SRI International

SEPARATING LINGUISTIC ANALYSES FROM LINGUISTIC THEORIES

Technical Note 422

May 28, 1987

By: Stuart M. Shieber, Computer Scientist

Artificial Intelligence Center
Computer and Information Sciences Division

Center for the Study of Language and Information
Stanford University

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

This research has been made possible by a gift from the System Development Foundation. An earlier shorter version was presented at the Alvey/ICL Workkshop on Linguistic Theory and Computer Applications, held at the University of Manchester Institute of Science and Technology, Manchester, England, in September 1985, and appears in the proceedings published by Academic Press.

SRI International®

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

Abstract

This paper explores the question of what level of linguistic practice is best suited for use in natural-language processing (NLP) efforts. In particular, I argue that because the goals and strategies of linguistic theory and NLP differ, linguistic theories and formalisms may be inappropriate for importation into NLP applications. The question of whether insights from linguistic analyses can still be taken advantage of without such importation is elucidated by making use of notational reductions from one formalism to another, thereby helping demonstrate the independence of analyses from formalisms. Several such reductions are informally discussed, and one, a reduction from the formalism used in work on generalized phrase-structure grammar to the PATR-II formalism is presented in more detail.

Contents

1 Introduction

Many of the papers in this volume stress the desirability of making computational use of insights from linguistics as an aid in the processing of natural language. Groups of researchers throughout the world are engaged in efforts that are often described as implementing a linguistic theory. In this paper I will start out by addressing the question of what *level* of linguistic practice is best for natural-language-processing (NLP) efforts[1], whether results from linguistic theory (LT) should be utilized by modeling a particular linguistic theory, interpreting a linguistic formalism, or embodying a linguistic analysis. That is, I will investigate what the role of linguistic theories and their associated formalisms and analyses should be in the engineering discipline of developing NLP applications. In particular, I will contend that, because the goals and strategies of LT and NLP research differ, linguistic theories and formalisms may be inappropriate for importation into NLP applications.

Nonetheless, I will adopt the possibly controversial position that NLP efforts can benefit from insights expressed in linguistic analyses of particular languages.[2] For instance, in writing systems to interpret English sentences that include, say, the comparative construction, a system builder or grammar writer might find it beneficial to make use of one or another of the

This research has been made possible by a gift from the System Development Foundation. I am indebted to John Bear, David Israel, Lauri Karttunen, Fernando Pereira, and Ray Perrault for their comments on earlier drafts of this paper and to Gerald Gazdar, Ivan Sag, Henry Thompson, and members of the Foundations of Grammar project at the Center for the Study of Language and Information for their helpful discussions during the development of this work.

An earlier shorter version of this paper was presented at the Alvey/ICL Workshop on Linguistic Theory and Computer Applications, held at the University of Manchester Institute of Science and Technology, Manchester, England, in September 1985, and appears in the proceedings published by Academic Press. I thank the organizers of that workshop for permission to publish portions of that paper here.

[1] In this paper, I will use "NLP" specifically to refer to computer applications requiring the processing of natural language, and not to the important, but separate, issues of computational or mathematical approaches to linguistics.

[2] That this view is controversial can be seen in an opinion shared by many NLP researchers, and voiced, for example, in Roger Schank's comments to the effect that "research on syntax should have stopped fifteen years ago" and that "the hard part of natural language is not the language" [17, p. 166]. Although it is true that there are exceptionally difficult problems in NLP research that are not traditionally dealt with by linguistics and about which "linguistic theory" as construed by linguists has little to say, it does not follow that those problems that are addressed by linguistics are straightforward or that linguists' expertise is not pertinent to NLP efforts.

previously described linguistic analyses of this construction. Given that this is so, the question arises as to what sort of commitment to linguistic theories and formalisms this necessitates, that is, to what extent analyses are dependent on the particular formulations found in the linguistic literature.

An instructive analogy can be drawn with the relationship of computer science to computer programming. Consider the following rewording of the previous paragraph:

> I will adopt the uncontroversial viewpoint that computer programming can benefit from using algorithms from computer science. For instance, in writing a system to, say, sort records in a database, a system builder or programmer might find it beneficial to make use of one or another of the previously described sorting algorithms. Given that this is so, the question arises as to what sort of commitment to programming languages this necessitates, that is, to what extent algorithms are dependent on the particular formulations of them found in the computer science literature.

This latter question is easy to answer. Algorithms from computer science are not at all tied to the particular formulations in which they are described. Despite the fact that quicksort was first described in ALGOL, it can still be used by programmers writing in Pascal. Indeed, many algorithms were originally described only informally, e.g., Earley's algorithm. Nonetheless, any effective procedure has an implementation in some programming language, that is, if we believe Church. And since any program in a given programming language can be reduced to an equivalent program in any other programming language (of sufficient power), it follows that they have implementations in all programming languages. This ability to reduce one programming language to another thus demonstrates that the use of algorithms from computer science does not in theory tie us to any particular language for expressing algorithms.

Similarly, the issue of whether a particular linguistic analysis requires allegiance to a given language for expressing analyses, i.e., grammar formalism, can be at least partially elucidated by exhibiting *reductions* from one linguistic formalism to another. As a side effect, such reductions help establish exactly which differences among linguistic theories that use the

formalisms are purely (though not merely) *notational*, and, on the other hand, which ones are *notional*.

Reductions from one formalism to another can convey several types of information of different degrees of granularity. At a coarse grain, a reduction from one formalism to another can precisely convey their relative computational power and complexity. Reductions showing the interreducibility of programming languages or NP-complete problems are of this type. This use of reductions gives evidence of what might be called the *absolute expressivity* of formalisms.

Second, the existence or nonexistence of certain reductions can give evidence at a finer grain of *functional expressivity*. Rather than deciding whether a problem in one formalism can be *encoded* in another, this type of inquiry determines whether the problem can be *directly stated* in the other formalism. In such a reduction, we disallow reencoding the problem statement in the language of the other formalism. Two formalisms that are expressively equivalent in the absolute sense may nevertheless not be equivalent in functional expressivity; the prime example of such a pair of systems is the simple and second-order typed lambda calculi. Although both calculi are Turing-equivalent, it can be shown that the second-order system can directly compute functions that can be computed only indirectly in a simply typed system [6].

Finally, a reduction from one formalism to another can elucidate their *notational expressivity*. Consider two programming languages, say, Pascal and Pascal′, a minor variant that lacks the *while...do...* construct but is otherwise identical. Clearly, programs in either language can be reduced to equivalent programs in the other. The reduction from Pascal′ to Pascal is, in fact, the identity reduction, whereas the reverse direction requires encoding of *while...do...* statements by means of some other construct, e.g., *repeat...until....* That the latter reduction is more complex than the former shows the intuitively obvious fact that Pascal is more notationally expressive than Pascal′ (though not more functionally or absolutely expressive). Though this example for programming languages is trivial, the use of the same technique for linguistic formalisms can be quite revealing, especially considering that the question of the precise relationships among the theories and formalisms has not been actively investigated. The details of reductions can show quite graphically just how close various formalisms are to one another. The simpler the reductions are, the closer the formalisms. In other words, reductions among formalisms can not only demonstrate what things

can be expressed in the various formalisms, but how easy those things are to express.

The question of whether linguistic analyses are separable from their formulations within particular linguistic theories and formalisms is much less easily answered than its computer language analogue, even though the technique used—notational reductions—is the same. The answer depends first of all on *what* the distinctions are among the concepts of analysis, formalism and theory, concepts whose meanings are not always agreed upon, let alone made precise in the linguistics literature. Second, it requires serious research on exactly *how* formalisms and theories can be interreduced and what properties the reductions possess. Finally, even if such a separation could be demonstrated—by exhibiting certain complete or partial reductions among linguistic formalisms—it remains to be shown *why* this separation would aid in NLP efforts.

This paper will be divided, therefore, into three parts corresponding roughly to these three questions: what, why, and how. First, I will briefly discuss the notions of theory, formalism, and analysis as they are employed in linguistics and other fields in an effort to clarify the terminology. Second, I will provide some methodological justification for the attempt to separate analyses from their formal and theoretical underpinnings. Third, I will describe some work in progress on characterizing the relations among certain formalisms within theories of syntax. I will concentrate on syntax because it is the subfield of linguistics most familiar to me. In particular, I will stress the lexical-functional grammar (LFG) and generalized phrase structure grammar (GPSG) theories as these seem to be the major "contenders" for use by people interested in utilizing syntactic theories for NLP computer applications. The reductions I will discuss concern the formalisms found in LFG and GPSG, and certain formalisms from computational linguistics—namely, the functional unification grammar (FUG) formalism of Martin Kay [10], and the PATR formalism from SRI International [13].

2 What: Some Terminology

Before discussing the separation of linguistic analyses from formalisms and theories, I need to make clear what I (and linguists themselves) mean by the terms *formalism*, *theory*, and *analysis*—especially because they are not

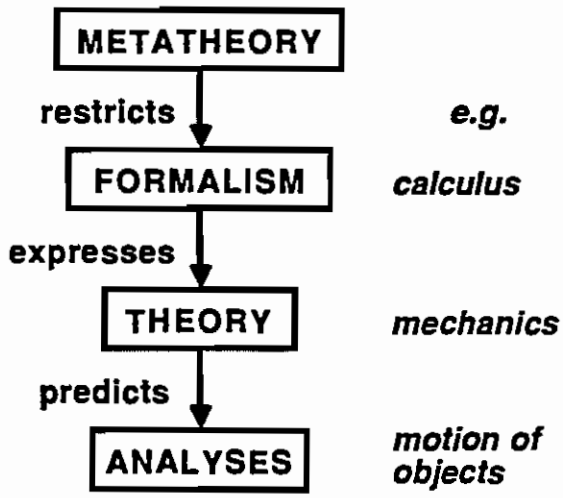

Figure 1: Relations Among Terms

always well differentiated in the linguistic literature and because they are used in linguistics in relatively nonstandard ways vis-à-vis other disciplines.

In many scientific fields, the scientist is interested in building a *theory* of some observable phenomena (say, the laws governing the motion of objects). He may use a special notation or *formalism*—the calculus, for instance—to write down one of several possible theories of motion. These theories in turn predict *analyses* of certain specific observable phenomena (given appropriate contingent facts such as boundary conditions); for instance, the motion of a ball thrown into the air or of people jumping in elevators can be analyzed by a theory of kinematics. The notations for describing theories are typically designed to be as general as possible, thereby allowing a broad spectrum of theories to be stated and facilitating comparison among them. A graphic representation of the relations among the concepts is given in Figure 1.

The terminology is used differently in linguistics. As before, attempts to characterize observable data are generally thought of as *analyses* (e.g., the analysis of a particular English sentence). But a description of an entire language (what would be referred to in nonlinguistic terminology as a theory of the language) usually falls under the label *analysis* as well (e.g., an analysis of English syntax). These analyses predict the observable phenomena,

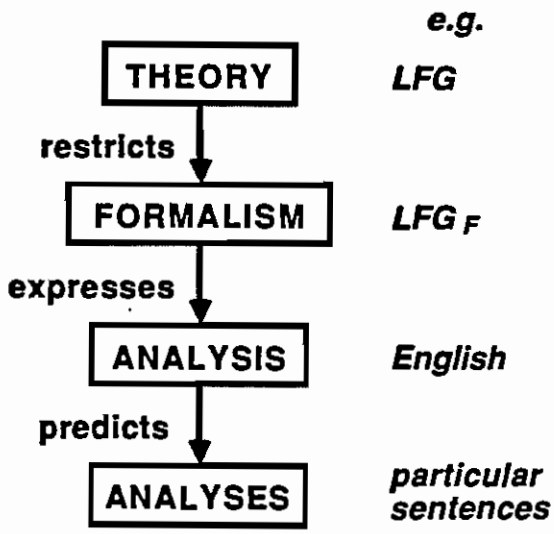

Figure 2: Relations Among Terms in Linguistics

e.g., the LFG analysis of English predicts the grammaticality and ambiguity properties of English sentences. The term *theory* is primarily reserved for a characterization of general properties of or relationships among analyses in this latter sense—what in other fields would be considered a *metatheory*. For instance, the study of possible geometries is not a theory of space, but a theory of theories of space, of which Euclidean geometry and Riemannian geometry are but two. There are metatheories in other fields as well, but in linguistics, unlike other fields, the metatheory is the subject's *raison d'être*.[3] A summary of the relations for linguistic terminology is pictured in Figure 2.

The differences, of course, are only in the words used to express the structure of the disciplines; the ideas themselves are much the same. But terminological differences like these can be confusing. For example, in other fields, theories are *expressed in* formalisms. In linguistics, analyses are expressed in formalisms and theories are expressed *alongside* formalisms. Since the primary objective of research in each case is development of a "theory", this distinction is important. The relationship between formalisms and the object of study in the two cases is quite different. In one case it is primarily

[3]One important exception is the field of mathematical logic. Since the discovery of the paradoxes, the metatheory of logic has been the primary study of logicians.

a tool, in the other a theoretical device.[4]

This analogy between the terminologies of linguistics and other sciences is not the only one that could be made. Linguistic theories do, for instance, share with physical theories the property of being unitary explanations for actually realized phenomena, a property that metatheories of physics would not share. The important point is that the various terms—metatheory, formalism, theory, analysis—are not interchangeable and can differ widely from one field to another.

In the rest of this paper I will follow linguistic practice in using the words *theory, formalism,* and *analysis*, rather than *metatheory, formalism,* and *theory*, respectively. Furthermore, to make explicit the distinction between formalisms and theories used in linguistics, I will henceforth use the acronym of the theory (say, LFG) to stand for the theory itself, and the acronym subscripted with an F (as in LFG_F) for the formalism associated with the theory. Note also that the notion of analysis is used ambiguously in linguistics jargon. In general, I will use the term *analysis* as vague (not ambiguous) between the two interpretations.

3 Why: Methodological Motivation

I now turn to the rationale for wanting to separate linguistic analyses from linguistic theories in the first place. Of course, there is the obvious intellectual advantage of thereby gaining a firmer understanding as to which differences among theories are intrinsic as opposed to emergent from notational aspects of the respective formalisms (i.e., which theoretical concepts from one formalism are difficult or impossible to state in another). But I am here concerned with the application of linguistic theory to NLP. In such an application, why not just incorporate into an NLP system an entire theory of linguistics, with its associated formalisms and analyses?

The reason is that the traditional methodological goals and strategies of research in linguistic theory and NLP are in direct conflict, and this conflict is reflected in actual practice in the two disciplines. In deciding whether

[4] Metatheoretic restrictions are of course embodied in the formalism used in stating theories in other sciences as well. Use of the calculus, for instance, presupposes that one's theory of space will have certain properties of continuity of fundamental magnitudes. However, such restrictions are seldom made explicit or studied in their own right outside of linguistics.

to incorporate a linguistic theory wholesale into an NLP system, we need to have an understanding of what the respective goals of LT and NLP are. Then, insofar as the goals of LT are consistent with and aid those of NLP, devices from the former can beneficially be incorporated into an NLP system. Where the goals are conflicting, however, the associated techniques should be avoided.

In this section, I identify these goals and strategies and show where they differ. In particular, I demonstrate that the goals linguists adhere to in their design of theories and their evaluation of formalisms diverge from those researchers desire for natural-language-processing purposes especially in the role of expressiveness of the theoretical constructs, and consider some implications of this divergence.

3.1 Goals for Linguistic Theories and Formalisms

3.1.1 Goals for Linguistic Theory

For reasons usually motivated by the problem of language acquisition, linguists have traditionally been interested specifically in universal properties of language. Chomsky summarizes the goals of research in linguistic theory thus:

> Let us recall the basic character of the problem we face. The theory of UG must meet two obvious conditions. On the one hand, it must be compatible with the diversity of existing (indeed, possible) grammars. At the same time, UG must be sufficiently constrained and restrictive in the options it permits to account for the fact that each of these grammars develops in the mind on the basis of quite limited evidence. [5, p. 3]

That is, a theory of linguistics should *characterize all and only the possible natural languages.* I have chosen a quite recent quotation to this effect, but, of course, the idea has pervaded the practice of linguistics since the appearance of *Syntactic Structures.*

Secondarily, of course, other requirements are usually imposed on the task, three examples of which I discuss here. First, it is typically desired that the characterization be "generative" in the sense of being explicit or rigorous. Chomsky defines the term thus:

> By a generative grammar I mean simply a system of rules that in some explicit and well-defined way assigns structural descriptions to sentences. [4, p. 8]

Second, Occam's razor tends to cut in favor of theories that admit relatively simple, elegant analyses. Although such a comparison is perforce based on aesthetic or subjective standards, it is hard to imagine any other criterion that could serve for comparison of theories that have the same expressive power.[5] Certainly, simplicity as a criterion for choosing theories is a common standard in other fields of research.[6]

In fact, we see this criterion of "linguistic felicity" used at least implicitly in linguistics. Gazdar notes its importance in choosing between base-generated and transformational approaches:

> If to do things exclusively by direct phrase structure generation was to lead inevitably to aesthetic disaster (re simplicity, economy, generality, empirical motivation, etc.) whilst competing transformational analyses were paragons of elegance and enlightenment, then one might reasonably feel inclined to reject the former in favour of the latter. [7, p. 134]

Finally, current practice seems to place a value on the declarative (or order-independent, or order-free) nature of the characterization. Kaplan and Bresnan write:

> The constraint of *order-free composition* is motivated by the fact that complete representations of local grammatical relations are effortlessly, fluently, and reliably constructed for arbitrary segments of sentences. [2, p. xlv]

Regardless of the soundness of this reasoning, the goal of declarativeness is shared by at least LFG, GPSG, and government-binding theory (GB).

[5] By expressive power, I mean to include not only absolute expressivity limitations, but notational expressivity as well, not only formal limitations on expressivity, but theoretical, noformal limitiations, whatever they might be.

[6] This notion of simplicity should not be confused with the sort of "simplicity metric" that, according to Chomskyan linguistics, governs a child's choice among grammars. Simplicity metrics of this sort, as Chomsky himself notes, cannot be used to choose among different theories.

These theories all define sentential grammaticality in terms of simultaneous well-formedness constraints, rather than procedures for recognition.

To summarize, then, the goals linguists pursue in designing their theories are, first and foremost, *completeness* (characterizing *all* the possible natural languages), and *restrictiveness* (characterizing *only* the possible natural languages); secondarily, they pursue such goals as *rigor, simplicity, elegance,* and *declarativeness.*

3.1.2 Goals for Linguistic Formalisms

These are the goals linguists have for their theories, but what of the goals they pursue in designing the associated formalisms? Current linguistic theories are usually most precise in the descriptions of their formalisms, in contradistinction to the unformalized "substantive" portion of the theory. It is generally impossible to determine if a computer system correctly and soundly realizes theoretical constructs that are not rigorously stated. Consequently, efforts to "implement a linguistic theory" typically, and for good reason, concern themselves only with that aspect of the theory embodied in, not stated with, the formalism. The question of what goals are pursued in the design of the formalisms therefore becomes especially important.

The answer to this question is complicated by the fact that the relationship between theory and formalism differs from theory to theory. One common notion is that the theory is fully determined by the formalism, that is, that the formalism itself should embody all the constraints on universal grammar. This view is explicitly propounded by the originators of GPSG:

> A grammatical framework can and should be construed as a formal language for specifying grammars of a particular kind. The syntax, and, more importantly, the semantics of that formal language constitute the substance of the theory or theories embodied in the framework. [8, p. 2]

The same view is also at least intimated by Bresnan and Kaplan:

> The lexical theory of grammar provides a formally explicit and coherent theory of how surface structures are related to representations of meaningful grammatical relations. [2, p. 4]

On the other hand, a view of grammar could be propounded in which the formalism is but one part of the theory. Constraints supplied by formal limitations (i.e., limitations of the formalism) would indeed constitute universal claims, but so would other constraints that, while not incorporated in the formalism, are nonetheless essential to the theory. This, I take it, is Bresnan's view, as expressed in the following excerpt:[7]

> Having no formal theory at all will lead to vague and inconsistent formulations at both the theoretical and descriptive levels. Despite its importance, however, a formal theory of grammar is only one step in the construction of a substantive linguistic theory of universal grammar. The present work adds to the theory of LFG a set of substantive postulates for a universal theory of control and complementation. [2, p. 282]

Thus, Bresnan distinguishes formal and substantive constraints in a theory,[8] whereas Kaplan [9] and Gazdar *et al.* would require all constraints to be formal. But, in either case, at least part of the burden of restrictiveness is placed on the formalism itself, that is, more restrictive formalisms are historically favored over less restrictive ones. If this were not so, arguments against LFG, say, on the basis of the overexpressiveness of LFG_F (cf. [1]) would have no force. Thus, the same criteria of restrictiveness, completeness, simplicity, rigor, and declarativeness that linguists apply to their theories are applied to their formalisms as well.

3.2 Goals for NLP Formalisms

What, then, of the goals of natural-language-processing efforts in computer applications? Simply put, such efforts aim to *characterize computationally one or a small number of languages.* As mentioned earlier, the grammar formalism, in this case at least, is of paramount importance, since substantive theory typically finds little place in computer application. For this reason, I will be concentrating on the desired characteristics of formalisms used

[7] I have assumed in interpreting this quotation that Bresnan uses the term "formal theory" to mean theoretical constraints embodied in the formalism.

[8] Chomsky also distinguishes notions of "formal" and "substantive"; however, these are slightly different in scope from Bresnan's usage. Though I find the choice of the term "substantive" unfortunate—formal constraints seem to me to be as substantive as any—I will continue to use Bresnan's terminology.

in NLP systems, that is, on the question of which design strategies NLP formalism designers should pursue.

Obviously, formalisms intended for NLP should be designed to facilitate such a computational characterization. They must therefore be at least expressive enough to characterize the subset of natural languages relevant to the application; we might call this criterion *weak completeness.* On the other hand, they are constrained by limitations of computational effectiveness, not only in the technical sense as used in the term "effective procedure", but in the informal sense of "usable under current technology" as well.

On a secondary level, of course, other characteristics are important. For instance, to aid in articulating the grammars of natural language and changing them as new consequences of a grammatical analysis are revealed, they should be as flexible and expressive as possible. To make implementation and verification of correctness easier, they should be kept simple, with a firm mathematical foundation. In short, they should obey all the rubrics of a good computer language as commonly construed in current computer science. Computer-language designers are increasingly putting forward concepts of rigorous mathematical foundations, declarativeness, simplicity, expressivity,[9] and flexibility. These criteria seem to be so well accepted in computer-language design that it seems almost ludicrous to call attention to them at all. They serve as the motivating influence, for instance, in current research efforts in programming-language design issues such as the theory of data types, functional and logic programming and explicit programming-language semantics; research in all these fields and others attests to the importance of such factors in the design of computer languages. In fact, these same criteria apply equally well, and for identical reasons, to computer languages for encoding linguistic information, namely, to grammar formalisms.

In sum, the characteristics of grammar formalisms promoted by the goals of NLP are, first of all, *weak completeness* and *computational effectiveness.* Secondarily, they are the goals of computer language design in general: *expressivity, simplicity, declarativeness, rigor,* etc.

[9] Clearly, since virtually all programming languages are equivalent in absolute expressive power, what I (and others) have in mind here is notational expressivity, in the sense introduced in Section 1.

3.3 Comparing the Goals

As is apparent from the foregoing discussion, there is a great deal of overlap between the goals for linguistic theory and the strategies for the design of NLP systems. For instance, both require formalisms that are rigorous and preferably declarative, and that allow simple and elegant formulation of grammars. The main differences are found in their view of whether expressivity should be constrained or promoted.

Linguistic-theory design is characterized by a dialectic counterposing completeness and restrictiveness. In NLP, the dialectic is between generality, on the one hand, and the constraints of computational effectiveness and practical felicity, on the other. Restrictiveness is not only not a primary goal, it is a characteristic to be avoided wherever possible. Again, consider an analogy with computer programming. If restrictiveness were a criterion in choosing a computer language for solving a problem, then someone planning to write a program to perform matrix multiplication should be attracted to a language in which only matrix multiplication programs could be written. Such an idea is patently ridiculous, for not only is such a restriction unnecessary, but the process of changing, augmenting, and maintaining programs practically guarantees that today's matrix multiplication program will be tomorrow's general statistical package. Similarly, choosing a grammar formalism because it is restrictive hobbles a grammar writer during the inevitable period of debugging, changing, rewriting and maintaining grammars. Unless one believes that linguists have designed the ultimately correct formalism and that grammar writers never make mistakes in its application, it is merely bad engineering to choose a restrictive formalism for restrictiveness' sake.

Thus, insofar as possible, those aspects of linguistic formalisms that support rigor, declarativeness, and linguistic felicity should be incorporated into NLP. However, those aspects of the formalisms that are geared towards restrictiveness should be eschewed—*unless*, of course, they provide auxiliary benefit by improving computational efficiency or simplifying analyses. Finally, aspects of the formalisms designed to aid the goal of completeness may or may not be pertinent to NLP, depending on the weakness of the particular application's completeness requirement.

Some examples of this sort of reasoning are in order. Certain particular formal constraints in the linguistic theories may be detrimental in a formalism for NLP if their inclusion decreases the expressivity of the formalism

without serving to simplify analyses. For instance, the functional locality principle of LFG falls into this class. By requiring that "designators in lexical and grammatical schemata may specify no more than two function-applications" [2, p. 278], the generality of the formalism is reduced in such a way that grammars become more, not less, complex. I am not arguing here against this formal restriction as a grammatical universal, but only against its inclusion as a device in NLP systems.

On the other hand, the head feature convention (HFC) of GPSG might be usefully added to an NLP formalism on the grounds that it makes the job of grammar writing simpler. It is no coincidence that typical implementations of LFG do not include functional locality, whereas those of GPSG do include the HFC. Ideally, however, we would want an even more general formalism—one in which the head feature convention or similar devices could be stated as a formal construct, as Martin Kay has argued for FUG.

4 How: Separating Analyses From Formalisms

Thus, we see that the goals of NLP and LT, though sharing many factors, nevertheless differ in certain key places, especially as regards the notion of restrictiveness. Since the constraints and restrictions in LT play such an important role, this brings us back to the question of how separable the analyses in the theories are from their theoretical and formal commitments. Is there a way of expressing such analyses in a formal language that is more consonant with the goals of NLP, less restrictive, simpler, more flexible? Reviewing the previous argument, the displaying of notational reductions from one formalism to another gives us a tool for deciding such questions.

In this section I will discuss several such reductions which can shed light on the question of which aspects of analyses are separable from particular formalisms and at what cost. These are the results of some research being carried out by a group at the Center for the Study of Language and Information, in which I have been participating, to find such reductions and their properties. The project is an attempt to extract and compare the main ideas in the various formalisms, theories, and analyses. To facilitate this enterprise, I will make use of certain formalisms that are not part of any linguistic theory—namely FUG and PATR. These formalisms are quite simple and expressive, and thus serve ideally as both targets and sources of reductions. Though FUG and PATR were designed from the outset to

be tools satisfying some of the goals of NLP rather than those of linguistic theory, the reductions in which they are involved do *not* show, nor are they intended to show, that the analyses can be stated in a formalism appropriate for an NLP system. Rather, they merely open up this abstract possibility by demonstrating that analyses can often, in principle at least, be separated from the formalisms in which they are traditionally stated.

It is also important to note that the research outlined here is necessarily programmatic to a great extent. Proving the correctness of a reduction of one formal system to another—Turing machines to Post machines, say—is only possible because the *meaning* of a Turing or Post machine program (i.e. a specification of its behavior) is well-defined by virtue of an explicit semantics for the machines. The grammar formalisms used in linguistics for the most part have never been given precise semantics. Indeed, of the languages discussed here, only PATR has had a mathematical semantics defined for it [11]. Therefore, we must rely on intuitive understandings of the semantics of these complex formalisms to convince ourselves of the correctness of the reductions. Since without a precise semantics, many facets of the interpretation of the formalisms are left to judgment, this program of reducing one formalism to another is prone to difficulties. Nonetheless, as the following sections should demonstrate, much useful information can be gleaned from these attempts, despite their frailties, even if the results can never be as rigorous as we might like.

The chart in Figure 3 summarizes the reductions that are at least currently proposed. I will give a brief overview of results for these reductions without presenting the reductions themselves. The claims in this section will be substantiated by a case study exemplifying the type of analysis and conclusions that this research leads to; to this end, I will present one of the reductions, that from $GPSG_F$ to PATR, in greater detail in Section 4.2. Throughout these sections, I will assume a familiarity with all of the formalisms mentioned so far.

4.1 An Overview of Some Reductions

4.1.1 PATR and FUG

I start with the reductions between PATR and FUG because these are the simplest of the reductions, yet they provide a good example of the role reductions can play in clarifying the differences among formalisms and theories.

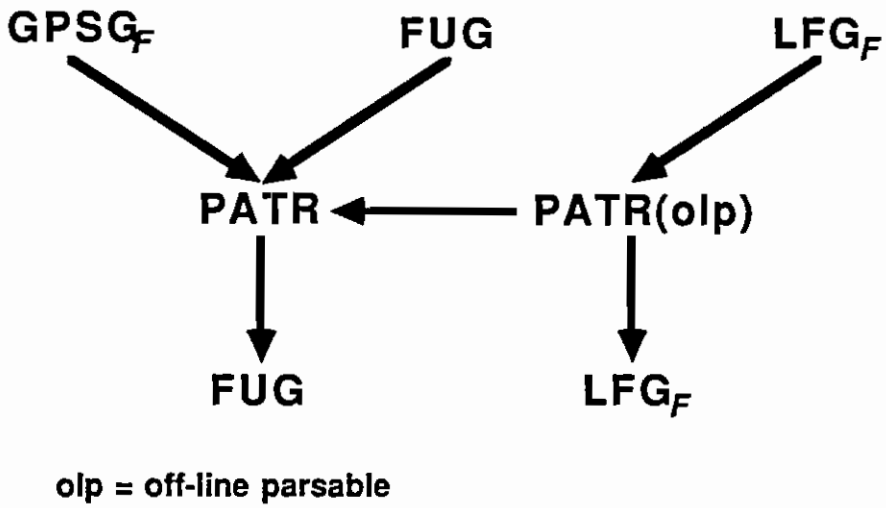

Figure 3: Proposed Reductions

PATR grammars use rules that simultaneously constrain string concatenation and directed-graph unifications. They can be viewed as FUG grammars with the following properties:

- They are in disjunctive normal form.
- No embedded *cset* features are allowed.
- No embedded *pattern* features are allowed.
- All values of the *pattern* features are of the following simple form: a sequence of *cset* elements.

This observation can be used as the basis of a simple notational reduction from PATR to FUG. The reduction thus engendered is *nonoptimal*, in the sense that it does not make full use of the FUG notation. It is this difference in expressivity that is exploited in the ability to implement (relatively) efficient parsers for PATR. The reduction is also *linear*, in the sense that the FUG grammar is related linearly in size to the PATR grammar. Finally, the reduction is itself declarative, in the sense that it can be stated independently of a procedure for building an FUG rule from a PATR rule.

In the reverse direction, any FUG can be converted to a PATR grammar by converting it to disjunctive normal form and somehow dealing with the special *ANY* values that FUG allows as existential constraints. The question of *ANY* values, along with constraint equations in general, will be touched upon again later. For the moment, I merely note that PATR is expressive enough to model them indirectly through a reduction that is not particularly perspicuous or attractive. This is thus a clue that *ANY* values constitute a true notional difference between the formalisms. The conversion to disjunctive normal form, it should be mentioned, is not a linear transformation; it is in fact exponential**refs to Ritchie, Rounds, Berwick. Thus, the difficulty in reducing FUG to PATR relative to the converse reduction lends support to the intuitive conclusion that PATR is a notational subset of FUG. Note that this conclusion holds even in the face of the formal equivalence of the two formalisms (since they are both Turing-equivalent).

4.1.2 LFG_F and PATR

The LFG_F formalism is a unification-based formalism quite similar to PATR in its overall design, though with certain extra devices and at least two additional well-defined formal constraints. It is not surprising, therefore, that most of the PATR formalism can be directly reduced to LFG. The main problems arise in overcoming the constraints of *off-line parsability*[10] [12] and *functional locality*. The former is provably impossible, in that the constraint diminishes the generative capacity (i.e. absolute expressivity) of LFG to a level below that of PATR. The use of extra features, however, allows PATR grammars to be reduced in such a way that functional locality is observed. Thus, off-line-parsable PATR grammars can be translated into LFG quite simply.

The reverse reduction, i.e., reducing LFG grammars to PATR, is more difficult because such extensions as set values, disjunction, negation, semantic forms, functional completeness, and constraint equations need to be modeled. I will briefly discuss only the latter three devices here as disjunction has been mentioned previously and the semantics of negation and set values is sufficiently complex and indeterminate that much further work is required. Semantic forms are easily modeled with the standard PATR logical-form

[10]The constraint of off-line parsability disallows LFG_F grammars whose context-free base admits derivations of the form $A \Rightarrow^+ A$ for some nonterminal A.

encoding. Constraint equations have been shown by Kelly Roach to be reducible to *ANY* values. Functional completeness can be modeled with *ANY* as well. Perhaps more surprisingly, Mark Johnson has demonstrated that both *ANY* values and constraint equations in general can be modeled in a purely monotonic system such as PATR. The reduction, as mentioned before, is reasonably complex and obscure, involving modeling a "memory" within the feature structures using unification, and replacing certain constraints with "assignments" to the memory. [11]

Similar arguments show that many aspects of LFG are directly embeddable within the PATR formalism. But the fact that the encoding of constraint equations and *ANY* is difficult, even though both are easily interreducible, points to another important notion in these grammar formalisms that is missing in PATR—the notion of nonmonotonic modes of combining information. Once again, the technique of reducing grammar formalisms has yielded an observation about what is notionally important in the formalisms.

4.2 A Case Study: $GPSG_F$ and PATR

As a case study of the detailed analysis of a formalism through the development of notational reductions, I will consider in greater detail a reduction from the $GPSG_F$ formalism to PATR.[12] I demonstrate that reductions can provide not only a more abstract understanding of the underlying ideas in the formalism, but can also lead to a simplification of the formalism's semantics. Readers uninterested in the technical details of the reduction should proceed to Section 4.2.4.

4.2.1 Overview

Like the other linguistic theories discussed here, the theory of generalized phrase structure grammar has described language axiomatically, that is, as a set of universal and language-specific constraints on the well-formedness of linguistic elements of some sort. In the case of GPSG, these elements are trees whose nodes are themselves structured entities from a domain of

[11] These results are at present unpublished. They have been presented at meetings of the Foundations of Grammar group at CSLI.

[12] This section is derived from an earlier paper on simplifying GPSG through compilation into PATR [15].

categories (a type of *feature structure* [14]). The proposed axioms have become quite complex, culminating in the ambitious recent volume by Gazdar, Klein, Pullum, and Sag entitled *Generalized Phrase Structure Grammar* [8] (henceforth GKPS). The coverage and detailed analysis of English grammar in this work are impressive, in part because of the complexity of the axiomatic system developed by the authors.

I examine the possibility that simpler descriptions of the same theory can be achieved through a slightly different, albeit still axiomatic, method. Rather than characterize the well-formed trees axiomatically, we progress in two stages by procedurally characterizing the well-formedness axioms themselves, which in turn characterize the trees; this characterization is given by a reduction procedure that converts GPSG grammars into grammars written in the PATR formalism.[13] PATR has its own declarative semantics, and can therefore be viewed as an axiomatization of string well-formedness constraints.

The characterization of GPSG thus obtained is simpler and better defined than the version described by GKPS. The semantics of the formalism is given directly through the reduction to PATR. Also, the PATR axiomatization has a clear constructive interpretation, unlike that used in GKPS, thus making the system more amenable to computational implementation. Finally, as we have mentioned previously, the characteristics of the reduction—the difficulty or ease with which the various devices can be encoded in PATR—can provide a measure of the relative expressiveness and indispensability of these devices.

4.2.2 The GPSG Axioms

A Summary of the Principles

GPSG describes natural languages in terms of various types of constraints on local sets of nodes in trees. Pertinent to the ensuing discussion are the following:

[13] However, a caveat is in order. A detailed analysis from this perspective of linear precedence (LP) rules, and feature cooccurrence restrictions is not discussed fully here, nor do I completely understand them. (See Section 4.2.3.) While in a confessional mood, I should add that the algorithm given here has not actually been implemented.

- ID (immediate dominance) rules, which state constraints of *immediate dominance* among categories;
- metarules, which state generalizations concerning classes of ID rules;
- LP (linear precedence) rules, which constrain the linear order of sibling categories;
- feature cooccurrence restrictions (FCR), which constrain the feature structures as to which are permissible categories;
- feature specification defaults (FSD), which provide values for features that are otherwise unspecified;

and, most importantly,

- universal feature instantiation principles, which constrain the allowable local sets of nodes in trees; these feature instantiation principles include the head feature convention (HFC), the foot feature principle (FFP), and the control agreement principle (CAP).

In GPSG all of these constraints are applied simultaneously. A local set of nodes in a tree is admissible under the constraints if and only if there is some base or derived ID rule (which we will call the *licensing rule*) for which the parent node's category is an extension of the left-hand-side category in the rule, and the children are respective extensions of right-hand-side categories in the rule, and, in addition, the set of nodes simultaneously satisfies all of the separate feature instantiation principles, ordering constraints, etc. By *extension*, we mean that the constituent has all the feature values of the corresponding category in the licensing rule, and possibly some additional feature values. The former type of value is called *inherited*, the latter *instantiated*.

The feature instantiation principles are typically of the following form: if a certain feature configuration holds of a local set of nodes, then some other configuration must also be present. For instance, the antecedent of the control agreement principle is stated in terms of the existence of a *controller* and *controllee*, which notions are themselves defined in terms of feature configurations. The consequent concerns identity of agreement features.

Interaction of Principles

Much care is taken in the definitions of the feature instantiation principles (and their ancillary notions such as controller, controllee, free features, privileged features, etc.) to control the complex interaction of the various constraints. For instance, the FFP admits local sets of nodes with *slash* feature values on parent and child where no such values occur in the licensing ID rule, i.e., it allows *instantiation of slash features.* But the CAP's aforementioned definition of control is sensitive to the value of the *slash* feature associated with the various constituents. A simple definition of the CAP would ignore the source of the slash value, whether inherited, instantiated by the FFP, or instantiated in some other manner. However, the appropriate definition of control needed for the CAP must ignore *instantiated* slash features, but not *inherited* ones. According to GKPS:

> We must modify the definition of control in such a way that it ignores perturbations of semantic type occasioned by the presence of instantiated *FOOT* features. [8, p. 87]

Thus, the CAP is in some sense blind to the work of the FFP. As GKPS note, this requirement makes stating the CAP a much more complex task.

The increased complexity of the principles resulting from this need for tracking the origins of feature values is evident not only in the CAP, but in the other principles as well. The head feature convention requires identity of the head features of parent and head child. The features *agr* and *slash*—features that can be inherited from an ID rule or instantiated by the CAP or FFP, respectively—are head features and therefore potentially subject to this identity condition. However, great care is taken to remove such instantiated head features from obligatory manipulation by the HFC. This is accomplished by limiting the scope of the HFC to the so-called *free* head features.

> Intuitively, the *free* feature specifications on a category [the ones the HFC is to apply to] is the set of feature specifications which can legitimately appear on extensions of that category: feature specifications which conflict with what is already part of the category, either directly, or in virtue of the FCRs, FFP, or CAP, are not *free* on that category. [8, p. 95]

That is, the FFP and CAP take precedence (intuitively viewed) over the HFC.

Finally, all three principles are seen to take precedence over feature specification defaults in the following quotation:

> In general, a feature is exempt from assuming its default specification if it has been assigned a different value in virtue of some ID rule or some principle of feature instantiation. [8, p. 100]

GKPS accomplish this by defining a class of *privileged* features and excluding such features from the requirement that they assume their default value. Of course, instantiated head features, slash features, and so forth are all considered privileged. However, a modification of these exemptions is necessary in the case of lexical defaults, i.e., default values instantiated on lexical constituents. I will not discuss here the rather idiosyncratic motivation for this distinction, but merely note that lexical-constituent defaults are to be insensitive to changes engendered by the HFC, as revealed in this excerpt:

> However, this simpler formulation is inadequate since it entails that lexical heads will always be exempt from defaults that relate to their HEAD features.... Accordingly, the final clause needs to distinguish lexical categories, which become exempt from a default only if they covary with a sister, and nonlexical categories, which become exempt from a default if they covary (in relevant respects) with *any* other category in the tree. [8, p. 103]

Thus, the interaction of these principles is controlled through complex definitions of the various classes of features they are applicable to. These definitions conspire to engender the following implicit precedence ordering on the principles;[14] principles earlier in the ordering are blind to the instantiations from later principles, which are themselves sensitive to (and exempt from applying to) features instantiated by the earlier principles.[15]

[14] Current efforts by at least certain GPSG practitioners are placing the GPSG type of analysis directly in a PATR-like formalism. This formalism, Pollard's head-driven phrase structure grammar (HPSG) variant of GPSG, uses a run-time algorithm similar to the one described in this paper. Highly suggestive is the fact that the HPSG run-time algorithm also happens to order the principles in substantially the same way.

[15] We use the symbol $\succ$ to denote one principle "taking precedence over" another.

$$\text{CAP} \succ \text{FFP} \succ \text{FSD}_{lex} \succ \text{HFC} \succ \text{FSD}_{nonlex}$$

Of course, all ID rules, both base and derived, are subject to all these principles; yet metarule application is not contingent on instantiations of the base ID rules. Conversely, LP constraints are sensitive to the full range of instantiated features. The precedence ordering can thus be extended as follows:

$$\text{META} \succ \text{CAP} \succ \text{FFP} \succ \text{FSD}_{lex} \succ \text{HFC} \succ \text{FSD}_{nonlex} \succ \text{LP}$$

The existence of such an ordering on the priority of axioms is of course not a necessary condition for the coherence of such an axiomatic theory. Undoubtedly, this inherent ordering was not apparent to the developers of the theory; its existence may even be the source of some surprise to them. Yet, the fact that such an ordering does exist and is strict leads us to a substantial simplification of the system. Instead of applying all the constraints simultaneously, we might do so sequentially, so that the precedence ordering—the blindness of earlier principles in the ordering to the effects of later ones—emerges simply because the later principles have not yet applied.

This solution harkens back to earlier versions of GPSG in which the semantics of the formalism was stated in terms of compiling the various principles and constraints into pure context-free rules. This compilation process can be combinatorially explosive, yielding vast numbers of context-free rules. Indeed, the whole point of the GPSG decomposition is to succinctly express generalizations about the possible phrasal combinations of natural languages. However, by carefully choosing a system for stating constraints on local sets of nodes—a formalism more compact in its representation than context-free grammars—we can compile out the various principles and constraints without risking this explosion in practice.[16]

The GPSG principles are stated in terms of identities of features. If we are to avoid the combinatorial problems of pure CF rules, we must have a formalism in which such equalities can be stated directly, without generating all the ground instances that satisfy the equalities. What is needed, in fact, is a unification-based grammar formalism [14]. As stated previously, I will

[16] Of course, even this compilation technique is exponential in the worst case.

use a variant of PATR [13] as the formalism into which GPSG grammars are compiled. In particular, I assume a version of PATR that has been extended by the familiar technique of decomposition into an immediate-dominance and linear-precedence component. This will allow us to ignore the LP portion of GPSG for the nonce.

PATR is ideal for two reasons. First, it is the simplest of the unification-based grammar formalisms, possessing only the apparatus that is needed for this exercise. Second, a semantics for the formalism has been provided, so that, by displaying this compilation, we implicitly provide a semantics for GPSG grammars as well.

4.2.3 The Compilation Algorithm

I postpone for the time being discussion of the metarules, LP constraints, and feature cooccurrence restrictions, concentrating instead on the central principles of GPSG, those relating to feature instantiation. The following nondeterministic algorithm generates well-formed PATR rules from GPSG ID rules. A GPSG grammar is compiled into the set of PATR rules generated by this algorithm.

Preliminaries

First, observe that a GPSG ID rule is only notationally distinct from an *unordered* PATR rule. Thus, the first step in the algorithm is trivial. For example, the ID rule

$$S \rightarrow X^2, H[-subj] \qquad (R_1)$$

is written in unordered PATR as

$$\begin{array}{l} X_0 \rightarrow X_1,\ X_2 \\ \quad \langle X_0\ n \rangle\ = - \\ \quad \langle X_0\ v \rangle\ = + \\ \quad \langle X_0\ bar \rangle\ = 2 \\ \quad \langle X_0\ subj \rangle\ = + \\ \quad \langle X_1\ bar \rangle\ = 2 \\ \quad \langle X_2\ subj \rangle\ = - \end{array} \qquad (R_2)$$

Note that abbreviations (like S for $[-n, +v, bar2, +subj]$) have been made explicit.

In fact, we will make one change in the structure of categories (to simplify our restatement of the HFC) by placing all head features under the single feature *head* in the corresponding PATR rule. We do not, however, add an analogous feature *foot*.[17] Thus, the preceding rule becomes

$$
\begin{array}{l}
X_0 \rightarrow X_1,\ X_2 \\
\quad \langle X_0\ head\ n \rangle\ = - \\
\quad \langle X_0\ head\ v \rangle\ = + \\
\quad \langle X_0\ head\ bar \rangle\ = 2 \\
\quad \langle X_0\ head\ subj \rangle\ = + \\
\quad \langle X_1\ head\ bar \rangle\ = 2 \\
\quad \langle X_2\ head\ subj \rangle\ = -
\end{array}
\qquad (R_3)
$$

We use an operation add_c (read "add conservatively"), which adds an equation to a PATR rule conservatively, in the sense that the equation is added only if the equations are not thereby rendered unsolvable. If addition would cause unsolvability, a weaker *set* of unifications is added (conservatively) instead—one for each feature in the domain of the value being equated. For instance, suppose that the operation $add_c(\langle X_0\ head \rangle = \langle X_1\ head \rangle)$ is called for, where the domain of the head feature values (i.e., the various head features) consists of the features a, b, and c. If the equations in the rule already specify that $\langle X_0\ head\ a \rangle \neq \langle X_1\ head\ a \rangle$, then this operation would add only the two equations $\langle X_0\ head\ b \rangle = \langle X_1\ head\ b \rangle$ and $\langle X_0\ head\ c \rangle = \langle X_1\ head\ c \rangle$, since addition of the given equation itself would cause rule failure. Thus, the earlier constraint of values for the a feature is given precedence over the constraint to be added.

In the description of the algorithm, a nonempty path p is said to be defined for a feature structure X if and only if p is a unit path $\langle f \rangle$ and $f \in dom(X)$ or $p = \langle f\ p' \rangle$ and p' is defined for $X(f)$. Our notion of a feature's being defined for a constituent corresponds to the GPSG concepts of being instantiated *or* of covarying with some other feature.

As in the previous definition, we will be quite lax with respect to our notation for paths, using $\langle \langle a\ b \rangle\ c \rangle$ and $\langle a\ \langle b\ c \rangle\ \rangle$ as synonymous with $\langle a\ b\ c \rangle$. Also, we will consistently blur the distinction between a set of equations and the feature structure it determines.[18]

[17] But recall that *slash* is a head feature and thus would fall under the path $\langle head\ slash \rangle$.

[18] See my paper [16] for details of the mapping that makes this possible.

The Algorithm Itself

Now our algorithm for compiling a GPSG grammar into a PATR grammar follows:

For each ID rule of GPSG (basic or derived by metarule) $X_0 \rightarrow X_1, \ldots, X_n$:

CAP If X_i controls X_j (determined by $Type(X_i)$ and $Type(X_j)$), then $add_c(\langle X_i\ con\rangle = \langle X_j\ con\rangle)$, where

$$con = \begin{cases} \langle head\ slash\rangle & \text{if } \langle head\ slash\rangle \text{ is defined for } X_i \\ \langle head\ agr\rangle & \text{otherwise} \end{cases}$$

FFP For each foot feature path p (e.g., $\langle head\ slash\rangle$), if p is not defined for X_0 , then $add_c(\langle X_i\ p\rangle = \langle X_0\ p\rangle)$ for zero or more i such that $0 < i \leq n$ and such that p is not defined for X_i.[19]

FSD$_{lex}$ For all paths p with a default value, say, d, and for all i such that $0 < i \leq n$, if $\langle X_i\ bar\rangle = 0$ and p is not defined for X_i, then $add_c(\langle X_i\ f\rangle = d)$.

HFC For X_i the head of X_0 and for each head feature f, $add_c(\langle X_i\ f\rangle = \langle X_0\ f\rangle)$.

FSD$_{nonlex}$ For all paths p with a default value, say, d, and for all i such that $0 < i \leq n$, if $\langle X_i\ bar\rangle \neq 0$ and p is not defined for X_i, then $add_c(\langle X_i\ f\rangle = d)$.

---

[19] Actually, the operation add_c is superfluous here. The equation can simply be added directly, since we have already guaranteed that the pertinent features are not yet instantiated. By a similar argument, we can conclude that only the add_c operations in the CAP and HFC are really necessary. We will use add_c, however, for uniformity. Note also that it is the FFP portion of the algorithm that is responsible for its nondeterminism.

An Example

Let us apply this algorithm to the preceding rule R_1.[20] We start with the PATR equivalent R_3. By checking the existing control relationships in this rule *as currently instantiated*, we conclude that the subject X_1 controls the head X_2. We conservatively add the unification $\langle X_2\ head\ agr\rangle = \langle X_1\rangle$. This can be safely added, and therefore is.

Next, the FFP step in the algorithm can instantiate the rule further. Suppose we choose to instantiate a slash feature on X_2. Then we add the equation $\langle X_0\ head\ slash\rangle = \langle X_2\ head\ slash\rangle$. Lexical default values require no new equations, since no constituents in the rule are given as bar level 0 at this point.

The HFC conservatively adds the equation $\langle X_0\ head\rangle = \langle X_2\ head\rangle$, as X_2 is the head of X_0. But this equation, as it stands, would lead to the unsolvability of the entire set of equations, since we already have conflicting values for the head feature *subj*. Thus, the following set of unifications is added instead:[21]

$$\begin{aligned}
&\langle X_0\ head\ n\rangle &&= \langle X_2\ head\ n\rangle\\
&\langle X_0\ head\ v\rangle &&= \langle X_2\ head\ v\rangle\\
&\langle X_0\ head\ bar\rangle &&= \langle X_2\ head\ bar\rangle\\
&\langle X_0\ head\ agr\rangle &&= \langle X_2\ head\ agr\rangle\\
&\langle X_0\ head\ inv\rangle &&= \langle X_2\ head\ inv\rangle\\
&\ldots
\end{aligned}$$

Finally, nonlexical defaults are introduced for features not in the domains of constituents.[22] Since the path $\langle head\ inv\rangle$ is defined for the constituents X_0 and X_2,[23] the default value (i.e., '$-$' according to FSD 1 in GKPS) is not instantiated on either constituent. Similarly, the *case* default value (*acc*,

[20] We do not include here the effect of the rule on every feature postulated by GKPS but only a representative sample.

[21] A more efficient representation of such sets could be achieved by the introduction of nonmonotonic operations such as overwriting or priority union. But such notational efficiency considerations need not concern us here.

[22] I have made the simplifying assumption that feature specification defaults are stated in terms of simple default values for features, rather than the more complex boolean conditions used in the GKPS text. The modifications to allow the more complex FSDs are probably complex leading to an exponential nondeterminism in this phase of compilation.

[23] The value of the feature *head* on the constituent X_0 has the feature *inv* in its domain because the unification $\langle X_0\ head\ inv\rangle = \langle X_2\ head\ inv\rangle$ gives as value to $\langle X_0\ head\ inv\rangle$ a variable, the same variable as the value for $\langle X_2\ head\ inv\rangle$. Thus, the path $\langle head\ inv\rangle$ is defined for X_0 and, similarly, for X_2.

FSD 10) is not instantiated on the subject NP. But the *conj* feature default[24] ('~') will be instantiated on all three constituents with the equations

$$\begin{aligned}\langle X_0\ conj\rangle &= \tilde{}\\ \langle X_1\ conj\rangle &= \tilde{}\\ \langle X_2\ conj\rangle &= \tilde{}\end{aligned}$$

The (partial) generated rule is the following:

$$\begin{aligned}&X_0 \rightarrow X_1,\ X_2\\ &\quad\langle X_0\ head\ n\rangle = -\\ &\quad\langle X_0\ head\ v\rangle = +\\ &\quad\langle X_0\ head\ bar\rangle = 2\\ &\quad\langle X_0\ head\ subj\rangle = +\\ &\quad\langle X_1\ head\ bar\rangle = 2\\ &\quad\langle X_2\ head\ subj\rangle = -\\ &\quad\langle X_2\ head\ agr\rangle = \langle X_1\rangle\\ &\quad\langle X_0\ head\ slash\rangle = \langle X_2\ head\ slash\rangle\\ &\quad\langle X_0\ head\ n\rangle = \langle X_2\ head\ n\rangle \qquad (R_1)\\ &\quad\langle X_0\ head\ v\rangle = \langle X_2\ head\ v\rangle\\ &\quad\langle X_0\ head\ bar\rangle = \langle X_2\ head\ bar\rangle\\ &\quad\langle X_0\ head\ agr\rangle = \langle X_2\ head\ agr\rangle\\ &\quad\langle X_0\ head\ inv\rangle = \langle X_2\ head\ inv\rangle\\ &\quad\ldots\\ &\quad\langle X_0\ conj\rangle = \tilde{}\\ &\quad\langle X_1\ conj\rangle = \tilde{}\\ &\quad\langle X_2\ conj\rangle = \tilde{}\\ &\quad\ldots\end{aligned}$$

Problems and Extensions

Several problems have been glossed over in the previous discussion. First, I have not mentioned the role of LP rules. Two possibilities are available for their interpretation: a "run-time" and a "compile-time" interpretation. We can augment the PATR formalism with LP rules in the same way as GKPS, providing for local sets of nodes to satisfy an ID PATR rule if and only if the nodes are extensions of elements in the ID rule such that the LP rules

[24] I assume here, contra GKPS, that '~' is a full-fledged value in its own right, at least as interpreted in this compilation. Since this value fails to unify with any other, e.g., '+' or '−', it has exactly the behavior desired, namely, that the feature is prohibited from taking any of its standard values.

are all satisfied. Alternatively, we can generate at compile time all possible rule linearizations compatible with the LP statements, but this leads us into the problem of interpreting LP statements relative to partially instantiated categories, an issue beyond the scope of this paper.

Second, feature cooccurrence restrictions were ignored in the previous discussion. One alternative is to modify the lattice of categories relative to which unification is defined[25] in such a way that all categories violating the FCRs are simply removed. Then unification over this revised lattice would be used instead of the simpler version and FCRs will automatically always be obeyed. Unfortunately, the possibility exists that unification over the revised lattice may not bear the same order-independence properties that characterize unification over the freely generated lattice. Of course, if this turned out to be the case, it would cast doubt on the well-foundedness of the original GKPS interpretation of FCRs as well, and thus is an interesting question to pursue.

Another alternative involves checking the FCRs at every point in the algorithm, throwing out any rules which violate them at any point. In addition, FCRs would be required to be checked during run time as well. This alternative, though more direct, violates the spirit of the enterprise of giving a compilation from the complex GKPS formulation to a simpler system.

A final problem concerns the ordering of the HFC and the CAP. The definitions of controller and controllee necessary for stating the CAP depend on the assignment of semantic types to constituents, which in turn depend on the configuration of features in the categories. We have already noted that the features pertinent to the definition of semantic type (and hence control) do not include instantiated foot features. Indeed, GKPS claim that "it is just HEAD feature specifications (other than those which are also FOOT feature specifications) and inherited FOOT feature specifications that determine the semantic types relevant to the definition of control" [8, p. 87]. Unfortunately, the ordering we have given precludes instantiated head features from participating in the definition of semantic type and hence in the CAP.[26] It seems that the HFC must apply before the CAP for the definition of semantic type, but after the CAP so that the CAP instantiations of head

[25] For the technical background of such a move, see the discussion by Pereira and me [11].

[26] I am indebted to Roger Evans and William Keller for pointing this problem out to me and for helpful discussion of alternative solutions.

features take precedence. Thus, our earlier claim of strict ordering may be falsified by this case.

However, the class of head features necessary for type determination and the class that must be instantiated after the FFP are disjoint. This can be seen from the parenthetical comment in the previous quotation. Consequently, we can merely split the application of the HFC in two, instantiating the former class before the CAP and the latter class after the FFP, as originally described. Alternatively, it might be possible merely to notate head features on the head constituent rather than on the parent as is conventionally done. In this case, the information needed by the CAP consists of inherited, not instantiated, head feature values, and is thus not subject to the ordering problem.

On the other hand, if the sets had not been disjoint, this would have presented a problem for our algorithmic analysis, though not necessarily for the definition of GPSG given by GKPS. Suppose that the HFC determines types in such a way that the CAP is required to apply and it instantiates head features thereby overriding the original values (since the CAP takes precedence) and changing the type determination so that the CAP does not apply. We would thus require the CAP to apply if and only if it does not apply. This paradox would appear as an ordering cycle in our algorithm.

4.2.4 Conclusion

The axiomatic formulation of generalized phrase structure grammar by GKPS is a quite subtle and complex system. Yet, as we have shown, GPSG grammars can be substantially converted to grammars in a simpler, and constructive, axiomatic system through a straightforward (albeit procedural) mapping. Intrinsic in this conversion is the use of a unification-based grammar formalism, so that axioms can be stated schematically, without enumerating all of their possible instantiations. In fact, we would contend that defining the semantics of a GPSG grammar in this way yields a much simpler formulation. The need for such a reconstruction is evident to anyone who has studied the GKPS text.

Furthermore, the characteristics of the reduction—for instance, the location of nondeterminism in the compilation algorithm and the complexity of that nondeterminism—provides a measure of the relative notational expressivity of the formalisms. The exponentiality of metarule compilation

and the pervasive nature of FCRs, for example, indicate that exactly those portions of $GPSG_F$ are truly essential for stating certain analyses, i.e., that analyses using those formal devices do so necessarily.

5 Summary

These arguments are not proofs of the equivalence of formalisms. Rather, they are informal demonstrations that some of the formalisms commonly employed in linguistics today share key concepts. To the extent that this is so, analyses from one formalism can be stated within another. For instance, the type of subcategorization analysis applied in many PATR grammars and used exclusively by HPSG, involving lexical encoding of complements in terms of lists of complements rather than sets of grammatical functions, could be embedded within the LFG_F formalism or FUG. Similarly, LFG-style analyses based on the lexical encoding of subcategorization with grammatical functions are easily embeddable in FUG or in PATR augmented with some nonmonotonic device (but less easily without one); furthermore, we learn that the choice of nonmonotonic device is not particularly critical.

Once this view of the mutual independence of analysis and formalism takes hold, we can start looking for the key ideas that are shared across analyses, across formalisms, and across theories, ideas which allow for simple reduction of one formalism to another. Such shared ideas include the structuring of expressions and association of expressions with partial information, constraints of equality over information structures, lexical encoding of phrasal combinatorics, and so forth. Conversely, those devices that engender complex reductions can be pinpointed as important notional extensions, and can be investigated in the abstract, independent of their instantiation in a particular formalism. As we have seen, notions such as disjunction, various nonmonotonic types of constraints, and complex constraints such as feature cooccurrence restrictions are examples of devices that engender intrinsic differences among the formalisms.

These reductions demonstrate that analyses from many current linguistic theories all share basic properties. This commonality shows a remarkable convergence of syntactic theories, echoed in the similarity of their associated formalisms. Certainly, NLP systems should take advantage of this convergence, especially as they are thereby enabled to use analyses from any and all of the theories.

6 Preventing Some Misconceptions

In an attempt to forestall certain inevitable questions concerning the points made in this paper, I would like to mention certain misconceptions to be avoided.

First of all, linguists might complain that I am ascribing too much significance to the notion of formalism in linguistic theory. On the contrary, by precisely delimiting the role formalisms play in furthering the goals of linguistic theory (for instance, by means of notational reductions) I believe that this work has demonstrated that differences among the various formalisms are considerably less than is commonly thought. In this sense, the research seems to put formalisms in their proper place, as particular notations for very general ideas that transcend the formalisms themselves.

Second, I would like to make it clear that I am not recommending PATR (or FUG for that matter) as a formalism in which to write LFG grammars or GPSG grammars. PATR is an egregious notation for LFG compared to LFG_F. In fact, the pure PATR formalism was never intended to serve as a formalism in which a grammar writer actually composes any kind of grammar at all.

> Clearly, the bare PATR-II formalism ... is sorely inadequate for any major attempt at building natural-language grammars.... However, given a simple underlying formalism, we can build more efficient, specialized languages on top of it, much as MACLISP might be built on top of pure LISP. And just as MACLISP need not be implemented (and is not implemented) directly in pure LISP, specialized formalisms built conceptually on top of pure PATR-II need not be so implemented. [13, p. 364]

Thus PATR is playing the same role here as kernel languages in computer science. Cardelli and Wegner have summarized the idea nicely:

> Although we have used a unified language throughout the paper, we have not presented a language design. In language design there are many important issues to be solved concerning readability, ease of use, etc. which we have not directly attacked. We propose [the unified language] as a framework to classify and

> compare existing languages. We do not propose it as a programming language, as it may be clumsy in many areas, but it could be the basis of one. [3, p. 41]

The claim here is that very general languages like PATR or FUG can serve as the bases of particular computer languages for encoding linguistic information—languages that can employ analyses from different syntactic theories. Furthermore, the conflicting goals of linguistic theory and natural-language processing make such an enterprise eminently desirable. Finally, research in this area can help to elucidate theoretical questions from linguistics concerning the true nature of constraints in theories, to determine whether they inhere in the formalism or require substantive restrictions.

References

[1] Robert Berwick and Amy Weinberg. *The Grammatical Basis of Linguistic Performance: Language Use and Acquisition.* MIT Press, Cambridge, Massachusetts, 1984.

[2] Joan Bresnan, editor. *The Mental Representation of Grammatical Relations.* MIT Press, Cambridge, Massachusetts, 1982.

[3] Luca Cardelli and Peter Wegner. Understanding types, data abstraction and polymorphism. Draft paper, 1985.

[4] Noam Chomsky. *Aspects of the Theory of Syntax.* MIT Press, Cambridge, Massachusetts, 1965.

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

[6] Steven Fortune, Daniel Leivant, and Michael O'Donnell. *The Expressiveness of Simple and Second Order Type Structures.* Research Report RC 8542, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1980.

[7] Gerald Gazdar. *Phrase Structure Grammar*, pages 131–186. D. Reidel, Dordrecht, Holland, 1982.

[8] Gerald Gazdar, Ewan Klein, Geoffrey K. Pullum, and Ivan A. Sag. *Generalized Phrase Structure Grammar.* Blackwell Publishing, Oxford, England, and Harvard University Press, Cambridge, Massachusetts, 1985.

[9] Ronald Kaplan. Personal communication, 1985.

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

[11] Fernando C. N. Pereira and Stuart 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.

[12] Fernando C. N. Pereira and David 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.

[13] Stuart M. Shieber. 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.

[14] Stuart M. Shieber. *An Introduction to Unification-Based Approaches to Grammar.* Volume 4 of *Lecture Note Series*, Center for the Study of Language and Information, Stanford, California, 1986.

[15] Stuart M. Shieber. A simple reconstruction of GPSG. In *Proceedings of the 11th International Conference on Computational Linguistics*, University of Bonn, Bonn, West Germany, 25-29 August 1986.

[16] Stuart M. Shieber. 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.

[17] Patrick H. Winston and Karen A. Prendergast. *The AI Business: Commercial Uses of Artificial Intelligence.* MIT Press, Cambridge, Massachusetts, 1984.