unification-based formalisms) these techniques become increasingly important.

Second, the formalism can be extended in various ways to be more expressive: special features with complex behavior in terms of unification can be added thereby leveraging the unification notion. Much of the following chapter on extended formalisms will have just this quality.

[9]Note that it does not arise from the use of unification itself. One could certainly write unification-based grammars that lack the lexical orientation of this third sample grammar, thereby putting more complexity in the rules and less in the lexicon.

CHAPTER 4

Extended Formalisms

4.1. Introduction

For various reasons including those just discussed, formalisms under the unification rubric typically include devices other than the rudiments found in PATR-II. In this section, we discuss several such extensions, organizing the discussion around the various formalisms in which they are found. At the same time, we will be presenting the notations and technical restrictions of the formalisms as well. Amidst all this verbiage, the reader should keep in mind that often, in fact usually, these techniques, notations, and restrictions are separable. A formalism designer can "mix and match" the various components in order to create an individually tailored tool.

The decision to present PATR-II before the other formalisms was based on expository convenience, it being by far the simplest of the lot. Justice could not have been done to any of the others in so short a time, nor can it be in the sections to follow. Especially lacking are the motivations behind the designs of the various formalisms, since we concentrate on comparing their formal and computational characteristics. For this reason, these sections may seem sterile from a linguistic viewpoint. The reader is therefore urged to read the cited articles for a more expansive presentation.

4.2. Two Classes of Formalisms

However, we can give an overview of some of the general motivations underlying certain aspects of the formalisms. The formalisms fall into two main classes: those designed as linguistic *tools* and those intended to be linguistic *theories*. As mentioned in Section 2.1, these goals are often at odds with each other, especially in the area of expressivity. Indeed, these differences are manifested in the types of extensions the formalisms include. Formalisms of the tool type (e.g., PATR-II, FUG, DCG) typically possess very general mechanisms for increasing expressive power, whereas those of the theory type (e.g., LFG,

GPSG) tend to incorporate devices designed for very specific purposes related to the particular type of linguistic analysis they prefer. Whereas understanding the design choices of the first type is comparatively independent of an understanding of the linguistic analyses embedded in their grammars, understanding the design choices of the second type is intimately tied to a comprehension of their linguistic analyses, for these choices have been claimed to embody universal linguistic principles. Consequently, we will be unable to present much detailed description for the second type of framework. Fortunately, detailed volumes are now available for both of the main unification-based linguistic theories.

Our discussion will begin with the tool-oriented formalisms.

4.3. Functional Unification Grammar

Functional unification grammar (previously called unification grammar, and, even earlier, functional grammar) was designed by Martin Kay as a general linguistic tool using unification as its only operation. Kay's motivations were twofold: first, to maintain a computational aspect to the formalism; second, to allow structural and functional notions to work side by side in the formalism. FUG uses several innovations to extend the expressivity of the formalism in a general way. We will discuss three of them, namely,

- patterns and constituent sets,
- disjunction, and
- *ANY* values.

4.3.1. Patterns and Constituent Sets. As mentioned in Section 3.3.1, shared values in feature structures and unifications are two sides of the same coin. Whereas PATR-II opts for notating the unification side of the coin in its rules, FUG uses reentrant feature structures exclusively. Thus an FUG rule is merely a feature structure (or functional structure, as it is called in the local jargon), with shared values where PATR-II would have unifications. For instance, the rule R_1 would be expressed in FUG as

$$\begin{bmatrix} cat: & S \\ head: & \boxed{1}\begin{bmatrix} subject: & \boxed{2} \end{bmatrix} \\ subj: & \begin{bmatrix} cat: & NP \\ head: & \boxed{2} \end{bmatrix} \\ pred: & \begin{bmatrix} cat: & VP \\ head: & \boxed{1} \end{bmatrix} \end{bmatrix} \qquad (R_1)$$

where *subj* and *pred* are the *functions* that will be unified with the subject noun phrase and predicate verb phrase, respectively.

But how is this last bit of information—the identities of the constituent functions—to be notated? FUG uses a special feature for this, the *constituent set* or *cset*. Its value is a set of feature structures, which are the components of the rule that are to be associated with subconstituents. Thus, we have

$$\begin{bmatrix} cset: & \{\boxed{3}\ \boxed{4}\} \\ cat: & S \\ head: & \boxed{1}\begin{bmatrix} subject: & \boxed{2} \end{bmatrix} \\ subj: & \boxed{3}\begin{bmatrix} cat: & NP \\ head: & \boxed{2} \end{bmatrix} \\ pred: & \boxed{4}\begin{bmatrix} cat: & VP \\ head: & \boxed{1} \end{bmatrix} \end{bmatrix} \qquad (R_1)$$

Finally, we must have a way of knowing how the strings associated with these subconstituents are composed to form the whole constituent. For this purpose, another new feature *pattern* is used, whose value is a sequence of the members of the constituent set; the order in the sequence corresponds to the concatenation order of the subconstituent strings.

$$\begin{bmatrix} cset: & \{\boxed{3}\ \boxed{4}\} \\ pattern: & \langle \boxed{3}\ \boxed{4} \rangle \\ cat: & S \\ head: & \boxed{1}\left[subject: \boxed{2} \right] \\ subj: & \boxed{3}\begin{bmatrix} cat: & NP \\ head: & \boxed{2} \end{bmatrix} \\ pred: & \boxed{4}\begin{bmatrix} cat: & VP \\ head: & \boxed{1} \end{bmatrix} \end{bmatrix} \qquad (R_1)$$

Actually, FUG uses a slightly different notation for functional structures, eschewing coindexing boxes for path specifications to mark reentrancy, using equal signs rather than colons, and notating sets and sequences with parentheses.

$$\begin{bmatrix} cset = (\langle subj \rangle \langle pred \rangle) \\ pattern = (\langle subj \rangle \langle pred \rangle) \\ cat = S \\ head = \left[subject = \langle subj\ head \rangle \right] \\ subj = \left[cat = NP \right] \\ pred = \begin{bmatrix} cat = VP \\ head = \langle head \rangle \end{bmatrix} \end{bmatrix} \qquad (R_1)$$

The example we have chosen is quite artificial, for, if we had been writing this rule from scratch in FUG, we would have taken more direct advantage of csets and patterns. A simpler formulation of the same idea, if not the same rule, is

$$\begin{bmatrix} cset = (\langle head\ subject\rangle\langle pred\rangle) \\ pattern = (\langle head\ subject\rangle\langle pred\rangle) \\ cat = S \\ head = \left[subject = \left[cat = NP \right] \right] \\ pred = \begin{bmatrix} cat = VP \\ head = \langle head\rangle \end{bmatrix} \end{bmatrix} \qquad (R_1)$$

Here we have exploited the fact that paths in the csets need not be top-level.

Thus, FUG incorporates in a single feature structure all the information found in the various parts of a PATR-II rule. It does this by introducing certain features (*cset, pattern*) with *special interpretations*. Unification of functional structures must be extended to handle these new features appropriately. This theme of extending formalisms by adding features with special interpretations in their unificational behavior is a common one which we will see recurring in subsequent discussion.

4.3.2. Disjunction. FUG takes seriously the concept of functional structures as the sole repositories of linguistic information. We have seen one instance of this, the idea that rules are not different kinds of things than functional structures. What does FUG do then with the concept of a grammar? How is the grammar to be interpreted as just another functional structure?

The problem is that the parts of a rule, like the parts of a functional structure, operate *conjunctively*—that is, they impose constraints that *all* have to be satisfied simultaneously. But a grammar is disjunctive. For a string to be admitted by a grammar, only *one* of the rules need be satisfied. We need a way of introducing *disjunction* into functional structures.

FUG incorporates a special notation for disjunction (or, as it is called in the FUG literature, *alternation*) in functional structures. Enclosing a set of (normal, conjunctive) functional structures within braces ('{}') is interpreted to mean that only one of the set of functional structures need be unified with. A grammar, then, is just a disjunctive set of functional structures each corresponding to a rule.

This type of general disjunction can be differentiated from so-called *value disjunction*, in which a feature is given a disjunctive specification as its value. Once we have both types of disjunction, we can use them in all sorts of ways,

since we can have disjunctions embedded within functional structures which are themselves embedded within disjunctions, and so forth. For instance, we might want to say that the present tense verb "storm" is either plural or first- or second-person singular. We can do this with the lexical pairing

$$storm \longmapsto \left[\begin{array}{ll} cat: & V \\ tense: & present \\ subject: & \left[\begin{array}{l} agreement: \left\{\begin{array}{l} \left[\begin{array}{ll} number: & singular \\ person: & \{first\ second\} \end{array}\right] \\ \left[number: plural\right] \end{array}\right\} \end{array}\right] \end{array}\right]$$

Again, the notion of unification of functional structures must be extended to handle value and/or general disjunction. Once this is done, the expressive power of disjunction is at our disposal.

4.3.3. *ANY* **Values.** The final FUG extension we discuss is not a feature but an atomic value with a special interpretation. The atomic value *ANY* in FUG has the following unificational behavior. *ANY* unifies with anything, just as a variable does. However, unlike variables, for a *final* functional structure to be well-formed, no *ANY*s may be present; they must each have unified with something else. The notion of final functional structure is crucial here. In FUG, it is intended to correspond to the functional structure of a whole sentence. One could imagine interpretations in which other definitions of "final" are employed.

A good example of the use of *ANY* is in encoding subcategorization facts. Suppose we wanted to use a "functional" approach to subcategorization, rather than the "sequence" approach used in the same PATR-II grammars. That is, we want to have features corresponding to various *grammatical functions* such as subject, direct object, indirect object, verbal complement, etc. This is just the extension of our first sample grammar to include grammatical functions other than the subject.

The reason we did not use this method for subcategorization in the PATR-II grammars presented earlier is simple. Although there is a convenient way of guaranteeing that a particular verb will not get, say, an indirect object—e.g., by putting in that verb's lexical entry the value *none* for the feature *indirect-object*—there is no convenient way of requiring that an indirect object be

present. This is exactly what *ANY* allows. By giving the value *ANY* to the feature *indirect-object* we require that something eventually unify with that feature—presumably the feature structure of an indirect object noun phrase. Of course, *ANY* can be used for other purposes; it is not restricted to handling subcategorization.

Because of this aberration in the definition of its unificational behavior—that it is a well-formed atomic value except in final functional structures—*ANY* can be viewed as being a nonmonotonic device. That is, a system with *ANY* values can have an ill-formed functional structure become well-formed through further unifications. In this sense, *ANY* violates the spirit of declarativeness, although it does so in such a weak way that we are likely to be willing to put up with it. Nonetheless, it raises an issue that warrants further attention. (See Section 5.1.)

4.4. Definite-Clause Grammars

Definite-clause grammars arose from work in Prolog by Pereira and Warren. DCG and its related formalisms (slot grammars, extraposition grammars, gapping grammars, modified structure grammars, etc.) all use a variety of unification based on *term structures* rather than feature structures. Term unification was originally developed for use in automatic theorem-proving, and was taken over by Prolog itself; it was therefore incorporated wholesale into DCGs.

4.4.1. Term Unification. Terms are the informational elements in DCG. A term (corresponding to a complex feature structure in PATR-II) is notated in the way familiar from logic and mathematics: a predicate symbol is followed by a parenthesized series of smaller terms. The basic terms are constants (corresponding to the atomic feature structures) and variables (corresponding to PATR-II variables). These are notated with lower- and uppercase strings, respectively, following the convention of Prolog. Reentrancy is notated by the sharing of variables.

For instance, the following are terms:

s(*head*(*SubjHead, Form*)))

np(*SubjHead*)

vp(*head*(*SubjHead,Form*)))

Terms differ from feature structures in two important ways.

- *Order:* Rather than identifying values by associating them with a feature, terms identify them by their linear order in the term structure.
- *Arity:* The number of elements in a term structure is significant in unification.

Thus, *agreement*(*third, singular*) will not unify with *agreement* (*singular, third*) because the order of arguments pairs *third* with *singular*, which do not unify. Also, it fails to unify with *agreement*(*third*) because the arities do not match. The first takes two arguments, whereas the second takes only one.

DCG rules, like those of PATR-II, use a context-free skeleton to associate the string-combining and information-combining operations. Rule R_1 would be expressed in DCG as

$$s(head(SubjHead,Form)) \rightarrow np(SubjHead), \quad vp(head(SubjHead,Form)). \qquad (R_1)$$

Note that, since only variables can be used to mark shared values, a clumsy encoding of the unification $\langle S\ head\rangle = \langle VP\ head\rangle$ was used that actually had to mention all the head features separately. Another disadvantage of term structures is that the lack of features labeling values increases the cognitive burden in interpreting the terms. A grammar writer must remember which argument positions correspond to which functions. Finally, since arity is significant, if a grammar writer wants to specify a value for one of the elements of a term, all the rest of the elements must be specified, at least by marking them as variables. This leads to grammars with lots of variables strewn across the rules.

Nonetheless, DCGs are quite useful for several reasons. First, because they can be run virtually directly as Prolog code, and efficient Prolog compilers exist, grammars written in DCG can be compiled directly into fast parsers.[1] The CHAT system (Pereira, 1983), a natural-language interface to a Prolog database, is capable of parsing a sentence, building a logical form translation, constructing a database query from the logical form and optimizing it,

[1]Often a criticism of DCG is made that because of the standard form of compilation into Prolog, DCG cannot handle left-recursive rules. This is actually misleading. The formalism itself can of course state left-recursive rules with no difficulty. Furthermore, compilation techniques (such as the BUP method (Matsumoto et al., 1983)) exist that can compile DCG into Prolog in such a way that enables left-recursive rules to be handled.

and retrieving the answer from a database in a matter of hundreds of milliseconds. Few if any of the current natural-language systems can match it in pure speed and performance. Second, because of the tight relationship between DCG and Prolog, DCG grammars can be easily integrated with programs written in Prolog. Consequently, integrating a DCG natural-language system with some other program, say, a database or expert system, can be quite straightforward. Finally, the human-engineering problems associated with the linear order and arity requirements of term unification are not terribly burdensome for small to medium-sized grammars. Thus, for the rapid development of simple and efficient natural-language systems, DCG can be the formalism of choice.

4.5. Lexical-Functional Grammar

In this section and the next we discuss two unification-based formalisms that were designed from a different perspective—as linguistic theories rather than tools. The first one, lexical-functional grammar, was developed as a theory of language with special emphasis on the mental representation of grammatical constructs, and on universal constraints on natural languages. LFG takes as primitive a notion of *grammatical function*; its style of analysis is delineated first by formal extensions that facilitate such a functional type of linguistic encoding, and second by a *substantive theory* of linguistic universals that are stated not as constraints on the formalism, but as universal claims about languages.

Bresnan (1982) has presented the LFG formalism and its extensive linguistic motivation. We merely discuss some of the formal highlights and describe their usage in lexical-functional grammars.

4.5.1. Notational Differences. First, we present some relatively minor notational changes. In place of "names" for the various constituents in rules and their paired feature structures (called f-structures in LFG and in the remainder of this section), LFG uses syntactic metavariables notated as arrows. Associated with a constituent in a rule, say the NP in R_1, a unification such as $(\uparrow\ subj) = \downarrow$ would correspond to the PATR-II unification $\langle S\ subj\rangle = \langle NP\rangle$. That is, the $\uparrow$ refers to the parent f-structure and the $\downarrow$ refers to the f-structure of the child with which the unification is associated. Thus, our rule R_1 might be expressed in LFG as

$$\begin{array}{lll} S \rightarrow & NP & VP \\ & (\uparrow head\ subj) = (\downarrow head) & (\uparrow head) = (\downarrow head) \end{array} \qquad (R_1)$$

Here again, we can write such a rule much more effectively by making use of some additional differences in LFG. The f-structures associated with constituents are not thought of as including the category information (as they are in the earlier formalisms). Alternatively, we can view the metavariables ↑ and ↓ as only referring to the noncategorial part of the information structure associated with the constituent. Thus, we have no need for distinguishing *cat* from *head* features. We can simply write

$$\begin{array}{lll} S \rightarrow & NP & VP \\ & (\uparrow subj)=\downarrow & \uparrow=\downarrow \end{array} \qquad (R_1)$$

4.5.2. Semantic Forms. One limitation of not including category information in f-structures is that a grammar cannot use this information in, say, subcategorization. In LFG, this is considered an advantage and licenses the addition of a special feature that handles both subcategorization and predicate-argument structure (similar to the constructed logical form of sample grammar three). This feature *pred* takes a special type of value, a *semantic form*. Semantic forms serve the purpose of the *subcat* list in the previous sample grammars, or the *cset* and *pattern* features in FUG, as the repository of subcategorization information. But they are also used to encode the predicate-argument structure. For instance, associated with a verb such as "storms" we might have the equations

$$(\uparrow tense) = present$$

$$(\uparrow pred) = \text{`}storm\langle(\uparrow subj)(\uparrow obj)\rangle\text{'}$$

The *pred* equation is such that it is satisfied only if

- all the grammatical functions referred to in the semantic form have values in the f-structure (i.e., the f-structure is *complete*), and
- no grammatical function other than these has a value in the f-structure (i.e., the f-structure is *coherent*).

Several comments are appropriate at this point. First, we require these conditions only of the final f-structure. Thus the coherence and completeness conditions play much the same role as the *ANY* of FUG. Second, this definition presupposes the notion of grammatical function. LFG postulates that a certain finite set of features constitutes the universal set of grammatical functions. Finally, observe that the semantic form contains the same information as the logical form construction in the lexical entries of Section 3.4.3. The

implicit embedding of semantic forms can, therefore, give us a representation of predicate-argument structure.

Thus the semantic-form value can be thought of as a compact way of representing subcategorization and semantic information. It leads to a particular style of analysis prevalent in LFG, one that involves subcategorization by grammatical function. The motivation for this style of analysis is based on arguments concerning linguistic universals that lie outside the scope of this introduction.

4.5.3. Other Devices. LFG contains many other devices that facilitate various linguistic analyses. Among these are

- *constraint equations* which, like *ANY*, are used to guarantee a value for a feature without specifying that value,
- *set values* which allow features to take sets of feature structures as their values,
- *disjunction*,
- *regular expressions* in the context-free skeleton,
- *long-distance metavariables* used in the analysis of Wh-movement and other unbounded dependencies,
- *lexical rules* for stating lexical generalizations about related words.

Conversely, LFG is restricted by a series of formal constraints, in addition to the constraints of the substantive theory. Constraints include

- *off-line parsability*, which disallows vacuous derivations in the context-free skeleton (thereby making the formalism decidable in the sense of complexity theory; see Section 5.1),
- *functional locality*, which disallows paths longer than two features to occur in rules,

and so forth.

4.5.4. Summary. The LFG formalism departs from the basic unification-based approach in two ways: first, by adding devices especially designed for its lexical- and grammatical-functional-oriented style of analysis, which is the basis for its claims regarding the psychological reality and universality of lexical-functional constructs; second, by imposing formal and substantive constraints so as to limit the formalism's expressive power, both computationally and linguistically. LFG has provided elegant analyses of constructions in a broad range of languages, with wildly different characteristic surface orderings, in so

doing revealing interesting properties of languages in general. Furthermore, LFG researchers have investigated implementational and psycholinguistic issues bearing on grammar formalisms and their relation to language-processing systems.

4.6. Generalized Phrase Structure Grammar

The second linguistic theory in the unification-based camp is generalized phrase structure grammar (GPSG). Like LFG, the motivation for many of the devices and notions peculiar to GPSG arose from its practitioners' particular linguistic perspective and style of analysis. GPSG grew out of an attempt to retain a formally restrictive system while handling a wide range of syntactic, and especially semantic, phenomena thought not to be possible in such a restricted system. For instance, GPSG described a method for handling unbounded dependencies without resorting to transformation of structures. Though the details have changed, the basic device of this analysis, the slash category, has persisted. With respect to semantics, GPSG built on the work of Montague, building a semantic component directly into the formalism. Current GPSG is considerably more complicated than the earlier systems, reflecting a more ambitious coverage of phenomena. Nonetheless, its formal restrictiveness still stands.

4.6.1. The Informational Domain. The GPSG informational domain is a highly restricted variant of feature structures. Recursivity in the feature structures (or categories, as we shall call them, following the GPSG practice) is severely constrained—so much so that there exist only a finite number of such structures. Furthermore, categories are required to satisfy certain grammar-dependent restrictions on the cooccurrence of features, which further limits the permissible categories.

Unifications or identities are also severely restricted in scope. There are no identities in rules at all. All identities follow from certain specialized principles governing the combinatorics of features, a topic which we will discuss in greater detail in Section 4.6.3.

4.6.2. The Combinatory Rules. GPSG grammar rules are quite different from those in the other formalisms. They are decomposed into several types:

- *Immediate dominance (ID) rules* which are similar to context-free rules except that they specify no ordering among the various sub-constituents.
- *Metarules* which capture redundancies among ID rules.[2]
- *Linear precedence (LP) rules* which specify the linear order for all sets of sibling constituents.
- *Other constraints* such as lexical and phrasal default feature values, and feature cooccurence restrictions.

The first and third of these constitute the *ID/LP format* for grammars.

4.6.3. Satisfying the Rules. A phrase structure tree satisfies the various rules of a GPSG grammar according to a quite complex condition on mutual satisfaction of the various rules listed above, along with certain grammar-independent principles that must also be satisfied. These principles are the workhorses of the theory, since they are the only source from which identifications among the various structural components of categories arise. The principles basically constitute specialized interpretations of certain special features as to their unificational behavior, just as principles of LFG specify specialized interpretations of the semantic forms and grammatical functions.

To give the reader an idea of the type of work the GPSG principles perform, we discuss just a few of them in informal terms.

- *The head feature convention*[3] (HFC) requires identity between the head features of a parent and its stipulated head child. Thus, since the VP is stipulated as the head of the S, this principle does roughly the job of the unification $\langle S\ head\rangle = \langle VP\ head\rangle$.
- *The control agreement principle* forces identity between the agreement features of a controller and controllee under suitable (semantical) definitions thereof. Such definitions would include a subject NP and its sibling VP for instance; consequently, in tandem with the HFC which identifies agreement features on the VP with those of the head V, the control agreement principle induces subject-verb agreement.

[2]Note that metarules in GPSG are giving way to lexical techniques. This trend has arisen from both computational and linguistic motivations.

[3]The head feature convention is so called for historical reasons. It is actually not a convention at all, although it was so construed at one point in the genesis of the theory.

- *The foot feature principle* governs identities of features involved in long-distance dependencies. Through this principle, phenomena such as Wh-movement, relative-clause formation and reflexivation are modeled.

4.6.4. GPSG Semantics. Early GPSG followed the Montagovian semantic tradition of separating the syntactic from the semantic information by keeping the two portions of a grammar separate but parallel in structure. Each syntactic rule was paired with a semantic rule, stated in an entirely separate type of formalism, intensional logic. Through these parallel rules, every grammatical phrase was given a denotation, a model-theoretic entity corresponding to the meaning of the phrase.

Current GPSG builds the denotations through one of its general principles, rather than individual rules. Although such principles are quite complex, they lead to elegant grammars in which semantic facts arise directly from the syntactic grammar.

4.6.5. Summary. The design of GPSG is based to a considerable degree on a detailed analysis of natural language semantics, building on its genesis from Montague semantics. It has been especially successful in dealing elegantly with the subtleties of coordination phenomena and long-distance dependencies. Moreover, all this has been accomplished within a framework that is quite weak in mathematical expressive power; unpublished work by the author suggests that even the most recent and elaborate version of GPSG may still be weakly context-free in power. It is thus by far the most formally constrained of the various unification-based formalisms.

4.7. Head Grammar and Head-Driven Phrase Structure Grammar

Two recent variants of GPSG were introduced by Pollard and subsequently developed by him and his associates at Hewlett-Packard as a result of their research into implementation of GPSG. The first formalism, head grammars, augmented GPSG by adding string-combining operations which Pollard called head-wrapping. Wrapping operations (though different from the particular formulation given by Pollard) appear earlier in the Montague grammar literature (Bach, 1980). Pollard used them to provide analyses of discontinuous constituents and certain complement control phenomena.

Based on GPSG and head grammars and inspired by FUG and LFG, head-driven phrase-structure grammar takes further advantage of the power of unification. HPSG reverts to allowing only concatenation, replaces the metarules of GPSG completely with lexical rules, and removes many of the restrictions yielding finiteness of the informational domain. Pairings (called *signs*) of strings and informational elements are determined by a bottom-up *rule application algorithm*; the formalism is therefore inherently procedural. Components of the algorithm correspond more or less closely to principles of GPSG, although several major differences in analysis should be noted. Paramount among these are the treatments of subcategorization and semantics, which, though notationally quite different, are remarkably similar in spirit to the analysis presented in the third sample grammar.

4.8. Lexical Organization

We return now to the issue raised in Section 3.4.5 concerning techniques for organizing lexical information. Recall that many of the formalisms discussed in this paper favor analyses that are lexically oriented—that is, they have simple rules of combination operating on lexical items with quite complex associated information structures. In such a case, the ability to organize the lexicon in such a way as to remove redundancies and encode generalizations is especially important. One need only refer to the lexical entries used in the third sample grammar to convince oneself that such techniques are indeed necessary.

We will concentrate on three quite general methods used frequently in these formalisms for encoding lexical generalizations: simple inheritance, default inheritance, and transformation of lexical information. A fourth sample grammar listed in the appendix demonstrates the use of these techniques within the PATR-II Experimental System.[4]

4.8.1. Simple Inheritance. Lexical entries often share much common structure. For instance, all verbs in our sample grammars share the following feature structure information:

$$\left[cat\colon v\ \right] \qquad (D_v)$$

Furthermore, all transitive verbs share this additional information:

[4]Note that in this grammar, unlike the previous grammars, the subcategorization frame is ordered with the subject first, rather than last. This aids the definition of various templates and lexical rules.

$$\left[\begin{array}{l} subcat: \left[\begin{array}{l} first: \left[cat: NP\right] \\ rest: \left[\begin{array}{l} first: \left[cat: NP\right] \\ rest: end \end{array}\right] \end{array}\right] \end{array}\right] \qquad (D_{trans})$$

Presumably we should like to eliminate this redundancy by merely asserting that a particular lexical item is transitive while having the information in D_{trans} stated only once as a general property of transitives. Furthermore, since all transitives are verbs, we should not have to include this fact in the lexicon, but rather, should inherit the verbal information D_v automatically. In PATR-II, this process of defining useful lexical abstractions of feature structures is manifested in the ability to define lexical *templates*, which are name-bearing feature structures that can be used in lexical entries. For instance, we might have a template called *Verb* corresponding to D_v. This could be defined in the following way:

Let *Verb* be $\langle cat \rangle = V$.

Similarly, the transitive notion can be abstracted as a template *Transitive*.

Let *Transitive* be $\langle subcat\ first\ cat \rangle = NP$
$\langle subcat\ rest\ first\ cat \rangle = NP$
$\langle subcat\ rest\ rest \rangle = end$
$\langle head\ trans\ arg1 \rangle = \langle subcat\ first\ head\ trans \rangle$
$\langle head\ trans\ arg2 \rangle = \langle subcat\ rest\ first\ head\ trans \rangle$

Alternatively, we can embed the *Verb* template hierarchically in the *Transitive* template to express the notion that transitives are verbs.

Let *Transitive* be *Verb*
$\langle subcat\ first\ cat \rangle = NP$
$\langle subcat\ rest\ first\ cat \rangle = NP$
$\langle subcat\ rest\ rest \rangle = end$
$\langle head\ trans\ arg1 \rangle = \langle subcat\ first\ head\ trans \rangle$
$\langle head\ trans\ arg2 \rangle = \langle subcat\ rest\ first\ head\ trans \rangle$

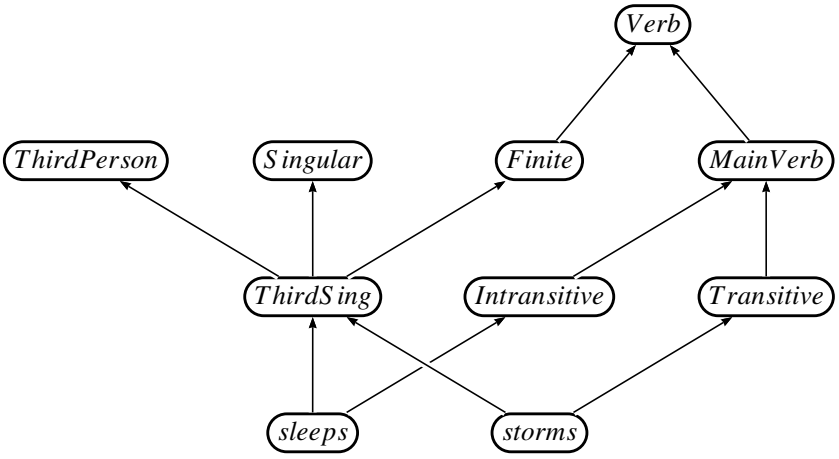


FIGURE 2

Since templates can be defined hierarchically, that is, in terms of other templates, this leads to a *structure-sharing* lexical organization.[5] In essence, we are building an *inheritance hierarchy* akin to those employed in many AI knowledge representation systems. We can make this similarity more explicit by using for our lexicon a node and arc notation familiar from AI work on semantic nets. Suppose we decide that at least part of our lexicon should be organized as in Figure 2. This would be accomplished with the following template definitions and lexical entries.

Let *Verb* be $\langle cat \rangle$ = *v.*

Let *Finite* be *Verb*
$\langle head\ form \rangle$ = *finite.*

Let *ThirdPerson* be $\langle subcat\ first\ head\ agreement\ person \rangle$ = *third.*

Let *Singular* be $\langle subcat\ first\ head\ agreement\ number \rangle$ = *singular.*

Let *ThirdSing* be *ThirdPerson*
Singular.

Let *MainVerb* be *Verb*
$\langle head\ aux \rangle$ = *false.*

[5] We are using the term "structure-sharing" here in a manner different from its use in terms like "structure-sharing dag representation methods," e.g., (Pereira, 1985; Karttunen and Kay, 1985).

Let *Transitive* be *MainVerb*
$\langle$*subcat first cat*$\rangle$= *NP*
$\langle$*subcat rest first cat*$\rangle$= *NP*
$\langle$*subcat rest rest*$\rangle$= *end*
$\langle$*head trans arg1*$\rangle$=$\langle$*subcat first head trans*$\rangle$
$\langle$*head trans arg2*$\rangle$=$\langle$*subcat rest first head trans*$\rangle$

Let *Intransitive* be *MainVerb*
$\langle$*subcat first cat*$\rangle$= *NP*
$\langle$*subcat rest*$\rangle$= *end*
$\langle$*head trans arg1*$\rangle$=$\langle$*subcat first head trans*$\rangle$

storms $\longmapsto$ *Transitive*
ThirdSing
$\langle$*head trans pred*$\rangle$ = *storm.*

sleeps $\longmapsto$ *Intransitive*
ThirdSing
$\langle$*head trans pred*$\rangle$ = *sleep.*

With these definitions, the simple lexical entries for "storms" and "sleeps" suffice to derive all of the feature structure information displayed in the earlier complex feature structures for these words. For instance, since "storms" includes the template *ThirdSing,* which itself includes *Finite*, in turn requiring that the *head*'s *form* feature be *finite,* the feature structure for the lexical entry will itself include this information; it is "inherited," so to speak, from the *Finite* template. The great power of templates and inheritance for simplifying lexicons is readily apparent.

Note the character of this organization. Just as in typical AI representation systems, the concepts defined are organized from most to least specific. The word "storms" is a type of third-person, singular verb which is a subclass of finite verbs, the latter itself a subclass of verbs in general.

Furthermore, a lexical entry or template can inherit information from more than one other template—that is, we allow *multiple inheritance*. One interesting special case of this phenomenon occurs in the network just defined. "Storms" inherits the *Verb* template because it is finite, but also because it is transitive and therefore a main verb. Since the information in all the various templates is combined by unification, such multiple inheritance of a single property presents no problems. Presumably, if contradictions were to arise in the unification of the assorted pieces of inherited information, the lexical

entry would simply be disallowed. Note that, because of the order independence of unification, the order in which we choose to traverse this network of inheritance in constructing the lexical entry's feature structure is completely immaterial; all traversals will generate the same feature structure. In the next section, however, we will see how this phenomenon of multiple inheritance can cause problems when information-combining methods other than unification are used.

Though the PATR-II system of templates first embodied an inheritance network organization for lexicons within a unification-based framework, it was the HPSG group at Hewlett-Packard that was most explicit in advocating such an approach (Flickinger et al., 1985). Their use of the HPRL knowledge representation language led directly to an inheritance-based organization for the HPSG lexicon. Besides simple inheritance, however, they also took advantage of another standard feature of knowledge representation languages—*default inheritance*.

4.8.2. Default Inheritance. We often want a lexical entry to inherit *most*, but not *all*, of the information associated with some node in the hierarchy. The simple inheritance method discussed in the last section does not allow for *exceptions* to be easily encoded. For instance, let us suppose that, rather than specifying in every verb entry whether the subject of the verb should be in the nominative or accusative case (depending on whether the verb is finite or nonfinite, respectively) we wanted to state the general fact that subjects of all except finite verbs are accusative.[6] We might try to do this with the following new templates.

Let *Verb* be $\langle cat \rangle = v$
$\langle subcat\ first\ head\ agreement\ case \rangle = accusative.$
Let *Finite* be *Verb*
$\langle subcat\ first\ head\ agreement\ case \rangle = nominative.$

But this merely leads to contradiction in the case of finite verbs. What we need instead is some way of defining a precedence of a certain segment of information over others. In default inheritance networks, the precedence is typically defined in the following manner: information lower in the network has precedence over information in higher nodes. Thus, if a lexical entry inherits the

[6]This example is taken from Flickinger et al. (1985), and is not being proposed as the preferred analysis of the phenomenon.

Verb template, in general it will take accusative subjects, *unless* some nodes lower in the template (such as *Finite*) requires otherwise. This approach to inheritance is utilized extensively by the builders of the HPSG lexicon.

A different approach to assigning precedence is allowed (albeit used sparingly) in the PATR-II lexical organization. The PATR-II Experimental System allows a method of combining structures by *overwriting* in addition to normal unification. Overwriting is a noncommutative operation akin to destructive unification except that, in the case of unification"clashes", one of the operands (say, the rightmost) is given precedence. Thus, unlike unification, overwriting never fails. For the example at hand, we could use overwriting in the template *Finite* to override the assignment of accusative case to the subject. The following template definition, which uses the symbol "=>" for overwriting, accomplishes this.

Let *Finite* be *Verb*

$\langle$*head agr case*$\rangle$ => *nominative.*

By using overwriting, or similar nonmonotonic operations, the effect of default feature inheritance (and many other even more unorthodox mechanisms) can be achieved in a lexicon. The cost of such a move is great, however, because the use of overwriting eliminates the order independence that is so advantageous a property in a formalism.

With either type of inheritance exception mechanism, a problem arises because of the interaction of exceptions and multiple inheritance. When a value is inherited through two chains of inheritance, but one of the chains overrides the value (either because a lower node has specified a different value or as a result of explicit overwriting), there is a question as to which (if either) of the two values to use. For example, the verb "storms" inherits accusative case for its subject from the *Verb* template in two ways—once through the *Intransitive* template, and again through its inclusion of the template *Finite;* in the latter instance, however, the template overrides the accusative specification with nominative case. Now, which of the various possible values—accusative or nominative—should be assigned as the case of the subject, or alternatively, should the conflict be construed as a failure of unification causing failure?

The answer depends roughly on the *order* in which the various constraints are imposed, whether or not both applications of the *Verb* template precede application of the *Finite* template. Put another way, the solution rests on assigning particular precedences to the various nodes in the hierarchy. Since

questions of just this sort have been wrestled with in the literature on default reasoning and default logics, we will not discuss the issue further here except to raise it as a relevant issue in the design of lexical mechanisms.

4.8.3. Transformation of Lexical Structure. Occasionally a more powerful mechanism is needed to represent systematic relationships among feature structures in the lexicon. Rather than merely amalgamating common elements of substructure, we often want to perform more complex transformations of feature structures in order to relate more disparate entries. There is a long tradition in linguistics of using *lexical rules* for just this purpose.

Within the context of unification-based approaches, LFG pioneered the use of lexical rules to express systematic relations among lexical items.[7] LFG's lexical rules are typically expressed as relations among (or transformations of) semantic forms (as described in Section 4.5.2). For instance, the LFG lexical rule for English passive[8] would be

$$(SUBJ) \rightarrow \phi/(BY\ OBJ)$$

$$(OBJ) \rightarrow (SUBJ)$$

which is intended to mean that the semantic form associated with an active and a corresponding passive are related by a transformation that takes the active form's subject into the by-object of the passive, and the object into the passive's subject. Alternatively, the subject of the active can be eliminated entirely in the passive. Thus, LFG's relation-changing lexical rules are one example of a method of transforming lexical structure for the purpose of stating systematic relationships among feature structures.

Lexical rules in the PATR-II system are construed as general transformations on feature structures, expressed in terms of unificational constraints on an input structure and its transformed output. For instance, the agentless passive rule of LFG would be described as a PATR-II lexical rule in the following manner:

Define *AgentlessPassive* as $\langle out\ subj \rangle = \langle in\ obj \rangle$
$\langle out\ obj \rangle = nil.$

[7] Often, in the LFG literature, these are referred to as *lexical redundancy rules* to highlight the fact that they are not applied in the syntactic derivation of sentences, but are merely statements expressing "patterns of redundancy that obtain among large but finite classes of lexical entries." (Kaplan and Bresnan, 1983, page 180)

[8] We follow the analysis of Kaplan and Bresnan (1983, page 9).

For the lexical entries of the sample grammars (especially the fourth, in which the subcategorization frame lists the subject first) a passive lexical rule might be expressed as follows:

Define *AgentlessPassive* as ⟨*out cat*⟩=⟨*in cat*⟩
⟨*out subcat*⟩=⟨*in subcat rest*⟩
⟨*out head*⟩=⟨*in head*⟩
⟨*out head form*⟩=>⟨*passiveparticiple*⟩

This rule could be used to build a passive lexical entry referred to as *out* from an active entry *in* such that the category feature information remains the same but the subcategorization frame has been modifed to remove the subject. The head feature information is also maintained, except (note the use of overwriting) that the form of the verb is marked as passive participle.

The HPSG system also makes extensive use of lexical rules as transformations on lexical information. Flickinger et al. (1985) discuss their particular formulation.

Once again, it should be noted that the introduction of lexical rules into the process of determining lexical information for a specific lexical item makes critical use of the notion of order of application, since arbitrary transformations of this sort are of course highly sensitive to the sequence in which they are applied.

4.8.4. Other Techniques of Lexical Organization. Other techniques have been proposed for more succinctly stating the lexical information associated with particular words in a language. Among these are

- *Abbreviatory conventions*, such as the GPSG use of $V[3]$ to stand for the more verbose $[[N-][V+][SUBCAT3]]$.
- *Feature specification defaults*, a variant of default feature inheritance used in the most recent version of GPSG (Gazdar et al., 1985), specify default values for features that receive no value by other means.
- *Feature cooccurrence restrictions* state constraints on the acceptable configurations of feature information and, as such, can serve to abbreviate lexical entries.

4.9. Other Extensions of Formalisms

Various other devices have been proposed for increasing the expressivity of formalisms. For instance, researchers have proposed augmenting unification with other operations or algebraic relations. In addition to disjunction (see Section 4.3.2), the following devices have been discussed at one time or another:

- *Negation* allows the grammar to specify that a feature does *not* have a particular value or type of value.
- *Priority union* and the closely related *overwriting* operations allow aggregation of incompatible information, using some suitable criterion for choosing which of two incompatible substructures takes precedence. The former is a recent addition to LFG.
- *Variable labels*, in which a feature name is itself the value of another feature, might be used. LFG employs this capability for certain analyses of free word order.
- *Cyclicity* of feature structures permits a structure to have itself as a substructure. Prolog-II (Colmerauer, 1982) and certain PATR-II implementations, for instance, allow cyclic terms and dags. They have been proposed for use in analyzing relative clauses, and can perform the same tasks as variable labels in the free-word-order analyses.
- *Generalization* is the dual of unification. The generalization of two feature structures is the most specific feature structure subsumed by both. Related devices have been proposed in coordination analyses in GPSG (Sag et al., 1984).

Many other extensions to unification-based formalisms have been proposed. Consequently, certain questions arise regarding how to evaluate the various alternatives. What is important to keep in mind when looking at the possible devices and techniques? This brings us to some concluding remarks about the principal issues in grammar formalisms that are raised by the unification-based approach.

CHAPTER 5

Conclusions

5.1. Emergent Issues Concerning Formalisms

Emerging from these formalisms are several important general issues concerning unification-based formalisms. We touch briefly on some of these issues here.

- *Linguistic motivation:* Chief among the issues engendered by the detailed makeup of grammar formalisms is the linguistic motivation for the various components of the systems. Regardless of whether the formalism is part of a linguistic theory or a linguistic tool, the structures and operations used should be those appropriate for stating linguistic information.
- *Declarativeness:* Certain operations or combinations of operations embedded in a formalism admit of no declarative interpretation. In such a case, the definition of the language admitted by a grammar is inherently procedural. Such cases are often quite subtle to detect, requiring a well-worked-out semantics for the formalism.[1]
- *Nonmonotonicity:* Operations that are used for default behavior of some sort appear in many of the formalisms (e.g., constraint equations, default feature values, overwriting, priority union, *ANY*). A growing body of linguistic evidence seems to show that such devices are needed for certain analyses. Many questions concerning the interchangeability, expressivity, and necessity of these devices remain unresolved.

[1] This use of the word "semantics" should not be confused with the more common usage denoting that portion of a grammar concerned with the meaning of object language sentences. Here we are concerned with the meaning of metalanguage sentences, i.e., of the grammars themselves. Pereira and the author (1984) discuss semantics for grammar formalisms in more detail.

- *Mathematical power:* In addition to the expressive power of the formalism, precise mathematical measures of power taken from complexity theory are pertinent to the evaluation of formalisms. FUG and PATR-II are the most powerful of the formalisms in this highly idealized sense. They can both characterize any recursively enumerable language. Thus, the recognition problem for these formalisms is undecidable. LFG recognition is decidable because of the off-line parsability constraint, but is NP-complete. GPSG, because of its equivalence to context-free systems, is polynomially recognizable, though exponential amounts of preprocessing may be necessary. Various other results are known with respect to the mathematical complexity of the various formalisms. Although such metrics are not very accurate barometers of actual performance characteristics in any sense, they can serve as a first approximation in characterizing expressive power.
- *Algebraic properties:* Understanding the algebraic properties of the various devices is essential to a full understanding of the semantics of a formalism. For instance, the algebraic notion of distributivity is closely related to the issue of declarativeness. Since generalization and unification are not distributive with respect to each other, the order of their application is important, and thus temporality enters the formalism. An understanding of the underlying algebra, therefore, is vital.
- *Lexical organization:* One of the chief issues separating many of the formalisms discussed here has to do with their organization of the lexicon. Since the analyses in these formalisms seem to be heading in a more lexical direction, this topic is of increasing interest.
- *Notation:* Which of the differences among the formalisms are notational rather than formally essential? Recent work in the area has tended to explicate this question more clearly, and many of the differences among the formalisms are now seen as notational. Note that this does not make such differences any less important, but it does provide a better understanding of the source of power in the formalisms: the notation or the semantics.

5.2. A Summary

Research pursuing these various issues in linguistic formalism design is delineating the similarities and differences among the various unification-based formalisms. Linguistic research, meanwhile, is quite concerned about differentiating the various formal constructs with respect to their linguistic basis. Natural-language-processing research is also being conducted, aimed at providing an understanding of the computational ramifications of design issues. Thus, from a variety of areas, the distinctions among the various unification-based formalisms is being tracked.

More surprising, however, is the fundamental observation that from a broad range of research directions—from varied work within linguistics, artificial intelligence, and computer science—researchers are converging upon a single approach to grammar of great flexibility and power, an approach in which declarative and procedural interpretations of grammars can coexist. The foundational issues in unification-based formalisms are only now beginning to be explored, but the efficacy of unification as a tool for linguistic analysis and computation seems irrefutable.

APPENDIX A

The Sample PATR-II Grammars

The following are the machine-readable versions of the grammars presented in Section 3.4.

A.1. Sample Grammar One

```
;;; -*- Mode: PATR -*-

;;;=================================================
;;; Demonstration Grammar One
;;;
;;; Includes: subject-verb agreement
;;;=================================================

Parameter: Start symbol is S.

Parameter: Restrictor is <cat>
                         <head form>.

Parameter: Attribute order is cat lex sense head
                              subject
                              form agreement
                              person number gender
                              s np vp v.

;;;=================================================
;;; Grammar Rules
;;;=================================================

Rule |sentence formation|
```

```
        S --> NP VP:

                <S head> = <VP head>
               <VP head subject> = <NP head>.

Rule |trivial verb phrase|

        VP --> V:

                <VP head> = <V head>.

;;;=====================================================
;;; Lexicon
;;;=====================================================

Lexicon root.

Word uther:

       <cat> = NP
       <head agreement gender> = masculine
       <head agreement person> = third
       <head agreement number> = singular.

Word knights:

       <cat> = NP
       <head agreement gender> = masculine
       <head agreement person> = third
       <head agreement number> = plural.

Word  sleeps:

       <cat> = V
       <head form> = finite
```

```
        <head subject agreement person> = third
        <head subject agreement number> = singular.

Word  sleep:

        <cat> = V
        <head form> = finite
        <head subject agreement number> = plural.
```

A.2. Sample Grammar Two

```
;;; -*- Mode: PATR -*-

;;;=================================================
;;; Demonstration Grammar Two
;;;
;;; Includes: subject-verb agreement
;;; Includes: complex subcategorization
;;;=================================================

Parameter: Start symbol is S.

Parameter: Restrictor is <cat>
                         <head form>.

Parameter: Attribute order is cat lex sense head
                              subcat first rest
                              form agreement
                              person number gender
                              s np vp vp_1 vp_2 vp_3 v.

;;;=================================================
;;; Grammar Rules
;;;=================================================

Rule |sentence formation|

        S --> NP VP:

                <S head> = <VP head>
                <S head form> = finite
                <VP subcat first> = <NP>
                <VP subcat rest> = end.

Rule |trivial verb phrase|
```

```
        VP --> V:

                <VP head> = <V head>
                <VP subcat> = <V subcat>.

Rule |complements|

        VP_1 --> VP_2 X:

                <VP_1 head> = <VP_2 head>
                <VP_2 subcat first> = <X>
                <VP_2 subcat rest> = <VP_1 subcat>.

;;;=================================================
;;; Lexicon
;;;=================================================

Lexicon root.

Word uther:

        <cat> = NP
        <head agreement gender> = masculine
        <head agreement person> = third
        <head agreement number> = singular.

Word cornwall:

        <cat> = NP
        <head agreement gender> = masculine
        <head agreement person> = third
        <head agreement number> = singular.

Word knights:

        <cat> = NP
```

```
        <head agreement gender> = masculine
        <head agreement person> = third
        <head agreement number> = plural.

Word sleeps:

        <cat> = V
        <head form> = finite
        <syncat first cat> = NP
        <syncat first head agreement person> = third
        <syncat first head agreement number> = singular
        <subcat rest> = end.

Word sleep:

        <cat> = V
        <head form> = finite
        <subcat first cat> = NP
        <subcat first head agreement number> = plural
        <subcat rest> = end.

Word sleep:

        <cat> = V
        <head form> = nonfinite
        <subcat first cat> = NP
        <subcat rest> = end.

Word storms:

        <cat> = V
        <head form> = finite
        <subcat first cat> = NP
        <subcat rest first cat> = NP
        <subcat rest first head agreement person> = third
        <subcat rest first head agreement number> = singular
```

```
        <subcat rest rest> = end.

Word stormed:

        <cat> = V
        <head form> = presentparticiple
        <subcat first cat> = NP
        <subcat rest first cat> = NP
        <subcat rest rest> = end.

Word storm:

        <cat> = V
        <head form> = nonfinite
        <subcat first cat> = NP
        <subcat rest first cat> = NP
        <subcat rest rest> = end.

Word has:

        <cat> = V
        <head form> = finite
        <subcat first cat> = VP
        <subcat first head form> = presentparticiple
        <subcat first syncat rest> = end
        <subcat first syncat first> = <subcat rest first>
        <subcat rest first cat> = NP
        <subcat rest first head agreement number> = singular
        <subcat rest first head agreement person> = third
        <subcat rest rest> = end.

Word have:

        <cat> = V
        <head form> = finite
        <subcat first cat> = VP
```

```
        <subcat first head form> = presentparticiple
        <subcat first syncat rest> = end
        <subcat first syncat first> = <subcat rest first>
        <subcat rest first cat> = NP
        <subcat rest first head agreement number> = plural
        <subcat rest rest> = end.

Word persuades:

        <cat> = V
        <head form> = finite
        <subcat first cat> = NP
        <subcat rest first cat> = VP
        <subcat rest first head form> = infinitival
        <subcat rest first syncat rest> = end
        <subcat rest first syncat first> = <subcat first>
        <subcat rest rest first cat> = NP
        <subcat rest rest first head agreement number> =
               singular
        <subcat rest rest first head agreement person> =
               third
        <subcat rest rest rest> = end.

Word to:

        <cat> = V
        <head form> = infinitival
        <subcat first cat> = VP
        <subcat first head form> = nonfinite
        <subcat first syncat rest> = end
        <subcat first syncat first> = <subcat rest first>
        <subcat rest first cat> = NP
        <subcat rest rest> = end.
```

A.3. Sample Grammar Three

```
;;; -*- Mode: PATR -*-

;;;=====================================================
;;; Demonstration Grammar Three
;;;
;;; Includes: subject-verb agreement
;;;           complex subcategorization
;;;           logical form construction
;;;=====================================================

Parameter: Start symbol is S.

Parameter: Restrictor is <cat>
                         <head form>.

Parameter: Translation at <head trans>.


Parameter: Attribute order is cat lex sense head
                              subcat first rest
                              form agreement
                              person number gender
                              trans pred arg1 arg2
                              s np vp vp_1 vp_2 vp_3 v.

;;;=====================================================
;;; Grammar Rules
;;;=====================================================

Rule |sentence formation|

        S --> NP VP:

                <S head> = <VP head>
                <S head form> = finite
```

```
                <VP subcat first> = <NP>
                <VP subcat rest> = end.

Rule |trivial verb phrase|

        VP --> V:

                <VP head> = <V head>
                <VP subcat> = <V subcat>.

Rule |complements|

        VP_1 --> VP_2 X:

                <VP_1 head> = <VP_2 head>
                <VP_2 subcat first> = <X>
                <VP_2 subcat rest> = <VP_1 subcat>.

;;;=====================================================
;;; Lexicon
;;;=====================================================

Lexicon root.

Word uther:

        <cat> = NP
        <head agreement gender> = masculine
        <head agreement person> = third
        <head agreement number> = singular
        <head trans> = uther.

Word cornwall:

        <cat> = NP
        <head agreement gender> = masculine
```

```
        <head agreement person> = third
        <head agreement number> = singular
        <head trans> = cornwall.

Word knights:

        <cat> = NP
        <head agreement gender> = masculine
        <head agreement person> = third
        <head agreement number> = plural
        <head trans> = knights.

Word sleeps:

        <cat> = V
        <head form> = finite
        <subcat first cat> = NP
        <subcat first head agreement person> = third
        <subcat first head agreement number> = singular
        <subcat rest> = end
        <head trans pred> = sleep
        <head trans arg1> = <subcat first head trans>.

Word sleep:

        <cat> = V
        <head form> = finite
        <subcat first cat> = NP
        <subcat first head agreement number> = plural
        <subcat rest> = end
        <head trans pred> = sleep
        <head trans arg1> = <subcat first head trans>.

Word sleep:

        <cat> = V
```

```
        <head form> = nonfinite
        <subcat first cat> = NP
        <subcat rest> = end
        <head trans pred> = sleep
        <head trans arg1> = <subcat first head trans>.

Word storms:

        <cat> = V
        <head form> = finite
        <subcat first cat> = NP
        <subcat rest first cat> = NP
        <subcat rest first head agreement person> = third
        <subcat rest first head agreement number> = singular
        <subcat rest rest> = end
        <head trans pred> = storm
        <head trans arg1> = <subcat rest first head trans>
        <head trans arg2> = <subcat first head trans>.

Word stormed:

        <cat> = V
        <head form> = presentparticiple
        <subcat first cat> = NP
        <subcat rest first cat> = NP
        <subcat rest rest> = end
        <head trans pred> = storm
        <head trans arg1> = <subcat rest first head trans>
        <head trans arg2> = <subcat first head trans>.

Word storm:

        <cat> = V
        <head form> = nonfinite
        <subcat first cat> = NP
        <subcat rest first cat> = NP
```

```
        <subcat rest rest> = end
        <head trans pred> = storm
        <head trans arg1> = <subcat rest first head trans>
        <head trans arg2> = <subcat first head trans>.

Word has:

        <cat> = V
        <head form> = finite
        <subcat first cat> = VP
        <subcat first head form> = presentparticiple
        <subcat first syncat rest> = end
        <subcat first syncat first> = <subcat rest first>
        <subcat rest first cat> = NP
        <subcat rest first head agreement number> = singular
        <subcat rest first head agreement person> = third
        <subcat rest rest> = end
        <head trans pred> = perfective
        <head trans arg1> = <syncat first head trans>.

Word have:

        <cat> = V
        <head form> = finite
        <subcat first cat> = VP
        <subcat first head form> = presentparticiple
        <subcat first syncat rest> = end
        <subcat first syncat first> = <subcat rest first>
        <subcat rest first cat> = NP
        <subcat rest first head agreement number> = plural
        <subcat rest rest> = end
        <head trans pred> = perfective
        <head trans arg1> = <subcat first head trans>.

Word persuades:
```

```
        <cat> = V
        <head form> = finite
        <subcat first cat> = NP
        <subcat rest first cat> = VP
        <subcat rest first head form> = infinitival
        <subcat rest first subcat rest> = end
        <subcat rest first subcat first> = <subcat first>
        <subcat rest rest first cat> = NP
        <subcat rest rest first head agreement number> =
                singular
        <subcat rest rest first head agreement person> =
                third
        <subcat rest rest rest> = end
        <head trans pred> = persuade
        <head trans arg1> =
                <subcat rest rest first head trans>
        <head trans arg2> = <subcat first head trans>
        <head trans arg3> = <subcat rest first head trans>.

Word to:

        <cat> = V
        <head form> = infinitival
        <subcat first cat> = VP
        <subcat first head form> = nonfinite
        <subcat first subcat rest> = end
        <subcat first subcat first> = <subcat rest first>
        <subcat rest first cat> = NP
        <subcat rest rest> = end
        <head trans> = <subcat first head trans>.
```

A.4. Sample Grammar Four

```
;;; -*- Mode: PATR -*-

;;;=====================================================
;;; Demonstration Grammar Four
;;;
;;; Includes: subject-verb agreement
;;;           complex subcategorization
;;;           logical form construction
;;;           lexical organization by templates
;;;           and lexical rules
;;;=====================================================

Parameter: Start symbol is S.

Parameter: Restrictor is <cat>
                         <head form>.

Parameter: Translation at <head trans>.

Parameter: Attribute order is cat lex sense head
                              subcat first rest
                              form agreement
                              person number gender
                              trans pred arg1 arg2
                              s np vp vp_1 vp_2 vp_3 v.

;;;=====================================================
;;; Grammar Rules
;;;=====================================================

Rule |sentence formation|

        S --> NP VP:

                <S head> = <VP head>
```

```
                <S head form> = finite
                <VP subcat first> = <NP>
                <VP subcat rest> = end.

Rule |trivial verb phrase|

        VP --> V:

                <VP head> = <V head>
                <VP subcat> = <V subcat>.

Rule |complements|

        VP_1 --> VP_2 X:

                <VP_1 head> = <VP_2 head>
                <VP_2 subcat first> = <VP_1 subcat first>
                <VP_2 subcat rest first> = <X>
                <VP_2 subcat rest rest> = <VP_1 subcat rest>.

;;;=====================================================
;;; Definitions
;;;=====================================================

Let Verb be <cat> = v.

Let Finite be Verb
              <head form> = finite.

Let Nonfinite be Verb
                 <head form> = nonfinite.

Let ThirdPerson be
        <subcat first head agreement person> = third.

Let Singular be
```

```
        <subcat first head agreement number> = singular.

Let Plural be
        <subcat first head agreement number> = plural.

Let ThirdSing be Finite
                 ThirdPerson
                 Singular.

Let MainVerb be Verb
                <head aux> = false.

Let Transitive be <subcat first cat> = NP
                  <subcat rest first cat> = NP
                  <subcat rest rest> = end
                  <head trans arg1> =
                        <subcat first head trans>
                  <head trans arg2> =
                        <subcat rest first head trans>.

Let Intransitive be MainVerb
                    <subcat first cat> = NP
                    <subcat rest> = end
                    <head trans arg1> =
                          <subcat first head trans>.

Let Raising be <subcat first cat> = NP
               <subcat rest first cat> = VP
               <subcat rest first subcat rest> = end
               <subcat rest first subcat first> =
                       <subcat first>
               <subcat rest rest> = end.

Define AgentlessPassive as <out cat> = <in cat>
                           <out subcat> = <in subcat rest>
                           <out head> = <in head>
```

```
                        <out head form> =>
                              passiveparticiple.

;;;=================================================
;;; Lexicon
;;;=================================================

Lexicon root.

Word uther:

        <cat> = NP
        <head agreement gender> = masculine
        <head agreement person> = third
        <head agreement number> = singular
        <head trans> = uther.

Word cornwall:

        <cat> = NP
        <head agreement gender> = masculine
        <head agreement person> = third
        <head agreement number> = singular
        <head trans> = cornwall.

Word knights:

        <cat> = NP
        <head agreement gender> = masculine
        <head agreement person> = third
        <head agreement number> = plural
        <head trans> = knights.

Word sleeps: Intransitive ThirdSing
             <head trans pred> = sleep.
```

```
Word sleep: Intransitive Plural
            <head trans pred> = sleep.

Word sleep: Intransitive Nonfinite
            <head trans pred> = sleep.

Word storms: Transitive ThirdSing
             <head trans pred> = storm.

Word stormed: Transitive AgentlessPassive
              <head trans pred> = storm.

Word storm: Transitive Nonfinite
            <head trans pred> = storm.

Word is: Raising ThirdSing
         <subcat rest first head form> = passiveparticiple
         <head trans> = <subcat rest first head trans>.
```

APPENDIX B

The Literature

This appendix attempts to point out certain key works in the various areas that touch upon the topics of this paper.

B.1. General Papers

The general considerations of Chapter 0 are described more fully by the author (Shieber, 1985a); the cited paper forms the basis of that chapter. The semantics of unification-based grammar formalisms in general is discussed by Pereira and the author (Pereira and Shieber, 1984).

B.2. Background and Overviews of the Formalisms

LFG has been canonized in Bresnan's book *The Mental Representation of Grammatical Relations* (1982). Details of the formalism itself are presented in Chapter 4 of that book. FUG is presented in several works by Kay, most accessibly, "Unification Grammar" (1983). Woods (1970) describes ATNs, from which LFG was partially developed.

Colmerauer describes his Q-systems (1970) and the later metamorphosis grammars (1978). Prolog and DCGs are discussed in an introductory text by Clocksin and Mellish (1981). Colmerauer and Roussel are the respective authors of two classic works (Colmerauer et al., 1973; Roussel, 1975). More extensive discussions of DCG and extraposition grammars are given by Pereira in his thesis (1983). Other DCG-related formalisms are described by McCord and Dahl (McCord, 1980; Dahl and Abramson, 1984; Dahl and McCord, 1983) and in works cited therein.

Early GPSG was described in the difficult to obtain "English as a Context-free Language" (Gazdar, 1979) and in several subsequent papers, now considered out of date by the authors. It is best introduced in a series of papers by Gazdar (Gazdar, 1982; Gazdar et al., 1982; Gazdar, 1981). The canonical work on the current formalism is the recently published *Generalized Phrase*

Structure Grammar (Gazdar et al., 1985). A simpler introduction to the current formalism is given by Sag et al. (1984). Pollard's head grammars are first described in his Ph.D. thesis (1984). The later HPSG formalism is documented in an unpublished note (Pollard, 1985a) and a short paper (Pollard, 1985b).

DIALOGIC, the precursor of PATR, is described by Robinson (1982). Rosenschein and the author present the original PATR formalism (Rosenschein and Shieber, 1982). PATR-II is introduced in an SRI report (Shieber, 1984) and further discussed in a recent compilation of papers (Shieber et al., 1984).

B.3. Handling Specific Linguistic Phenomena

The LFG volume (Bresnan, 1982) presents detailed analyses of a wide range of phenomena from radically differing languages. Chapters 1 and 5 provide an introduction to the style of analysis upon which LFG is based. The monumental bibliography of works dealing with relational grammar (Dubinsky and Rosen, 1983) contains many references to papers in the LFG literature and more using the shared tenet with relational grammar of grammatical functions as primitives.

Since the GPSG volume (Gazdar et al., 1985) concentrates primarily on English linguistics, more detailed analysis was therefore possible. Chapter 2 presents the informational domain of categories, Chapter 3 the combinatory rules. Chapter 5 is devoted to the general principles upon which the theory is based. Other chapters provide detailed analyses of English phenomena (especially long-distance dependencies and coordination) and semantic interpretation. For GPSG analyses of other languages, see, for instance, the compilation *Order, Concord and Constituency* (Gazdar et al., 1983) and references in footnote 6 of the introduction to the GPSG volume (Gazdar et al., 1985).

The analysis of subcategorization, agreement, and semantics given in Section 3.4, such as it is, has been previously presented by the author (Shieber et al., 1983).

B.4. Related Formalisms and Languages from Computer Science

Besides the obvious connections to logic and theorem-proving research (unification was originally discussed as a component of the resolution procedure for automatic theorem-proving (rob), and this connection is still evident

in Prolog and DCG) other research from computer science bears directly on the topic at hand.

There is a close relationship between the type theory of computer science and the algebraic structure of feature systems. Roughly speaking, the similarity is between feature structures and named product types (or numbered product types for DCG terms) with or without sum types. Reynolds (1985) presents a good introduction to the approriate type theory, including a lucid explanation of the differences between named and numbered products.

Building on this relationship with type theory, Ait-Kaci (1985) discusses a calculus of syntactic types that bears a remarkable resemblance to the feature structures used in unification-based formalisms. The mathematics of subsumption, unification, and other algebraic properties of his calculus are investigated in depth. The intended application of the formalism was to knowledge representation—it was originally described as a generalization of Prolog terms—but some brief natural-language examples are given by Ait-Kaci.

Cardelli (1984), in a reconstruction of object-oriented programming, proposes a typing system based on named product types with sums for disjunction. This type system also bears a close resemblance to the feature structure domains. He proposes this as a technique for modeling object classes with multiple inheritance in a strongly typed language.

B.5. Related Implementation Techniques

Implementation of systems that use unification-based grammar formalisms is aided by the vast literature available from the automatic theorem-proving and Prolog communities on implementing unification, and from the programming language and compiler design communities on implementing parsing algorithms.

The following research is seminal in this field. rob first discusses the use of unification in automatic theorem-proving. Later work by Boyer and Moore (1972) used structure sharing to improve the efficiency of unification. Nelson and Oppen (1978) describe an algorithm for efficiently computing graph closures, with an application to solving equations of the type found in unification-based formalisms. Their algorithm forms the basis for an implementation of unification for LFG. Structure-sharing methods for implementing feature structures are discussed by Karttunen and Kay (1985) and Pereira

(1985). Efficient implementation of Prolog makes use of these techniques and others developed by Warren (1983).

Among the parsing techniques that have been modified for use in unification-based parsing are many of the techniques designed for parsing programming languages. The two-volume reference by Aho and Ullman (1972) provides a complete introduction to the available techniques. Kaplan (1973) discusses chart parsing for natural language analysis. A technique specifically for parsing unification-based formalisms has been described by the author (Shieber, 1985b). Kay (1985) discusses parsing FUG.

B.6. Implementations of Unification-Based Formalisms

Several systems implementing unification-based grammar formalisms have been devised. The following references are to papers describing the systems themselves. A user manual for the LFG system developed at the Xerox Palo Alto Research Center will be forthcoming from Xerox. GPSG systems have been developed by Thompson (1981), and Evans and Gazdar (1984). Work on the GPSG and subsequent HPSG systems at Hewlett-Packard is reviewed in several papers (Gawron et al., 1982; Proudian and Pollard, 1985; Pollard and Creary, 1985). The various PATR-II systems are described in SRI reports (Shieber et al., 1983; Shieber, 1985a). Other references to papers on computer implementations of phrase-structure grammars of various sorts can be found in Gazdar's short bibliography (Gazdar, 1984).

Bibliography

.

Ades, Anthony E., and Mark J. Steedman. 1982. On the order of words. *Linguistics and Philosophy* 4(4):517–558.

Aho, Al V., and Jeffrey D. Ullman. 1972. *Theory of parsing, translation and compiling*, vol. 1. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Ait-Kaci, H. 1985. A new model of computation based on a calculus of type subsumption. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA.

Bach, Emmon W. 1980. In defense of passive. *Linguistics and Philosophy* 3(3):297–341.

Boyer, R. S., and J S. Moore. 1972. The sharing of structure in theorem-proving programs. In *Machine intelligence 7*, 101–116. New York, New York: John Wiley and Sons.

Bresnan, Joan, ed. 1982. *The mental representation of grammatical relations*. Cambridge, Massachusetts: MIT Press.

Cardelli, L. 1984. A semantics of multiple inheritance. Tech. Rep., Bell Laboratories, Murry Hill, NJ.

Carpenter, Bob. 1992. *The logicof typed feature structures: With applications to unification grammars, logic programs, and constraint resolution*, vol. 32 of *Cambridge Tracts in Theoretical Computer Science*. New York, NY: Cambridge University Press.

Clocksin, W. F., and C. S. Mellish. 1981. *Programming in Prolog*. Springer-Verlag, Berlin.

Colmerauer, Alain. 1970. Les systèmes-q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur. Internal Publication 43, Département d'Informatique, Université de Montreal, Canada.

———. 1978. Metamorphosis grammars. In *Natural language communication with computers*, ed. L. Bolc. Springer-Verlag, Berlin. First appeared as 'Les Grammaires de Metamorphose', Groupe d'Intelligence Artificielle,

Université de Marseille II, November 1975.

———. 1982. Prolog II reference manual and theoretical model. Tech. Rep., Groupe Intelligence Artificielle—ERA CNRS 363.

Colmerauer, Alain, H. Kanoui, R. Pasero, and Phillipe Roussel. 1973. Un systeme de comunication homme-machine en francais. Rapport, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille II.

Dahl, Veronica, and Harvey Abramson. 1984. On gapping grammars. In *Proceedings of the second international logic programming conference*, ed. Sten-Åke Tärnlund, 77–88. Uppsala, Sweden: Ord and Form.

Dahl, Veronica, and Michael C. McCord. 1983. Treating coordination in logic grammars. *American Journal of Computational Linguistics* 9(2):69–91.

Dubinsky, Stanley, and Carol Rosen. 1983. *A bibliography on relational grammar through April 1983 with selected titles from lexical functional grammar*. Bloomington, Indiana: Indiana University Linguistics Club.

Evans, Roger, and Gerald Gazdar. 1984. The ProGram manual. Cognitive Science Research Paper 35, University of Sussex, Sussex, England.

Flickinger, Dan, Carl Pollard, and Thomas Wasow. 1985. Structure-sharing in lexical representation. In *Proceedings of the 23rd annual meeting of the Association for Computational Linguistics*, 262–267. Chicago, Illinois: University of Chicago.

Gawron, Jean Mark, Jonathon King, John Lamping, Egon Loebner, Elizabeth Anne Paulson, Geoffrey K. Pullum, Ivan A. Sag, and Thomas Wasow. 1982. Processing English with a generalized phrase structure grammar. In *Proceedings of the 20th annual meeting of the Association for Computational Linguistics*, 74–81. Toronto, Ontario, Canada: University of Toronto.

Gazdar, Gerald. 1979. English as a context-free language. Cognitive Studies Programme, School of Social Sciences, University of Sussex.

———. 1981. Unbounded dependencies and coordinate structure. *Linguistic Inquiry* 12(2):155–184.

———. 1982. *Phrase structure grammar*, 131–186. Dordrecht, Holland: D. Reidel.

———. 1984. Recent computer implementations of phrase structure grammar. *Computational Linguistics* 10(3-4):212–214.

Gazdar, Gerald, Ewan Klein, and Geoffrey K. Pullum, eds. 1983. *Order, concord and constituency*. Dordrecht, Holland: Foris Publications.

Gazdar, Gerald, Ewan Klein, Geoffrey K. Pullum, and Ivan A. Sag. 1985. *Generalized phrase structure grammar*. Blackwell Publishing, Oxford, England, and Harvard University Press, Cambridge, Massachusetts.

Gazdar, Gerald, Geoffrey K. Pullum, and Ivan A. Sag. 1982. Auxiliaries and related phenomena in a restrictive theory of grammar. *Language* 58(3):591–638.

Johnson, Mark. 1988. *Attribute-value logic and the theory of grammar*, vol. 16 of *CSLI Lecture Notes*. Stanford, CA: Center for the Study of Language and Information.

Kaplan, Ron, and Joan Bresnan. 1983. Lexical-functional grammar: A formal system for grammatical representation. In *The mental representation of grammatical relations*, ed. Joan Bresnan. Cambridge, Massachusetts: MIT Press.

Kaplan, Ron M. 1973. A general syntactic processor. In *Natural language processing*, ed. R. Rustin. New York, New York: Algorithmics Press.

Karttunen, Lauri, and Martin Kay. 1985. Structure sharing with binary trees. In *Proceedings of the 23rd annual meeting of the Association for Computational Linguistics*. Chicago, Illinois: University of Chicago.

Kay, Martin. 1983. Unification grammar. Tech. Rep., Xerox Palo Alto Research Center, Palo Alto, California.

———. 1985. Parsing in functional unification grammar. In *Natural language parsing: Psychological, computational and theoretical perspectives*, chap. 7, 251–278. Studies in Natural Language Processing, Cambridge, England: Cambridge University Press.

Matsumoto, Yuji, Hozumi Tanaka, Hideki Hirakawa, Hideo Miyoshi, and Hideki Yasukawa. 1983. BUP: A bottom-up parser embedded in Prolog. *New Generation Computing* 1:145–158.

McCord, M. C. 1980. Slot grammars. *American Journal of Computational Linguistics* 6(1):255–286.

Montague, Richard. 1974. The proper treatment of quantification in ordinary English. In *Formal philosophy*, ed. Richmond H. Thomason, 188–221. New Haven, Connecticut: Yale University Press.

Nelson, Greg, and Derek C. Oppen. 1978. Fast decision algorithms based on congruence closure. Tech. Rep. AIM-309, Stanford Artificial Intelligence Laboratory, Stanford University, Stanford, California. Also in *Proceedings*

of the 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, October, 1977.

Pereira, Fernando C. N. 1983. Logic for natural language analysis. Technical Note 275, Artificial Intelligence Center, SRI International, Menlo Park, California.

———. 1985. A structure-sharing representation for unification-based grammar formalisms. In *Proceedings of the 23rd annual meeting of the Association for Computational Linguistics*. Chicago, Illinois: University of Chicago.

Pereira, Fernando C. N., and Stuart M. Shieber. 1984. The semantics of grammar formalisms seen as computer languages. In *Proceedings of the tenth international conference on computational linguistics*. Stanford University, Stanford, California.

Pereira, Fernando C. N., and David H. D. Warren. 1980. Definite clause grammars for language analysis—a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence* 13:231–278.

Pollard, Carl. 1984. Generalized phrase structure grammars, head grammars, and natural languages. Ph.D. thesis, Stanford University, Stanford, California.

———. 1985a. Lecture notes on head-driven phrase-structure grammar. Center for the Study of Language and Information, unpublished.

———. 1985b. Phrase structure grammar without metarules. In *Proceedings of the fourth west coast conference on formal linguistics*. Los Angeles, California: University of Southern California.

Pollard, Carl, and Lewis Creary. 1985. A computational semantics for natural language. In *Proceedings of the 23rd annual meeting of the Association for Computational Linguistics*. Chicago, Illinois: University of Chicago.

Proudian, Derek, and Carl Pollard. 1985. Parsing head-driven phrase structure grammar. In *Proceedings of the 23rd annual meeting of the Association for Computational Linguistics*. Chicago, Illinois: University of Chicago.

Reynolds, John C. 1985. Three approaches to type structure. To appear in the Springer-Verlag Lecture Notes in Computer Science.

Robinson, Jane J. 1982. DIAGRAM: A grammar for dialogues. *Communications of the ACM* 25(1):27–47.

Rosenschein, Stanley J., and Stuart M. Shieber. 1982. Translating English into logical form. In *Proceedings of the 20th annual meeting of the Association*

for Computational Linguistics, 1–8. Toronto, Ontario, Canada: University of Toronto.

Roussel, Phillipe. 1975. Prolog: Manuel de référence et utilisation. Tech. Rep., Groupe d'Intelligence Artificielle, Université d'Aix-Marseille II, Marseille, France.

Sag, Ivan A., Gerald Gazdar, Thomas Wasow, and Steven Weisler. 1984. Coordination and how to distinguish categories. Report CSLI-86-3, Center for the Study of Language and Information, Stanford, California. Also to appear in *Linguistics and Philosophy*.

Sag, Ivan A., and Thomas Wasow. 1999. *Syntactic theory: A formal introduction*, vol. 92 of *CSLI Lecture Notes*. Center for the Study of Language and Information, Stanford, CA: CSLI Publications.

Shieber, Stuart M. 1984. The design of a computer language for linguistic information. In *Proceedings of the tenth international conference on computational linguistics*. Stanford University, Stanford, California.

———. 1985a. Criteria for designing computer facilities for linguistic analysis. *Linguistics* 23:189–211.

———. 1985b. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *Proceedings of the 22nd annual meeting of the Association for Computational Linguistics*. University of Chicago, Chicago, Illinois.

Shieber, Stuart M., Lauri Karttunen, and Fernando C. N. Pereira. 1984. Notes from the unification underground: A compilation of papers on unification-based grammar formalisms. Tech. Rep. 327, Artificial Intelligence Center, SRI International, Menlo Park, California.

Shieber, Stuart M., Hans Uszkoreit, Fernando C. N. Pereira, Jane J. Robinson, and Mabry Tyson. 1983. The formalism and implementation of PATR-II. In *Research on interactive acquisition and use of knowledge*. Menlo Park, California: Artificial Intelligence Center, SRI International. SRI Final Report 1894.

Thompson, H. 1981. Chart parsing and rule schemata in GPSG. In *Proceedings of the 19th annual meeting of the association for computational linguistics*, 167–172. Stanford University, Stanford, California: Association for Computational Linguistics.

Warren, David H. D. 1983. Applied logic—its use and implementation as a programming tool. Tech. Rep. 290, Artificial Intelligence Center, SRI

International, Menlo Park, California.

Woods, William. 1970. Transition network grammars for natural language analysis. *Communications of the ACM* 13(10).