Criteria for Designing Computer Facilities for Linguistic Analysis

SRI International

CRITERIA FOR DESIGNING COMPUTER FACILITIES FOR LINGUISTIC ANALYSIS

Technical Note 354

April 1985

By: Stuart Shieber
Computer Scientist
Artificial Intelligence Center
SRI International
and
Center for the Study of Language and Information
Stanford University

This paper to appear in *Linguistics*.

This research was made possible,in part by a gift from the Systems Development Foundation; additional support was provided by the Defense Advanced Research Projects Agency under Contract N00039-84-K-0078 with the Naval Electronics Systems Command. The views and conclusions contained in this document should not be interpreted as representative of the official policies, either expressed or implied, of the Defense Research Projects Agency or the United States government.

SRI International®

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

Criteria for Designing Computer Facilities for Linguistic Analysis

Stuart M. Shieber
Artificial Intelligence Center
SRI International
and
Center for the Study of Language and Information
Stanford University

Abstract

In the natural-language-processing research community, the usefulness of computer tools for testing linguistic analyses is often taken for granted. Linguists, on the other hand, have generally been unaware of or ambivalent about such devices. We discuss several aspects of computer use that are preeminent in establishing the utility for linguistic research of computer tools and describe several factors that must be considered in designing such computer tools to aid in testing linguistic analyses of grammatical phenomena. A series of design alternatives, some theoretically and some practically motivated, is then based on the resultant criteria. We present one way of pinning down these choices which culminates in a description of a particular grammar formalism for use in computer linguistic tools. The PATR-II formalism thus serves to exemplify our general perspective.

This research was made possible, in part, by a gift from the Systems Development Foundation; additional support was provided by the Defense Advanced Research Projects Agency under Contract N00039-84-K-0078 with the Naval Electronics Systems Command. The views and conclusions contained in this document should not be interpreted as representative of the official policies, either expressed or implied, of the Defense Research Projects Agency or the United States government.

The author is indebted to William Croft, Gerald Gazdar, Ewan Klein, Fernando Pereira, and Hans Uszkoreit for their comments on earlier drafts of this paper.

1 Introduction

This paper discusses factors that must be considered in designing computer tools to aid in testing linguistic analyses of grammatical phenomena. A series of design alternatives, some theoretically and some practically motivated, is then based on the resultant criteria. We present one way of pinning down these choices which culminates in a description of a particular grammar formalism for use in computer linguistic tools. The PATR-II formalism thus serves to exemplify our general perspective. But before discussing a device of this sort, some justification may be required. Why do we need computer tools for linguistic research at all?[1]

2 Why Computer Tools for Linguistics?

In the natural-language-processing research community, the usefulness of computer tools for testing linguistic analyses is often taken for granted. Linguists, on the other hand, have generally been unaware of or ambivalent about such devices. Three aspects of computer use are preeminent in establishing the utility of such tools: the computer as straitjacket, as touchstone, and as mirror.

2.1 The Computer as Straitjacket

The Chomskyan revolution of formal syntax provided linguists with their first formal tools for the precise description of syntactic phenomena. The generative framework opened up a veritable Pandora's box of options for such formal analyses, linguists being quite clever at designing formal (and quasi-formal) manipulations to describe phenomena. Now, more than ever, the literature in syntax and semantics is producing a plethora of such devices.

Unfortunately, this freedom may soon seduce one into building analyses by using such a variety of techniques or devices, or by using them in such diverse ways, that the overall description is no longer consistent. Decisions in one part of the grammar, while internally consistent, may not cohere

[1] In the discussion to follow, we will concentrate primarily on tools for modeling syntactic and semantic phenomena, although we believe that computer tools are equally useful for a much broader range of linguistic phenomena.

with interacting decisions in another part. In such cases, any claims made about either part of the grammar evaporate, since the two cannot be put together to make a coherent whole. The problem becomes especially acute as linguists attempt to encompass more and more of the phenomena of a language with their formal techniques. In this light, it is interesting to note that one rarely finds a linguistics monograph or dissertation that provides a full listing (e.g., as an appendix) of the final versions of the rules which are postulated in the course of the discussion—perhaps because these rules are not (and were not even necessarily intended to be) consistent.

The computer can play a role in forcing rigorous consistency. A program that interprets grammatical rules to yield some pseudolinguistic behavior—such as parsing or generating sentences, or performing any of the numerous other tasks artificial intelligence researchers have assigned to natural-language-processing programs—allows no room for inconsistent analyses of phenomena. Furthermore, if a portion of the task is not to be included in the formal analysis, the machine's behavior makes that fact painfully apparent. As Geoffrey K. Pullum has been known to say, with computers "there is no rug." The idealizations one makes are forced to be explicit. In this way, the envelope of one's theory is clearly delineated. Hand-waving is impossible when one's arms are in a straitjacket.

2.2 The Computer as Touchstone

The computer serves another role by manifesting behavior under the guidance of a particular formal analysis. Its behavior is an undeniable semaphore indicating the correctness and completeness of an analysis. The linguist argues for particular rules or laws of grammar by showing that they account for the distribution of judgments of grammaticality, synonymy, ambiguity, entailment, etc. The computer, in modeling the judging process, serves as an impartial adjudicator of these claims.

A popular misconception among some linguists is that, while computers may perform the minor, ancillary function of finding typographical or otherwise inconsequential errors in an analysis, they serve no purpose in the real heart of linguistics, because they are incapable of uncovering nontrivial and unanticipated conceptual problems. The experience among artificial intelligence researchers engaged in natural-language-processing work certainly contradicts this view. Robinson has noted that

> "A problem is always incurred when extending the rules to cover more expressions, whether by writing new rules explicitly or by deriving them from old rules.... Introducing new rules almost inevitably has a perturbing effect as they interact with the old rules *in unforeseen ways*. [Emphasis added.] These perturbations are worth studying for the light they shed on the English language, or more precisely, on a grammarian's intuitions about the English language." [22]

With an increasing number of linguists outside the artificial-intelligence community using computers to help build and test their theories, we find evidence from that sector, as well. For instance, Hewlett-Packard has undertaken an effort to implement a natural-language system based on a generalized phrase-structure analysis of English under the guidance of several of the linguist founders of the formalism. They note that

> "In some cases we actually changed our minds about what the correct analysis was when we saw the machine draw out the full range of consequences of a given proposal. Some consequences of an entire grammar cannot be seen by the unaided human brain, just as some visual details cannot be seen by the unaided human eye." [21]

In fact, we have found that among those who have actually attempted to write a computer-interpretable grammar, the experience has been invaluable in revealing real errors that had not been anticipated by the Gedanken-processing typically used by linguists to evaluate their grammars—errors usually due to unforeseen interactions of various rules or principles.

A side effect of the computer's ability to verify analyses is that it can be quite effective in helping a linguist find deficiencies. Grammar "debugging" is at best a long and difficult process. The large grammars compiled by Sager [23] and Robinson [22] have been under constant development since 1963 and 1974 respectively, much of that time being devoted to debugging the grammar. Computer tools can expedite this process by presenting useful information about the grammar and the way it relates to specific pieces of language.

The computer thus serves as a touchstone for verifying the correctness of a grammatical analysis. Unlike the actual touchstone used for determining the purity of precious metals, however, this test also has the alchemic potential of converting the spurious to the genuine.

2.3 The Computer as Mirror

We have already alluded to the manifold possibilities of formal analyses for grammatical phenomena. But if, as we have said, the computer is a straitjacket, an unforgiving touchstone of correctness, why should we want to use it; would it not actually keep us from exploring these alternatives?

In fact, it does not. Although the computer requires a precise and internally consistent analysis, it imposes few a priori limits as to which analysis it uses. This is the paradox of the computer as a modeling tool.[2]

Of course, the actual computer tools that have been implemented do involve such a priori constraints. Some do so because of ideology: the program is intended to manifest the same constraints that humans are claimed to be subject to by virtue of universal grammar or performance limitations. Other computer tools do so for more pragmatic reasons: without the constraints, the implementation would be far more difficult or, given the state of the art, even impossible. Nonetheless, the computer does provide a degree of flexibility that allows it to assist in arbitration among diverse linguistic analyses and theories, since it can be used to reflect such analyses objectively and independently.[3]

Indeed, this raises another methodological question. If the goal of linguistics is to form *constrained* theories of grammar, should computer tools permit this latitude of freedom? Prima facie the answer should be "no," but it is important not to confuse linguistic tools with linguistic theories. A powerful, flexible computer tool can be used to test many (perhaps highly constrained) theories of grammar. In working towards constrained theories

[2]In fact, all that seems to be required is *effectiveness* or *recursivity* (in the technical sense of complexity theory). Though some linguists—especially Langendoen and Postal [14]—deny the effective character of natural languages, we will not discuss this issue here.

[3]We do not mean to imply that choices among analyses made on the basis of computer implementation are inherently objective, only that subjectivity is necessarily limited to the evaluation of the analyses on the basis of accurate, objective information.

of grammar, nothing should prevent us from using as powerful a tool as possible to test these theories; the tool is not itself the theory.

The computer serves as a mirror, objectively reflecting everything within its purview. It is thus a linguistic agnostic, bound to no particular analysis or theory, yet requiring precision and consistency of all analyses and theories. Paradoxically both constraining and anarchistic, it constitutes an ideal instrument with which to compare, and even unify, disparate theories of grammar and analyses of linguistic phenomena.

3 Considerations in Designing Computer Tools for Linguistic Analysis

3.1 General Considerations

Broadly delineated, computer facilities for testing linguistic analyses operate by interpreting symbolic encodings of the analyses, i.e., grammars, in some way that yields useful information. Various modes of interpretation have been utilized; among the most useful is the analysis of sentences with respect to the grammar, thereby yielding the grammar's implicit judgment of sentential grammaticality, ambiguity, semantic content, etc. Its usefulness derives from the fact that it yields much the same information that linguists employ to build the analyses in the first place.[4]

The choice of the language in which the analyses are encoded, the *grammar formalism*, is critical, since it determines the following three parameters which serve as important evaluative criteria:

[4]In addition, interpretation by generating sentences has been widely used. Less common is interpretation by symbolic manipulation of grammars, e.g., a program that could determine whether certain properties of a grammar (say, context-freeness, off-line parsability [18]) provably obtained. Such a program might be used to determine if some postulated axiom of one theory might be an emergent property of grammars in another. This approach merits much more attention than it has previously received.

Note that, in our view, the fact that a framework is "generative" does not preclude analysis as a mode of grammar interpretation, nor does it indicate the primacy of generation as interpretive mode. It merely indicates a particular style of delineating the language described by a grammar—namely, the language that is generated by a standard generating function operating on the grammar.

Linguistic felicity: The degree to which descriptions of linguistic phenomena can be directly (or indirectly) stated as linguists tend 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 such devices do exist, what computational limitations inhere in them.

The trade-offs among these criteria preclude them from coexisting optimally within any single language. For instance, as the power of the formalism grows, sufficiently efficient algorithms for parsing may no longer exist. Alternatively, as a formalism becomes oriented toward the style of analysis of one particular linguistic theory, the class of expressible analyses may diminish.[5]

Nevertheless, these criteria can serve to divert us from certain prospective grammatical formalisms. For instance, a general-purpose programming language meets the second criterion; it is certainly a powerful tool for testing analyses, since it can be used to write parsers for an object language—often efficient ones. However, programming languages fail miserably as linguistic tools because they encode analyses at the wrong level linguistically—that is, they fail the first criterion. Linguists typically state grammars *declaratively*, as rules, filters, and constraints—that is, they describe *what* the strings of the language are like; with few exceptions, programming languages are too *procedural* to be used in this manner—they describe *how* to compute certain properties of a string.[6]

3.2 Some Particular Design Choices

Let us consider how these admittedly programmatic criteria can be applied to the actual design decision process. As we have noted, these criteria

[5] Recall that, although this is the *point* of a constrained linguistic theory, it is a *detriment* for a linguistic tool.

[6] To a lesser extent, ATNs [27] suffer from the same problem of procedurality. It should be noted, however, that a programming language with an independent declarative interpretation, Prolog, has been found quite useful for natural-language processing in the direct implemention of definite-clause grammars. Pereira and Warren [17] discuss these issues more thoroughly.

do not force a particular choice of formalism. Consequently, the decisions discussed here will not be the only ones possible, but are merely examples demonstrating how this perspective on linguistic tools might lead to the choice of a formalism.

Our interest is in building a computational tool to test analyses by performing automatic analysis of sentences relative to a grammar written in the selected grammar formalism. Inherent in this mode of interpretation is the requirement that the analyses we encode be *surface-based*—that is, they should at some point describe the actual surface order of string elements, associating with the strings information about the particular sentential analysis.[7] In summary, the interpretation of the grammar yields a pairing between strings in the language and elements from some informational domain. Given this quite broad requirement derivable from our notion of linguistic tool, we consider each of the foregoing three criteria individually.

3.2.1 Linguistic Felicity

In ensuring that the formalism will allow statements to be made in the way linguists tend to make them, the felicity criterion supports two further design decisions in the formalism. First, linguistic analyses are inductive; in other words, the pairings are defined recursively, new pairings being derived by merging substrings according to string-combining operations (concatenation, wrapping, substitution, etc.) and merging the associated informational elements by information-combining operations (logical operations, unification, even phrase-marker building, etc.). Second, the informational elements tend to be broadly characterizable as associations between *features* (also called *attributes, labels,* etc.) and *values* taken from some well-defined, possibly structured set.[8]

As we will discuss more fully in Section 4.1, we can take this domain of informational elements 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

[7] This requirement makes problematic the use of government-and-binding-style analyses [6], until such time as the rules in the phonological-form component have developed sufficiently to explicate the connection of GB grammars to surface order.

[8] By "characterizable" we mean that the linguistic formalisms either use such structures directly or can encode them within a feature-value domain. This distinction exemplifies the difference between the direct versus indirect encoding of analyses.

abstraction of the notion of informational element which admits of several combinatorial operations currently in use in linguistics. For example, consider the combination of two sets of feature/value pairs which 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, recursively combining these values. This mode of combination can be defined formally as a graph-combining process to reflect this informal description, and is called *unification*, a primary operation of functional unification grammar (FUG), lexical-functional grammar (LFG), generalized phrase-structure grammar (GPSG), and definite-clause grammar (DCG). Other operations (e.g., generalization, disjunction, and overwriting) can be similarly defined.

3.2.2 Expressiveness

In Section 5 we will discuss mathematical measures of the expressiveness of a particular formalism falling within this methodological genus. The following list is intended to help the reader develop an intuitive appreciation of the breadth and diversity of formalisms that express analyses in this manner.

Categorial grammar: A pure categorial grammar, allowing forward application only, uses string concatenation to form constituents. The informational elements are complex categories which 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 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.

Ades/Steedman grammar: Similarly, the categorial system of Ades and Steedman [1], although including forward and backward application and composition, still fits within this class.

Montague grammar: Montague grammars (e.g., [15]) are directly stated as pairings of string-combining and denotation-combining rules—and as such fall squarely within this genus. The informational elements can be thought of as being comprised by a complex category feature

(as described above for categorial grammar) and a feature whose value is the denotation of the expression.

GPSG: GPSG [7] (as well as LFG and DCG), since it uses a context-free base, involves only concatenation to build up the surface string. Its feature system is a straightforward feature/value system, involving both simply-valued features (e.g., *number, case*) and complex-valued features (e.g., *slash, refl*).[9]

Head grammars: Head grammars [19] and head-driven phrase-structure grammars (HPSG) [20] extend GPSG by 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.

LFG: LFG's [8] informational structures, 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).

FUG: Through FUG's [12] patterns, concatenation is mandated as the constituent-forming operation.[10] Functional structures, as the informational entities, are a generalized feature/value system.

DCG: Terms are the basic information-bearing structures in DCG [17]. 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, and three argument features with respective values a, b, and c.

In fact, viewed from a computational perspective, it is not surprising that such a broad class of analyses can be directly encoded with generalized feature/value structures of this sort. Structures of exactly this kind have been put forward by various computer scientists as general mechanisms for

[9]The use of metarules requires some flexibility in interpreting this paradigm. 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 rather to establish generalizations in the lexicon.

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

knowledge representation [2] and data structures [5]. Thus, we have hardly constrained ourselves at all by being 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.

3.2.3 Computational Effectiveness

Ideally, we would like our formalism to be able to model any analysis within this class. Unfortunately, computational limitations require us to be more modest in our approach. Instead the formalism we will discuss is a first-order approximation to the general case of inductively defined complex-feature-based surface analyses—albeit the most general such approximation achievable within our current capabilities.

The constraints we impose in the name of computational effectiveness 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, as discussed in Section 5).

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 [3] and substitution [15]. 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 3.1) 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, and its typical applications—pattern-matching,

equality testing, and feature passing—are found in an even wider range of linguistic analyses. Furthermore, unification can be used to model analyses with many other combining operations, and can sometimes even substitute for nonconcatenative string operations.

Keep in mind that these constraints are *technological* in nature, *not* linguistic. As we become better able to provide rigorous definitions of computationally effective formalisms that overcome such constraints, they will ipso facto be reduced. In fact, certain relaxations of these constraints are already known to be feasible. Efficient algorithms for parsing formalisms that augment concatenation with head-wrapping operations are known [19]. Certain of our own tools allow, in addition to unification, operations of disjunction, negation and overwriting.[11]

4 PATR-II

We have developed the PATR-II formalism to embody these design decisions in an actual grammar formalism. PATR-II is a language for writing grammars that makes exactly those choices that were outlined in the preceding sections. The quite simple syntax of the PATR-II formalism has been discussed in previous work, as has its use in modeling various syntactic and semantic phenomena [26,24]; in addition, its semantics has been rigorously defined by applying the techniques of Scott's domain theory [16]. We will discuss the PATR-II formalism itself only briefly here, offering as an example the construction of a grammar fragment that embodies an analysis of agreement and control that loosely resembles LFG.[12] An appendix discusses

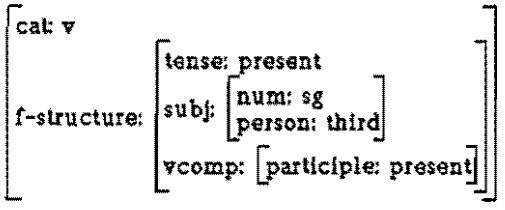
[11]Overwriting is a noncommutative operation akin to unification except that in the case of unification "clashes" one of the operands (say, the rightmost) is given precedence. Overwriting, certain types of negation, and generalization all have the unfortunate property of eliminating order-independence and magnifying the difficulty of providing simple denotational semantics for the formalisms. Nonetheless, they can prove useful if used sparingly. In particular, Karttunen [9] discusses linguistic motivation for negation and disjunction.

[12]This example is for expository purposes only and is not being advanced as the recommended modeling of LFG in PATR-II. Though it does follow rather closely analyses of Bresnan and Kaplan ([8], p. 206), we have ignored aspects (such as semantic forms) which require more complex encodings. See [26] for a discussion of semantics in PATR-II. Also, see Figure 1 for an example of control that manifests itself in the semantics.

the implemented linguistic computer tool for which the formalism serves as the basis.

4.1 Feature System

The informational structure associated with phrases in PATR-II is the *dag* (an acronym that will be elucidated below), which is a straightforward generalization of feature/value systems. Dags can be thought of as sets of feature/value pairs, in which the values are drawn from a finite set of atomic symbols plus the set of dags themselves. This view of dags as sets of feature/value pairs allows for a notation—akin to the functional-structure notation of FUG or the f-structure notation of LFG—in which the set of feature/value pairs is listed within square brackets, with a colon separating the feature label from the value (which is itself notated in this manner). To model the information LFG associates with a constituent, we might use a feature *cat* for the syntactic category, and a feature *f-structure* for the f-structure, with the latter itself having such features as *subj*, *obj*, *tense*, and *num*. The dag notated as

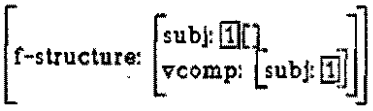


might be the informational structure associated with a third-person, singular, present tense verb such as "is" (though we have purposefully left out the subject control information). The fact that the feature values are themselves structured leads us to the term "complex-feature-based formalism," to avoid confusion with simple feature systems in which values are required to be atomic—namely, systems based on so-called "feature bundles." The former obviously subsume the latter.

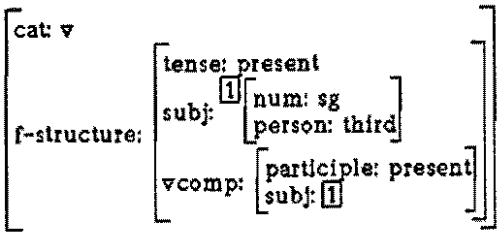
An important property of dags is that two features can share the same subdag as their common value. This leads to feature elements having a reentrant nature, that is, one can arrive at a given node by following more than one path in the dag. When such a node is instantiated further through

unification, this new information is visible whichever of the paths one reaches the dag by.

Making use of these properties, we could express the fact that a verb such as "is" displays subject control by unifying with the verb's dag the following "subject control" dag:



The boxed numbers mark the reentrancy, indicating that the values of the two features are the same. Thus, if information is added to one, it will affect the other as well. Combining this information with the previous dag for "is" through the process of unification, we get



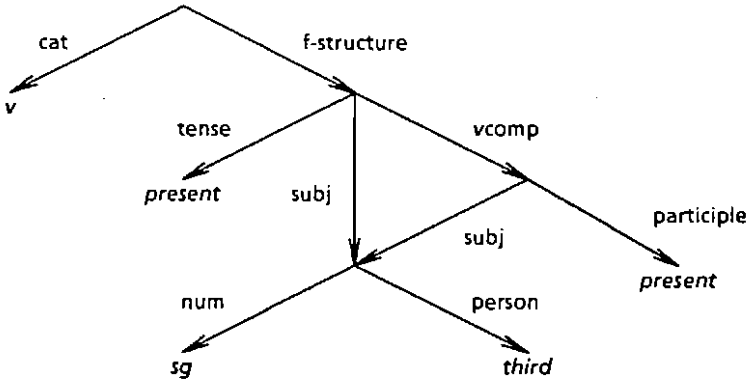
Note that the unification of the reentrant dag has caused the verb's subject features to be placed on the subject of the verb's participial complement. The rationale for placing the subject control information in a separate dag will be discussed in Section 4.3.1.

4.1.1 Feature structures as graphs

Dags can be viewed as rooted, directed, acyclic[13] 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.

[13]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*)) =↓.

The dag notated above would be expressed in a graph-structural notation[14] as



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 possible 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. As mentioned in Section 3.2.3, we distinguish unification as the combining operation on dags. From an intuitive standpoint, unification of dags corresponds to aggregating the information in the dags. It was used initially in logic and theorem-proving research, more recently finding its way into the linguistic theater as a basic operation in LFG, FUG and GPSG.

The dag notion is thus the generalization of feature/value systems that PATR-II uses as the basic informational structure. In keeping with the general linguistic methodology outlined in Section 3, elements from this domain of dags are recursively associated with phrases by using the operation of unification to combine the information from constituent dags. How these combinatory rules are specified is the topic of the next section.

[14]Reentrancy in the graph corresponds to coindexing in the feature matrix notation.

4.2 Grammar Rules

PATR-II grammars consist of specifications of the rules of combination. Recall that the basic string-combining operation is concatenation and that the basic dag-combining operation is unification. A combinatory rule must therefore specify how the dag associated with the whole string is related to the dags that are associated with the concatenated substrings. This is done in PATR-II by a rule consisting of a context-free base with a set of unifications. For example, the following is a well-formed PATR-II rule.

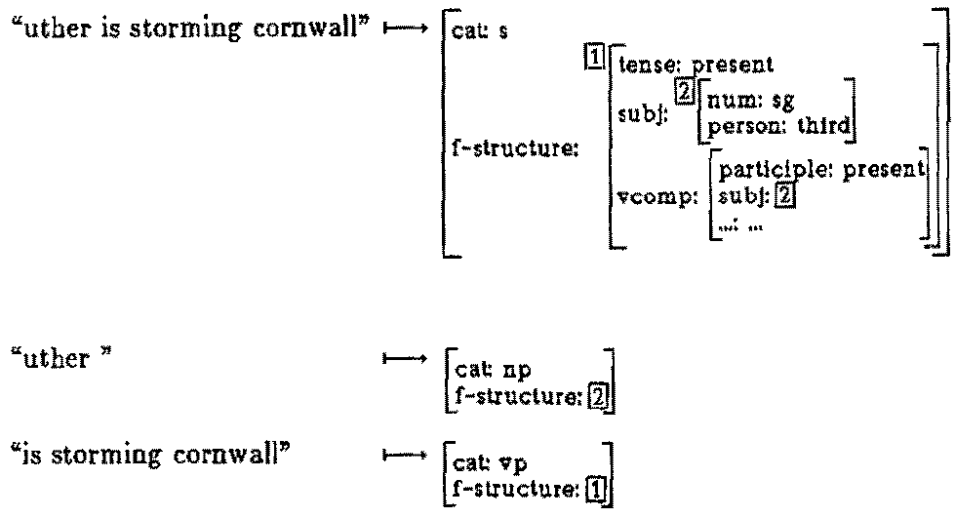
S → NP VP
<S f-structure> = <VP f-structure>
<S f-structure subj> = <NP f-structure>

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.[15] Furthermore, the first unification requires that the f-structure associated with the VP be equal to (because unified with) the f-structure of the S. Finally, the subject feature of the S is equal to the f-structure of the NP.

For these unifications to succeed, the f-structure associated with the NP would have to be compatible with the VP's subject feature. Given the reentrancy in the dag shown above for the VP, this in turn requires compatibility with the subject of the VP's verbal complement. In other words, the subject NP fills the role of the complement's subject.

As an example of string/dag pairs admitted by this rule, consider the following pairings.

[15]The treatment of *cat* features in this special manner (requiring their presence and atomicity) is the only further technological limitation on the general characterization of rule combination presented in Section 3.2.3. Other than this restriction—i.e., that every constituent have a value for the *cat* feature—any combinatory rule involving concatenation of strings and unification of dags can be expressed in PATR-II. The most recent implementation even removes this constraint by allowing the use of a special nonterminal *X* in the context-free base that imposes no restriction upon the *cat* feature, thus regaining full generality. The original limitation derived from the need for efficient parsing algorithms; only recently has parsing without it become feasible [25].



Note that the NP is marked as being masculine in gender. Because of the reentrancy in the dag for "is" the subject of the *vcomp* is also marked as masculine, so that on the assumption that reflexives agree in gender with the subject of the enclosing f-structure, only the reflexive form "himself" will be allowed. Similarly, semantic effects of control can follow directly from this approach (as in Figure 1). Thus, this rule, in coordination with the dags presented above, models a fragment of an LFG-like analysis of subject control.

4.3 Using PATR-II to model analyses

With just these simple tools, a wide variety of analyses can be encoded in PATR-II. Some are directly statable; others require that the devices in PATR-II be used to model indirectly those employed in the analysis. To facilitate such modeling, a further set of tools is added to the implementation that allows tailoring the system to a particular style of analysis. Development of these tools is just beginning. Consequently, they have been geared towards modeling the style of analysis we have been most interested in, namely, those with a lexical orientation. With this caveat, we will discuss briefly *templates* and *lexical rules*, two devices for capturing lexical generalizations in this framework.

4.3.1 Templates

Dags similar to that shown above for present tense, third-singular Vs might be employed quite frequently in, say, lexical entries for verbs. In particular, such a template could be associated with the verbal suffix "-s" and made use of during morphological analysis. By defining dag *templates*, the user can build up a library of such frequently utilized dags. For instance, we can define a template as follows:

```
Let Pres3Sing be
        <f-structure tense> = present
        <f-structure subj person> = third
        <f-structure subj num> = sg.
```

We might want templates for the notions "verb" and "subject control".

```
Let Verb be
        <cat> = v.
Let SubjectControl be
        <f-structure subj> = <f-structure vcomp subj>.
```

Alternatively, we can use templates to define a hierarchy of such concepts:

```
Let PresTense be
        <f-structure tense> = present.
Let 3Sing be
        <f-structure subj num> = sg
        <f-structure subj person> = third.
Let Pres3Sing be
        PresTense
        3Sing.
```

Defining a verb as being Pres3Sing and SubjectControl (and adding participial-form information) will thus associate with it the dag presented in Section 4.1.1. By appropriate definition of lexical templates we can encode assumptions and generalizations about the interrelationships of linguistically salient notions.

4.3.2 Lexical Rules

Lexical rules are an even more powerful mechanism for manipulating the dags associated with lexical entries. They provide a way of actually decomposing and restructuring dags, while still using unification as the basic combinatory operation. Once again, we present the concept by example. Consider an LFG-like definition of passive, one in which the object subdag becomes the subject. We could define a lexical rule to model this analysis as follows:

Define AgentlessPassive as
<out cat> = <in cat>
<out f-structure participle> = passive
<out f-structure subj> = <in f-structure obj>
<out f-structure obj> = nil.

This rule builds a passive lexical entry (referred to as *out*) from an active entry (*in*) such that the category feature information remains the same, but the subject and object features have been changed appropriately. Lexical rules have been used in PATR-II grammars for treating passives, "there" insertion, extraposition and other phenomena commonly viewed as relation-changing.

4.4 How Has PATR-II Been Used?

The PATR-II formalism has been used to build grammars for fragments of English with steadily increasing coverage. We have experimented with grammars covering a range of styles of analysis, from phrase-structural to categorial, from highly lexical to predominantly syntactic. To convey an intuitive sense of the expressive power of the formalism, we list here some samples of the kinds of phenomena we have dealt with in our computer implementation.

- *Verbal subcategorization* for NPs, PPs, $\bar{S}$s, VPs, including raising and equi phenomena, syntactic control, and auxiliary structure.
- *Relation-changing rules*, including active/passive, "there" insertion, and extraposition.

- *Unbounded dependencies* including Wh-movement and relative clauses.
- *Complex NPs and PPs.*
- *Adverbials* of certain types.
- *Semantics* for these constructs, given as encodings of logical formulae in dag form.

Though by no means an exhaustive list of the coverage of our grammars, this should provide evidence that nontrivial linguistic constructs can be described effectively in PATR-II. The reader is urged to refer to previous publications [26,9] for a more thorough discussion as to how some of these phenomena can be modeled and how the system is used for semantic interpretation as well.

5 Conclusion

Looking back at the criteria of Section 3.1, we see that PATR-II meets them in the following way:

Linguistic felicity: PATR-II has a completely declarative interpretation (made explicit in its denotational semantics [16]) that allows rules in the grammars to be thought of modularly—as separate, independent constraints on a natural language—as is the common view of such rules in linguistics. It is similar to several of the popular grammatical formalisms in use in linguistics and artificial intelligence (including many of those listed in Section 3.2.2), and can be used to directly model analyses from them.

Expressiveness: A precise, albeit unenlightening, characterization of the expressive power of PATR-II can be gleaned from the existence of PATR-II grammars for any recursively enumerable language. This puts them into the most powerful class of the Chomsky hierarchy, well in excess of context-free power [26]. Of course, not all languages are expressed with equal ease, since the formalism is designed to facilitate stating the kinds of constructs found in natural languages. But the broad class of formalisms listed in Section 3.2.2 seems amenable to modeling with PATR-II. To the extent that this is so, PATR-II should

be considered a successful point in the space of design alternatives discussed in this paper.

Computational effectiveness: The simplicity of the PATR-II language's formal definition enables a degree of rigor not normally found in grammatical formalisms. It is its simplicity which allows a denotational semantics for the formalism to be given (which, incidentally, can thereby provide a semantics for the other grammar formalisms it models).

There exist algorithms for implementing parsers for grammars written in PATR-II (i.e., programs that provide a procedural interpretation of the declarative semantics). Since PATR-II is a completely declarative formalism (and thus interpretation of grammars is independent of the order of processing), various algorithms can be (and have been) used for parsing, including top-down backtrack parsing, Earley's algorithm, the Cocke-Kasami-Younger algorithm, and, most recently, an extended Earley's algorithm designed especially for such complex-feature-based formalisms [25]. Efforts to implement a wide range of the aforementioned related formalisms (such as current work being done not only at SRI, but also at Hewlett-Packard and Xerox) are constantly improving the efficiency of these algorithms.

In a paper discussing computer tools for linguistics, it may seem ironic that we have emphasized the design of a specific grammar formalism—a language for encoding linguistic analyses—relegating to an appendix any mention of its use as the basis for an actual implemented tool for testing linguistic analyses. This emphasis stems from our particular perspective: that the critical properties of such tools are their linguistic felicity, their expressiveness, and their computational effectiveness; that these considerations make the choice of grammar formalism of paramount importance; and that they should be used not only to evaluate such tools, but also to guide their design.

The computer as linguistic tool is a powerful concept. Our hope is that linguists will take full advantage of this impartial mirror, this theoretical touchstone, this liberating straitjacket.

A The PATR-II Experimental System

The PATR-II formalism is the basis of several implementations. The PATR-II Experimental System is one of these, an implemented computer tool for building and testing grammars. Researchers at SRI have been using it to develop front ends for the KLAUS natural-language-processing system.[16] It supports all the functionality presupposed by this and earlier papers on the PATR-II formalism (and includes some capabilities not discussed therein). Written in Zetalisp for the Symbolics 3600 Lisp Machine, the PATR-II Experimental System uses the style of window- and mouse-oriented user interface characteristic of that machine.

The functionality of the PATR-II Experimental System can be roughly divided into two classes:

Analysis: The system can analyze sentences with respect to a PATR-II grammar, in the process developing the pairings of the sentence and its subconstituents with their corresponding dags.

Grammar development: The system allows grammars to be edited and compiled, and information derived from sentence analysis to be displayed, perused and traced.

A.1 Analysis of sentences

Given a grammar written in the PATR-II formalism, the system is able to analyze sentences with respect to a grammar by using chart-parsing algorithms designed for this formalism. Every analysis leaves behind it a *chart* of information concerning complete and partial subphrases formed [11], along with their associated dag. By using the grammar information to combine these subphrases, increasingly longer phrases (with their dags) can be constructed, possibly culminating in deriving a dag (or, in the case of ambiguous sentences, dags) corresponding to the sentence as a whole. If the language

[16] Other implementations are discussed in more detail in [26]. These include an Earley algorithm parser written in Prolog by Fernando Pereira that uses structure-sharing dag representations and a version for the DEC 20 computer in Interlisp that uses a variant of the Cocke-Kasami-Younger parsing algorithm.

Unfortunately, this software is not presently available from SRI International, since the system is highly experimental and under constant development.

of the grammar does not include the sentence (i.e., the sentence is ungrammatical) then the parser will allow no such derivation.

Two parsing algorithms are currently incorporated into the system, the user determining which one is to be actually invoked for parsing.[17] One is a left-corner parsing algorithm with top-down filtering that is based on the work of John Bear [4]; the other is an extended version of Earley's algorithm specifically designed for complex-feature-based formalisms [25], that increases the amount of top-down filtering over that available from either the left-corner algorithm or Earley's.

Morphological analysis of the input sentence is carried out by an analyzer written by Bear; it is based on the two-level morphological model of Koskenniemi [13] and related work by Karttunen [10]. Morphological analysis of the individual words yields a list of morphemes and their associated template dags or lexical rules. The combination of these templates and lexical rules yields the dags associated with the words themselves. These pairings serve as the basis for the grammar's inductive definition of phrase/dag pairing.

A.2 Grammar Development

Besides being able to analyze sentences, the system makes available a set of tools for extracting information from the parse and interactively building and modifying the grammar. The grammar development tools provide for:

Grammar compiling: Grammars written in PATR-II can be compiled into tables suitable for use by the parsers.

Chart and grammar perusal: The chart, grammar rules, lexical items, etc. can be displayed by means of a mouse-oriented, "browsing" mode of interaction.

Grammar updating: Rules can be edited using a general-purpose editor (ZMACS) and the changes compiled *incrementally* for immediate availability and testability.

Tracing: Rules can be traced, so that each invocation during parsing displays information to the user concerning that invocation.

[17] Since we are also trying out different parsing algorithms, this allows us to compare them directly with one another.

All of these services are available through a consistent graphic user interface; operations are chosen by means of a "mouse", with which a menu- and icon-based interface is controlled. Figure 1 shows a snapshot of the user interface after parsing a sentence. The user has displayed one of the passive edges developed during the parse. The mouse cursor is situated over an icon representing the rule used in building this edge, and the icon has been highlighted by a circumscribing box. By clicking the mouse buttons, the user can cause this rule to be displayed, edited, traced, and so forth. Similar operations are possible for the other types of information the system manipulates, information concerning words, morphemes, edges, rules, templates, lexical rules, etc.

References

[1] Ades, A. E. and M. J. Steedman. On the order of words. *Linguistics and Philosophy*, 4(4):517–558, 1982.

[2] Ait-Kaci, H. *A New Model of Computation Based on a Calculus of Type Subsumption.* PhD thesis, University of Pennsylvania, 1985.

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

[4] Bear, J. S. *A Breadth First Syntactic Component.* Master's thesis, University of Texas, Austin, Texas, May 1981.

[5] Cardelli, L. *A Semantics of Multiple Inheritance.* Technical Report, Bell Laboratories, Murray Hill, New Jersey, 1984.

[6] Chomsky, N. *Lectures on Government and Binding.* Foris Publications, Dordrecht, Holland, 1982.

[7] Gazdar, G., E. Klein, G. K. Pullum, and I. A. Sag. *Generalized Phrase Structure Grammar.* Blackwell Publishing, Oxford, England, and Harvard University Press, Cambridge, Massachusetts, 1985.

[8] Kaplan, R. and J. Bresnan. Lexical-functional grammar: a formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, Massachusetts, 1983.

[9] Karttunen, L. Features and values. In *Proceedings of the Tenth International Conference on Computational Linguistics*, Stanford University, Stanford, California, 2-7 July 1984.

[10] Karttunen, L. KIMMO: a general morphological processor. *Texas Linguistic Forum*, 22:161–185, December 1983.

[11] Kay, M. *Algorithm Schemata and Data Structures in Syntactic Processing.* Technical Report, Xerox Palo Alto Research Center, Palo Alto, California, 1980. A version will appear in the proceedings of the Nobel Symposium on Text Processing, Gothenburg, 1980.

[12] Kay, M. *Unification Grammar.* Technical Report, Xerox Palo Alto Research Center, Palo Alto, California, 1983.

[13] Koskenniemi, K. *A Two-Level Model for Morphological Analysis and Synthesis.* PhD thesis, University of Helsinki, Helsinki, Finland, 1983.

[14] Langendoen, T. D. and P. Postal. *The Vastness of Natural Language.* Blackwell, Oxford, England, 1984.

[15] Montague, R. The proper treatment of quantification in ordinary English. In R. H. Thomason, editor, *Formal Philosophy*, pages 188–221, Yale University Press, New Haven, Connecticut, 1974.

[16] Pereira, F. C. N. and S. M. Shieber. The semantics of grammar formalisms seen as computer languages. In *Proceedings of the Tenth International Conference on Computational Linguistics*, Stanford University, Stanford, California, 2-7 July 1984.

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

[18] Pereira, F. C. N. and D. H. D. Warren. Parsing as deduction. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 137–144, Massachusetts Institute of Technology, Cambridge, Massachusetts, 15-17 June 1983.

[19] Pollard, C. *Generalized Phrase Structure Grammars, Head Grammars, and Natural Languages.* PhD thesis, Stanford University, Stanford, California, 1984.

[20] Pollard, C. Lecture notes on head-driven phrase-structure grammar. February 1985. Center for the Study of Language and Information, unpublished.

[21] Pullum, G. K. Personal communication, 1985.

[22] Robinson, J. J. DIAGRAM: a grammar for dialogues. *Communications of the ACM*, 25(1):27–47, January 1982.

[23] Sager, N. *Natural Language Information Processing.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1981.

[24] Shieber, S. M. The design of a computer language for linguistic information. In *Proceedings of the Tenth International Conference on Computational Linguistics*, Stanford University, Stanford, California, 2-7 July 1984.

[25] Shieber, S. M. 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, July 1985.

[26] Shieber, S. M., H. Uszkoreit, F. C. N. Pereira, J. J. Robinson, and M. Tyson. The formalism and implementation of PATR-II. In *Research on Interactive Acquisition and Use of Knowledge*, SRI International, Menlo Park, California, 1983.

[27] Woods, W. Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10), October 1970.

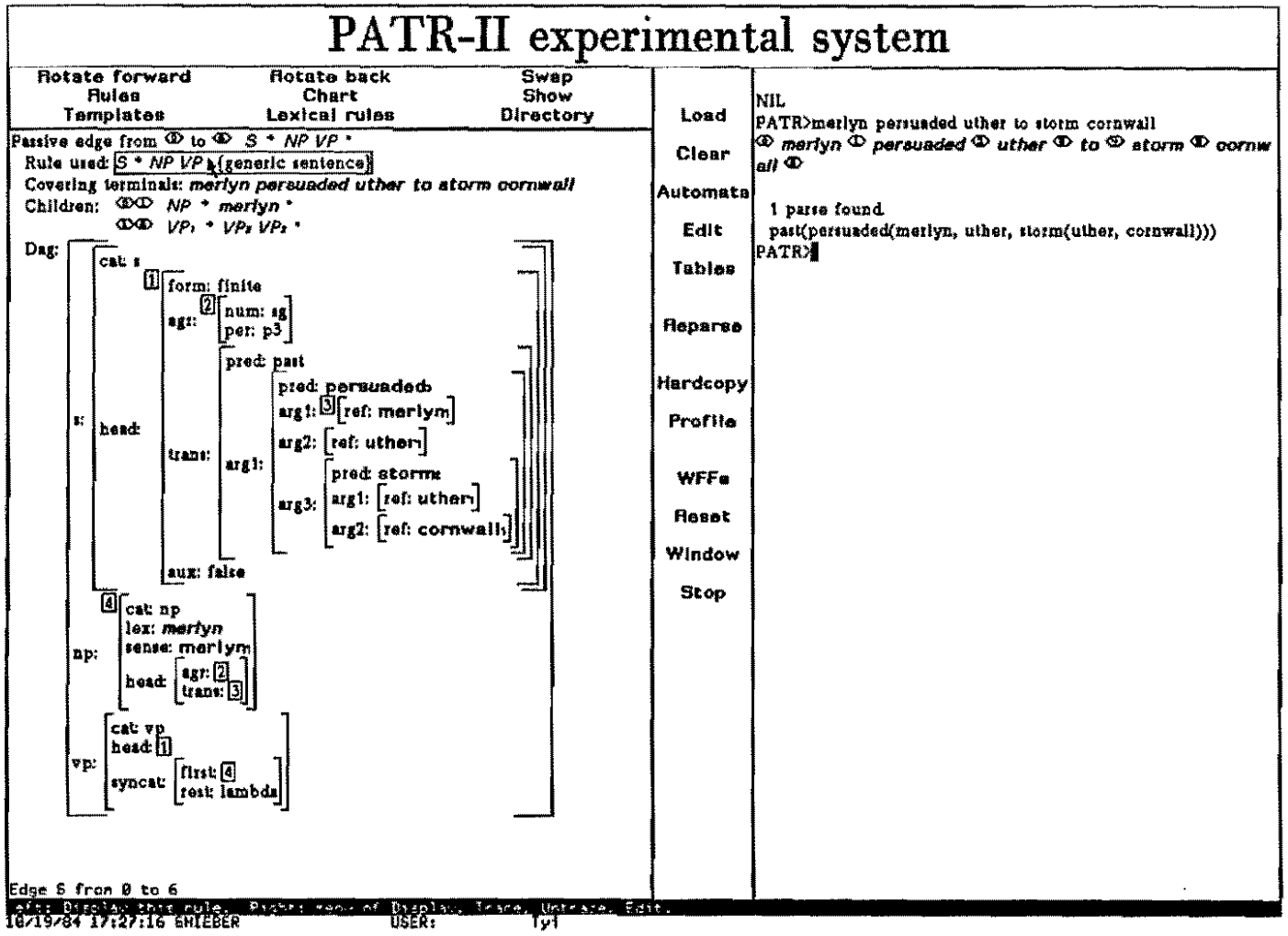


Figure 1: Snapshot of the PATR-II Experimental System after parsing a sentence and displaying one of the passive edges built.