Computational Complexity and Lexical-Functional Grammar

Robert C. Berwick

Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

1. Introduction

An important goal of modern linguistic theory is to characterize as narrowly as possible the class of natural languages. One classical approach to this characterization has been to investigate the generative capacity of grammatical systems specifiable within particular linguistic theories. Formal results along these lines have already been obtained for certain kinds of Transformational Generative Grammars: for example, Peters and Ritchie 1973a showed that the theory of Transformational Grammar presented in Chomsky's *Aspects of the Theory of Syntax* 1965 is powerful enough to allow the specification of grammars for generating any recursively enumerable language, while Rounds 1973,1975 extended this work by demonstrating that moderately restricted Transformational Grammars (TGs) can generate languages whose recognition time is provably exponential.[1]

These moderately restricted theories of Transformational Grammar generate languages whose recognition is widely considered to be computationally intractable. Whether this "worst case" complexity analysis has any real import for actual linguistic study has been the subject of some debate (for discussion, see Chomsky 1980, Berwick and Weinberg 1982). Results on generative capacity provide only a worst-case bound on the computational resources required to recognize the sentences specified by a linguistic theory.[2] But a sentence processor might not have to explicitly reconstruct deep structures in an exact (but inverse) mimicry of a transformation derivation, or even recognize every sentence generable by a particular transformational theory. For example, as suggested by Fodor, Bever and Garrett 1974, the human sentence processor could simply obey a set of heuristic principles and recover the right representations specified by a linguistic theory, but not according to the rules of that theory. To say this much is to simply restate a long-standing view that a theory of linguistic performance could well differ from a theory of linguistic competence – and that the relation between the two could vary from one of near isomorphism to the much weaker input/output equivalence implied by the Fodor, Bever, and Garrett position.[3]

In short, the study of generative capacity furnishes a mathematical characterization of the computational complexity of a linguistic system. Whether this mathematical characterization is cognitively relevant is a related, but distinct, question. Still, the determination of the computational complexity of a linguistic system is an important undertaking. For one thing, it gives a precise description of the class of languages that the

[1] In Rounds's proof, transformations are subject to a "terminal length non-decreasing" condition, as suggested by Peters and Myhill (cited in Rounds 1975). A similar "terminal length increasing" constraint (to the author's knowledge first proposed by Petrick 1965) when coupled with a condition on recoverability of deletions, yields languages that are recursive but not necessary recognizable in exponential time.

[2] Usually, the recognition procedures presented actually recover the structural description of sentences in the process of recognition, so that in fact they actually parse sentences, rather than simply recognize them.

[3] The phrase "input/output equivalence" simply means that the two systems – the linguistic grammar and the heuristic principles – produce the same (surface string, underlying structure) pairs. Note that the "internal constitution" of the two systems could be wildly different. The intuitive notion of "embedding a linguistic theory into a model of language use" as it is generally construed is much stronger than this, since it implies that the parsing system follows some (perhaps all) of the same operating principles as the linguistic system, and makes reference in its operation to the same system of rules. This intuitive description can be sharpened considerably. See Berwick and Weinberg 1983 for a more detailed discussion of "transparency" as it relates to the embeddability of a linguistic theory in a model of language use, in this case, a model of parsing.

0362-613X/82/030097-13$03.00

system can generate, and so can tell us whether the linguistic system is in principle descriptively adequate. This method of argument was used in Chomsky's original rejection of finite-state languages as an adequate characterization of human linguistic competence. Second, as mentioned, the resource bound on recognition given by a complexity-theoretic analysis tells us how long recognition will take in the worst possible case.

Since unrestricted TGs can generate computationally "hard" languages, then plainly, in order to make TGs efficiently parsable, one must supply additional restrictions. These could be either modifications to the theory of TG itself, or constraints on the parsing mechanism. For example, the current theory of TG (see Chomsky 1981) contains several restrictions on the way in which displaced constituents such as wh-phrases may be linked to their "canonical" position in predicate-argument structure. (E.g., Who in Who did Bill kiss is assumed to be linked to a canonical argument position after the verb kiss.) As an example of a constraint on the parsing mechanism, one could proceed as did Marcus 1980, and posit constraints dictating that TG-generated languages must have parsers that meet certain "locality conditions".[4] For instance, the Marcus constraints amount to an extension of Knuth's 1965 LR(k) locality condition to a (restricted) version of a two-stack deterministic pushdown automaton.[5]

Recently, a new theory of grammar has been advanced with the explicitly stated aim of meeting the dual demands of learnability and parsability – the Lexical-Functional Grammars (LFGs) of Kaplan and Bresnan 1981. The theory of Lexical-Functional Grammar is claimed to be at least as descriptively adequate as Transformational Grammar, if not more so. Moreover, it is claimed to have none of TG's computational unruliness, in the sense that it is claimed that there is a "natural" embedding of an LFG into a parsing mechanism (a performance model) that accounts for human sentence processing behavior. In LFG, there are no transformations (as classically described); the work formerly ascribed to transformations such as "passive" is shouldered by information stored in lexical entries associated with lexical items. The underlying representation of surface strings that is built is also different from the deep structures of classical transformational theory; the representation makes reference to *functionally* defined notions of grammatical terms like "Subject", rather than defining them structurally, as was done in classical transformation theory. The elimination of transformational power and the use of a different kind of underlying representation for sentences naturally gives rise to the hope that a lexical-functional system would be computationally simpler than a transformational one.

An interesting question then is to determine, as has already been done for the case of certain brands of Transformational Grammar, just what the "worst case" computational complexity for the recognition of LFG languages is. If the recognition time complexity for languages generated by the basic LFG theory can be as complex as that for languages generated by a moderately restricted transformational system, then presumably LFG will also have to add additional constraints, beyond those provided in its basic theory, in order to ensure efficient parsability. Just as with transformational theories, these could be constraints on either the theory or its performance model realization.

The main result of this paper is to show that certain Lexical-Functional Grammars can generate languages whose recognition time *is* very likely computationally intractable, at least according to our current understanding of algorithmic complexity. Briefly, the demonstration proceeds by showing how a problem that is widely conjectured to be computationally difficult – namely, whether there exists an assignment of 1's and 0's (or "T"'s and "F"'s) to the atoms of a Boolean formula in conjunctive normal form that makes the formula evaluate to "1" (or "true") – can be re-expressed as the problem of recognizing whether a particular string is or is not a member of the language generated by a certain Lexical-Functional Grammar. This "reduction" shows that in the worst case the recognition of LFG languages can be just as hard as the original Boolean satisfiability problem. Since it is widely conjectured that there cannot be a polynomial-time algorithm for satisfiability (the problem is NP-complete), there cannot be a polynomial-time recognition algorithm for LFGs *in general* either. Note that this results sharpens that in Kaplan and Bresnan 1981; there it is shown only that LFGs

[4] It is important not to confuse the requirement that TG-generated languages have parsers that meet certain constraints with the claim that such parsers transparently embed TGs. As stated, the only requirement is one of weak input/output equivalence – i.e., that the parser construct the same (surface string, underlying representation) pairs as the TG. Actually, one *can* show that a modified Marcus parsing system goes beyond this requirement and operates according to the same principles as the recent transformational theory of Chomsky. That is, such a modified Marcus parser makes reference to the same base constraints and representational units as the linguistic theory. Since it abides by the same rules and representations as TG, one is justified in claiming that the model embeds a TG. Note that the Marcus parser does *not* mimic earlier theories of TG (as presented in *Aspects of the Theory of Syntax)*; there is no rule-for-rule correspondence between an *Aspects* grammar and the rules of the Marcus parser. But neither is there a rule-for-rule correspondence between modern theories of TG and the *Aspects* theory. For example, there is no longer a distinct rule of "passive" or "dative movement". A detailed demonstration of this claim would go far beyond the purpose of this paper. See Berwick and Weinberg forthcoming.

[5] The possible need for LR(k)-like restrictions in order to ensure efficient processability was also suggested by Rounds 1973.

(weakly) generate some subset of the class of context-sensitive languages, and, therefore, in the worst case, exponential time is known to be *sufficient* (though not necessary) to recognize any LFG language. The result in Kaplan and Bresnan 1981 therefore does not address the question of how much time, in the worst case, is *necessary* to recognize LFG languages.[6] The result of this paper indicates that in the worst case more than polynomial time will *probably* be necessary. (The reason for the hedge "probably" will become apparent below; it hinges upon the central unsolved conjecture of current complexity theory.) In short then, this result places the LFG languages more precisely in the complexity hierarchy of languages.

It also turns out to be instructive to inquire into just *why* a lexical-functional approach can turn out to be computationally difficult, and *how* computational tractability may be guaranteed. Advocates of lexical-functional theories may have thought (and some have explicitly stated) that the banishment of transformations is a computationally wise move because transformations are computationally costly. Eliminate the transformations, so this causal argument goes, and one has eliminated all computational problems. Intriguingly though, when one examines the proof to be given below, the ability to express co-occurrence constraints over arbitrary distances across terminal tokens in a string (as in Subject-Verb number agreement), when coupled with the possibility of alternative lexical entries, seems to be all that is required to make the recognition of LFG languages intractable.

This leaves the question posed in the opening paragraph: just what sorts of constraints on natural languages *are* required in order to ensure efficient parsability? As it turns out, even though *general* LFGs may well be computationally intractable, it is easy to imagine a variety of additional constraints for LFG theory that provide a way to avoid this problem. All of these additional restrictions amount to making the LFG theory more restricted, in such a way that the reduction argument cannot be made to work. For example, one effective restriction is to stipulate that there can only be a finite stock of features with which to label lexical items. In any case, the moral of the story is an unsurprising one: specificity and constraints can absolve a theory of grammar from computational intractability. What may be more surprising is that the requisite locality constraints seem to be useful for a variety of theories of grammar, from Transformational Grammar to Lexical-Functional Grammar.

[6] This result can also be established by showing that LFGs can generate at least all the indexed languages as defined by Aho 1968. See Berwick 1981 for details.

2. A Review of Reduction Arguments

The demonstration of the computational complexity of LFGs relies upon the standard complexity-theoretic technique of reduction. Because this method may be unfamiliar to many readers, a short review is presented immediately below; this is followed by a sketch of the reduction proper.

The idea behind the reduction technique is to take a difficult problem, in this case the problem of determining the satisfiability of Boolean formulas in conjunctive normal form (CNF), and show that the problem can be quickly transformed into the problem whose complexity remains to be determined, in this case the problem of deciding whether a given string is in the language generated by a given Lexical-Functional Grammar. Before the reduction proper is reviewed, some definitional groundwork must be presented. A Boolean formula in conjunctive normal form is a conjunction or disjunction of literals, where a literal is just an atom (like X_i) or the negation of an atom ($\overline{X}_i$). A formula is *satisfiable* just in case there exists some assignment of T's and F's (or 1's and 0's) to the atoms of a formula that forces the evaluation of the entire formula to be T (true); otherwise, the formula is said to be *unsatisfiable*. For example, the following formula is satisfiable:

$$(X_2 \vee X_3 \vee X_7) \wedge (X_1 \vee \overline{X}_2 \vee X_4) \wedge (\overline{X}_3 \vee \overline{X}_1 \vee X_7)$$

since the assignment of X_2=T, X_3=F, X_7=F, X_1=T, and X_4=F makes the whole formula evaluate to "T". The reduction in the proof below uses a somewhat more restricted format where every term comprises the disjunction of exactly three literals, so-called 3-CNF (or "3-SAT").[7]

How does a reduction show that the LFG recognition problem must be at least as hard (computationally speaking) as the original problem of Boolean satisfiability? The answer is that any decision procedure for LFG recognition could be used as a correspondingly fast decision procedure for 3-CNF, as follows:

(1) Given an instance of a 3-CNF problem (the question of whether *there exists* a satisfying assignment for a given formula in 3-CNF), apply the transformational algorithm provided by the reduction; this algorithm is itself assumed to execute quickly, in polynomial time or less. The algorithm outputs a corresponding LFG decision problem, namely: (i) a Lexical-Functional Grammar and (ii) a string to be tested for membership in the language generated by the LFG. The LFG recognition problem represents or mimics the decision problem for 3-CNF in the sense that the "yes" and "no" answers to both satisfiability problem and member-

[7] This restriction entails no loss of generality (see Hopcroft and Ullman 1979, Chapter 12), since this restricted format can be easily shown to have the power to express any CNF formula.

ship problem must coincide (if there is a satisfying assignment, then the corresponding LFG decision problem should give a "yes" answer, etc.).

(2) Solve the LFG decision problem – the string-LFG pair – output by Step 1. If the string is in the LFG language, the original formula is satisfiable; if not, it is unsatisfiable.[8]

To see how a reduction can tell us something about the "worst case" time or space complexity required to recognize whether a string is or is not in an LFG language, suppose for example that the decision procedure for determining whether a string is in an LFG language took only polynomial time (that is, takes time n^k on a deterministic Turing machine, for some integer k, where n is the length of the input string). Then, since the composition of two polynomial algorithms can be readily shown to take only polynomial time (see Hopcroft and Ullman 1979, Chapter 12), the entire process sketched above, from input of the CNF formula to the decision about its satisfiability, will take only polynomial time.

However, CNF (or 3-CNF) has no *known* polynomial time algorithm, and indeed, it is considered *exceedingly* unlikely that one could exist. Therefore, it is just as unlikely that LFG recognition could be done (in general) in polynomial time. What the reduction shows is that LFG recognition is *at least as hard as* the problem of CNF. Since the latter problem is widely considered to be difficult, the former inherits the difficulty.

The theory of computational complexity has a much more compact term for problems like CNF: CNF is NP-complete. This label is easily deciphered:

(1) CNF satisfiability is in the class NP. That is, the problem of determining whether an arbitrary CNF formula is satisfiable can be computed by a *non*-deterministic Turing machine in polynomial time. (Hence the abbreviation "NP", for "non-deterministic polynomial". To see that CNF is indeed in the class NP, note that one can simply *guess* all possible combinations of truth assignments to literals, and check each guess in polynomial time.)

(2) CNF is *complete*. That is, all *other* problems in the class NP can be quickly reduced to some CNF formula. (Roughly, one shows that Boolean formulas can be used to "simulate" any valid computation of a non-deterministic Turning machine.)

Since the class of problems solvable in polynomial time on a deterministic Turing machine (conventionally notated, P) is trivially contained in the class so solved by a non-deterministic Turing machine, the class P must be a subset of the class NP. A well-known, well-studied, and still open question is whether the class P is a *proper* subset of the class NP. In other words, are there problems solvable in non-deterministic polynomial time that cannot be solved in deterministic polynomial time? Because all of the several thousand NP-complete problems now catalogued have so far proved recalcitrant to deterministic polynomial time solution, it is widely held that P must indeed be a proper subset of NP, and therefore that the best possible algorithms for solving NP-complete problems must take more than polynomial time. (In general, the algorithms now known for such problems involve exponential combinatorial search, in one fashion or another; these are essentially methods that do no better than to brutally simulate – deterministically, of course – a non-deterministic machine that "guesses" possible answers.)

To repeat the force of the reduction argument then, if *all* LFG recognition problems were solvable in polynomial time, then the ability to quickly reduce CNF formulas to LFG recognition problems would imply that all NP-complete problems would be solvable in polynomial time, and that the class P = the class NP. This possibility seems extremely remote. Hence, our assumption that there is a fast (general) procedure for recognizing whether a string is or is not in the language generated by an arbitrary LFG must be false. In the terminology of complexity theory, LFG recognition must be NP-hard – "as hard as" any other NP problem, including the NP-complete problems. This means only that LFG recognition is *at least as hard* as other NP-complete problems – it could still be more difficult (lie in some class that contains the class NP). If one could also show that the languages generated by LFGs are in the class NP, then LFGs would be shown to be NP-complete. This paper stops short of proving this last claim, but simply conjectures that LFGs *are* in the class NP.

3. A Sketch of the Reduction

To carry out this demonstration in detail one must explicitly describe the transformation procedure that takes as input a formula in CNF and outputs a corresponding LFG decision problem – a string to be tested for membership in a LFG language and the LFG itself. One must also show that this can be done quickly, in a number of steps proportional to (at most) the length of the original formula to some polynomial power.

[8] Note that the grammar and string so constructed depend upon just what formula is under analysis; that is, for each different CNF formula, the procedure presented above outputs a *different* LFG grammar and string combination. In the LFG case it is important to remember that "grammar" really means "grammar plus lexicon" – as one might expect in a lexically-based theory. S. Peters has observed that a slightly different reduction allows one to keep most of the grammar fixed across all possible input formulas, constructing only different-sized lexicons for each different CNF formula. Details are provided below.

One caveat is in order before embarking on a proof sketch of this reduction. The grammar that is output by the reduction procedure will *not* look very much like a grammar for a natural language, although the grammatical devices that will be employed will in every way be those that are an essential part of the LFG theory.[9] In other words, although it is most unlikely that any *natural* language would encode the satisfiability problem (and hence be intractable) in just the manner outlined below, no "exotic" LFG machinery is used in the reduction. Indeed, some of the more powerful LFG notational formalisms – long-distance binding, existential and negative feature operators – have not been exploited. (An earlier proof made use of an existential operator in the feature machinery of LFG, but the reduction presented here does not.)

To make good this demonstration one must set out just what the satisfiability problem is and what the decision problem for membership in an LFG language is. Recall that a formula in conjunctive normal form is satisfiable just in case every conjunctive term evaluates to *true,* that is, at least *one* literal in *each* term is true. The satisfiability problem is to find an assignment of T's and F's to the atoms at the bottom (note that complements of atoms are also permitted) such that the root node at the top gets the value "T" (for *true*). How can we get a Lexical-Functional Grammar to represent this problem? What we want is for *satisfying* assignments to correspond to *well-formed* sentences of some corresponding Lexical-Functional Grammar, and *non-satisfying* assignments to correspond to sentences that are *not* well-formed, according to the LFG, as indicated in Figure 1. Since one wants the satisfying/non-satisfying assignments of any particular formula to map over into well-formed/ill-formed sentences, one must obviously exploit the LFG machinery for capturing well-formedness conditions on sentences. To make the discussion clear to the reader will require a brief account of the LFG theory itself.

$$\begin{tikzcd}
\text{satisfiable formula} \arrow[d] & \text{non-satisfiable formula} \arrow[d] \\
\text{sentence w IS in LFG language L(G)} & \text{sentence w IS NOT in LFG language L(G)}
\end{tikzcd}$$

Figure 1. A reduction preserves solutions to the original problem.

Just as in a transformational theory, a Lexical-Functional Grammar associates with each generable surface string (sentence) a number of distinct representations. For our purposes here we need to focus on just two of these: the constituent structure of a sentence (its "c-structure", roughly, a labeled bracketing of the surface string, annotated with certain feature complexes); and the functional structure of a sentence (its "f-structure", roughly, a representation of the underlying predicate-argument structure of a sentence, described in terms of grammatical relations such as Subject and Object.) Unlike a Transformational Grammar, however, a Lexical-Functional Grammar does not generate surface sentences by first specifying an explicit, context-free deep structure followed by a series of categorially-based transformations. "Categorially-based" simply means that the transformations move constituents defined in terms of categories, like NP or PP.) Rather, predicate-argument structure is mapped directly into c-structure, on the basis of predicates that are grounded upon grammatical relations (like Subject and Object). The conditions for this mapping are provided by a set of so-called functional equations associated with the context-free rules for generating permissible c-structures, along with a set of conventions that in effect convert the functional equations into well-formedness predicates for c-structures.

In more detail, an LFG c-structure is generated by a base context-free grammar. A necessary condition for a sentence (considered as a string) to be in the language generated by a Lexical-Functional Grammar is that it can be generated by this base grammar; such a sentence is then said to have a well-formed constituent structure. For example, if the base rules included S=>NP VP; VP=>V NP, then (glossing over details of Noun Phrase rules) the sentence John kissed the baby would be well-formed but John the baby kissed would not. Note that this assumes, as usual, the existence of a lexicon that provides a categorization for each terminal item, e.g., that baby is of the category N, kissed is a V, etc. Importantly then, this well-formedness condition requires us to provide at least one legitimate parse tree for the candidate sentence that shows how it may be derived from the underlying LFG base context-free grammar. (There could be more than one legitimate tree if the underlying grammar is ambiguous.) Note further that the choice of categorization for a lexical item may be crucial. If baby was assumed to be of category V, then both sentences above would be ill-formed.

Since the base grammar is context-free, there are well-known algorithms for checking the well-formedness of the strings it can generate in polynomial time. Intractability cannot arise on this score, then.

A Lexical-Functional Grammar consists of more than just a base context-free grammar, however. As mentioned, a second major component of the LFG theory is the provision for adding a set of so-called

[9] These include feature agreement, the lexical analog of Subject or Object "control", lexical ambiguity, and a garden variety context-free base grammar.

functional equations to the base context-free rules. The functional equations define an implicit f-structure associated with every c-structure, and this f-structure must itself be well-formed. Part of the linguistic role of f-structures is to account for the co-occurrence restrictions that are an obvious part of natural languages (e.g., Subject-Verb agreement).

How exactly do the functional equations work? Their job is to specify how the f-structure of a sentence gets built. This is done by associating possibly complex features with lexical entries and with the nonterminals of specified context-free rules; these features have values. The features are pasted together under the direction of the functional equations to form f-structures associated with the sub-constituents of the sentence; these (now possibly complex) f-structures are in turn assembled to form a master f-structure associated with the root node of the sentence.

Note then that in this theory a "feature" can be something as simple as an atomic object that is binary valued; for example, a Subject feature could be either plural or singular in value. But denominators can also have a range of values, and – more crucial for the purposes of the demonstration here – a feature can itself be a complex, hierarchically structured object that contains other features as sub-constituents. For example, the "feature" that eventually becomes associated with the root node of a sentence is in fact an f-structure that represents the full propositional structure of the sentence. Thus if the surface string was the sentence, The girl promised to kiss the baby, then the f-structure associated with the root node of the sentence is a complex "feature" that itself contains an embedded f-structure corresponding to the embedded proposition the girl to kiss the baby.

As mentioned, well-formedness is also determined by functional equations, dictating (according to certain conventions) how feature complexes are to be assembled. By and large the f-structure complex at a node X is assembled compositionally in terms of the f-structure complexes of the nodes below it in the constituent structure tree. For example, the root node of a sentence will have an associated f-structure with Subject and Predicate sub-features. These structures are themselves complex – the entire Subject NP and Verb-Verb Complement structures, respectively. For instance, the Subject NP in turn has a sub-feature Number; the Predicate contains complex sub-features corresponding to the Verb and Verb Complements. The basic assembly directive is the notation $(\uparrow = \downarrow)$.[10] When attached to a particular node X, it states that the f-structure of the node *above* X is to share all the f-structure of the nodes *below* X. The effect is to merge and "pass up" all the f-structure values of the nodes below X to the node above X. One can also pass along just particular subfields of the f-structure below X by specifying a subfield on the right-hand side of the expansion rule. As an example, the notion $(\uparrow = \downarrow \text{Number})$ attached to a node X states that the f-structure of the node above X is to contain at least the value of the Number feature. (This "value" may itself be an f-structure.) Similarly, a particular subfield of the f-structure above a node X may be specified by providing a subfield label on the left-hand side of the arrow notation. For example, the notation, $(\uparrow \text{Subject} = \downarrow)$ means that the Subject subfield of the f-structure built at the node above X must contain the f-structure built below X.

A basic constraint on f-structures is that the f-structure assembled at X must be uniquely determined; that is, it cannot contain a feature F_1 with conflicting values. This entails, for example, that the Subject sub-f-structure that is built at a root-S node cannot have a Number sub-field that is filled in from one place beneath with the value Singular and from another place with the value Plural. More generally, this restriction means that two or more f-structures that are "passed up" from below according to the dictates of an arrow notation at a single node above must be *unifiable* – any common sub-fields, no matter how hierarchically complex, must be mergeable without conflict.

For example, consider Subject-Verb agreement and the sentence the baby is kissing John. The lexical entry for baby (considered as a Noun) might have the Number feature, with the value singular. The lexical entry for is might assert that the number feature of the Subject above it in the parse tree *must* have the value singular, via the annotation $(\uparrow \text{Subject} = \text{singular})$ attached to the verb. Meanwhile, the feature values for Subject are automatically found by the annotation $(\uparrow \text{Subject} = \downarrow)$ associated with the Noun Phrase portion of S=>NP VP) that grabs whatever features it finds below the NP node and copies them up above to the S node. Thus the S node gets the Subject feature with whatever value it has passed from baby below – namely, the value singular; this accords with the dictates of the verb is, and all is well. In contrast, in the sentence, the boys in the band is kissing John, boys passes up the number value plural, and this clashes with the verb's constraint; as a result this sentence is judged ill-formed, as Figure 2 shows.

[10] More generally, the assembly directive is specified via the notation $(\uparrow \mathit{feat1} = \downarrow \mathit{feat2})$, where *feat1* and *feat2* are metavariables specifying a subfield of the f-structure immediately above or below the node to which the the annotation is attached. If no field is given, then the entire f-structure is assumed. For example, the notation $(\uparrow \text{Subject Number} = \downarrow)$ attached to a node X means that the Number subfield of the Subject subfield of the f-structure associated with the node above X is to be filled in with the value of the entire f-structure below X.

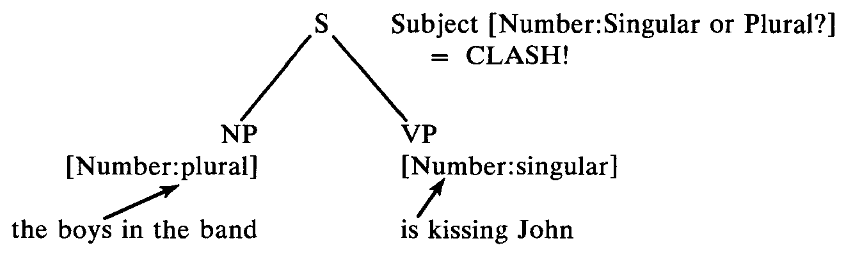

Figure 2. Co-occurrence restrictions are enforced by feature checking in a Lexical-Functional Grammar.

It is important to note that the feature compatibility check requires (1) a particular constituent structure tree (a parse tree); and (2) an assignment of terminal items (words) to lexical categories – e.g., in the first Subject-Verb agreement example above, baby was assigned to the category N, a Noun. The tree is obviously required because the feature-checking machinery propagates values according to the links specified by the derivation tree; the assignment of terminal items to categories is crucial because in most cases the values of features are derived from those listed in the lexical entry for an item (as the value of the number feature was derived from the lexical entry for the Noun form of baby). One and the same terminal item can have two distinct lexical entries, corresponding to distinct lexical categorizations; for example, baby can be both a Noun and a Verb. If we had picked baby to be a Verb, and hence had adopted whatever features are associated with the Verb entry for baby to be propagated up the tree, then the string that was previously well-formed, the baby is kissing John, would now be considered deviant. If a string is ill-formed under all possible derivation trees and assignments of features from possible lexical categorizations, than that string is not in the language generated by the LFG. The ability to have multiple derivation trees and lexical categorizations for one and the same terminal item plays a crucial role in the reduction proof: it is intended to capture the satisfiability problem of deciding whether to given an atom X_i a value of “T” or “F”.

Finally, LFG also provides a way to express the familiar patterning of grammatical relations (e.g., “Subject” and “Object”) found in natural language. For example, transitive verbs must have objects. This fact of life (expressed in an *Aspects*-style Transformational Grammar by subcategorization restrictions) is captured in LFG by specifying a so-called PRED (for predicate) feature with a Verb; the PRED can describe what grammatical relations like “Subject” and “Object” must be filled in after feature passing has taken place in order for the analysis to be well-formed. For instance, a transitive verb like kiss might have the pattern, kiss<(Subject)(Object)>, and thus demand that the Subject and Object (now considered to be “features”) have *some* value in the final analysis. The values for Subject and Object might of course be provided from some other branch of the parse tree, as provided by the feature propagation machinery; for example, the Object feature could be filled in from the Noun Phrase part of the VP expansion. See Figure 3. But if the Object were *not* filled in, then the analysis is declared *functionally incomplete,* and is ruled out. This device is used to cast out sentences such as the baby kissed.

So much for the LFG machinery that is required for the reduction proof. (There are additional capabilities in the LFG theory, such as long-distance binding, but these will not be called upon in the demonstration below.)

What then does the LFG representation of the CNF satisfiability problem look like? Basically, there are three parts to the satisfiability problem that must be mimicked by the LFG: (1) the assignment of values to atoms, e.g., X_2=>“T”; X_4=>“F”; (2) the consistency of value assignments in the formula; e.g., the atom X_2 can appear in several different terms, but one is not allowed to assign it the value “T” in one term and the value “F” in another; and (3) the preservation of CNF satisfiability, in that a string will be in the LFG language to be defined just in case its associated CNF formula is satisfiable. Let us now go over how these components may be reproduced in an LFG, one by one.

(1) Assignments: The input string to be tested for membership in the LFG will simply be the original formula, sans parentheses and the operators $\wedge$ and $\vee$; the terminal items are thus just a string of X_i's. Recall that the job of checking the string for well-formedness involves finding a derivation tree for the string, solving the ancillary co-occurrence equations (by feature propagation), and checking for functional completeness. Now, the context-free grammar constructed by the transformation procedure will be set up so as to generate a virtual copy of the associated formula, down to the point where literals X_i are assigned their value of “T” or “F”. If the original 3-CNF form had *n* terms, then denoting each by the symbol E_p, $p=1, ..., n$, this part of the grammar would look like the following:[11]

$$S => E_1\ E_2 \ldots E_n$$
$$E_p => Y_i\ Y_j\ Y_k$$

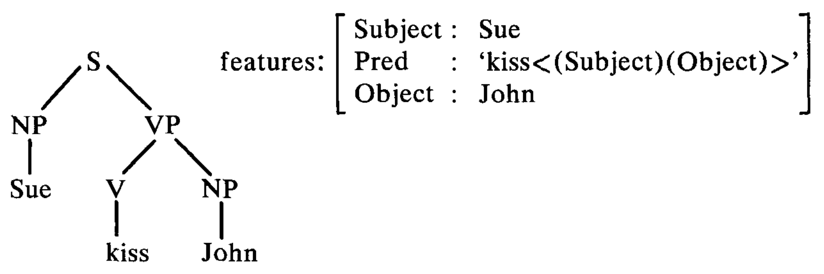

Figure 3. Predicate templates can demand that a subject or object be filled in.

(p=1,2,...,n)

The subscripts i, j, and k correspond to the actual subscripts in the original formula. Further, the Y_i are *not* terminal items, but are non-terminals that will be expanded into one of the non-terminals T_i or F_i.[12]

Note that so far there are no rules to extend the parse tree down to the level of terminal items, name the X_i. The next step does this and at the same time adds the power to choose between "T" and "F" assignments to atoms. One adds to the context-free base grammar *two* productions deriving each terminal item X_i, namely, $T_i => X_i$ and $F_i => X_i$, corresponding to an assignment of "T" or "F" to the atoms of the formula (it is important not to get confused here between the atoms of the formula – these are terminal elements in the Lexical-Functional Grammar – and the non-terminals of the grammar.) Plainly, one must also add the rules $Y_i => T_i \mid F_i$, for each i, and rules corresponding to the assignment of truth-values to the negations of literals, $T_i => \overline{X}_i$ and $F_i => \overline{X}_i$. Note that these are not "exotic" LFG rules: exactly the same sort of rule is required in the baby case, i.e., N=>baby or V=>baby, corresponding to whether baby is a Noun or a Verb. Now, the lexical entries for the "T_i" categorization of X_i will look very different from the "F_i" categorization of X_i, just as one might expect the N and V forms for baby to be different. Here is what the entries for the two categorizations of X_i look like:

X_i:	T_i	(↑Truth-assignment)	=T
		(↑Assign X_i)	=T
X_i:	F_i	(↑Assign X_i)	=F

Putting aside for the moment the "Truth-assignment" feature in this entry, the feature assignments for the negation of the literal X_i must be the complement of this entry:

$\overline{X}_i$:	T_i	(↑Truth-assignment)	=T
		(↑Assign X_i)	=F
$\overline{X}_i$:	F_i	(↑Assign X_i)	=T

The upward-directed arrows in the entries reflect the LFG feature propagation machine. Remember that T_i and F_i are just non-terminal categories, like Noun and Verb. For example, if the T_i categorization for X_i is selected, the entry says to "make the Truth-assignment feature of the node *above* T_i have the value T, and make the X_i portion of the Assign feature of the node above have the value T." This feature propagation device reproduces the assignment of T's and F's to the CNF literals. If we have a triple of such elements, and at least *one* of them is expanded out to T_i, then the feature propagation machinery of LFG will *merge* the common feature names into one large structure for the node above, reflecting the assignments made; moreover, the term will get a filled-in truth assignment value just in case at least one of the expansions selected a T_i path. This is depicted in Figure 4. Features are passed transparently through the intervening Y_i nodes via the LFG "copy" device, (↑=↓); this simply means that all the features of the node below the node to which the "copy" up-and-down arrows are attached are to be the same as those of the node above the up-and-down arrows.

It should be plain that this mechanism mimics the assignment of values to literals required by the satisfiability problem.

(2) Coordination of assignments: One must also guarantee that the X_i value assigned at one place in the tree is not contradicted by the value of an X_i or $\overline{X}_i$ elsewhere. To ensure this, we use the LFG co-

[11] The context-free base that is built depends upon the original CNF formula that is input, since the number of terms, n, varies from formula to formula. In Stanley Peters's improved version of the reduction proof [personal communication], the context-free base is fixed for all formulas with the rules:

S=>S S'

S'=>T T T or T T F or T F F or T F T or ...

(remaining twelve triples containing at least one "T")

The Peters grammar works by recursing until the right number of terms is generated (any sentences that are too long or too short cannot be matched to the input formula). Thus, the number of terms in the original CNF formula need not be explicitly encoded into the base grammar.

[12] This grammar will have to be slightly modified in order for the reduction to work, as will become apparent shortly

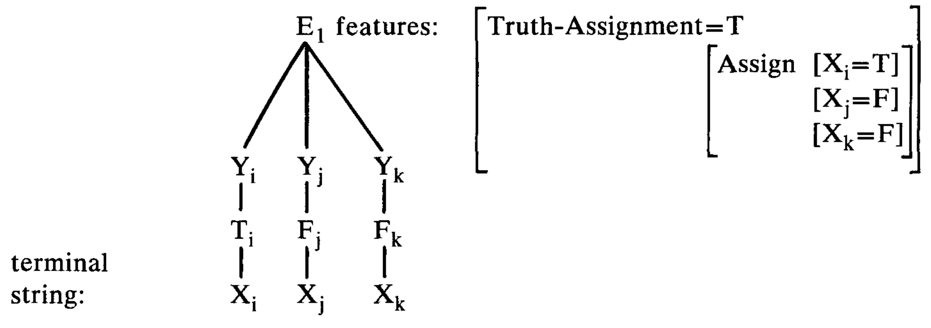

Figure 4. The LFG feature propagation machinery is used to percolate feature assignments from the lexicon.

occurrence agreement machinery: the Assign feature-bundle is passed up from each term to the highest node in the parse tree (one simply adds the ($\uparrow=\downarrow$) notation to each E_i rule in order to indicate this). The Assign feature at this node will contain the *union* of all assign feature bundles passed up by all terms. If any X_i values conflict, then the resulting structure is judged ill-formed. Thus, only *compatible* X_i assignments are well-formed. Figure 5 depicts this situation.

(3) Preservation of satisfying assignments: Finally, one has to reproduce the *conjunctive* character of the 3-CNF problem – that is, a sentence is satisfiable (well-formed) if and only if each term has at least one literal assigned the value "T". Part of the disjunctive character of the problem has already been encoded in the feature propagation machinery presented so far; if at least one X_i in a term E_1 expands to the lexical categorization T_i, then the Truth-assignment feature gets the value T. This is just as desired. If one, two, or three of the literals X_i in a term select T_i, then E_1's Truth-assignment feature is T, and the analysis is well-formed. But how do we rule out the case where *all three* X_i's in a term select the "F" path, F_i? And how do we ensure that *all* terms have at least one T below them?

Both of these problems can be solved by resorting to the LFG functional completeness constraint. The trick is to add a Pred feature to a "dummy" node attached to each term; the sole purpose of this feature will be to refer to the feature Truth-assignment, just as the predicate template for the transitive verb kiss mentions the feature Object. Since an analysis is not well-formed if the "grammatical relations" a Pred mentions are not filled in from somewhere, this will have the effect of forcing the Truth-assignment feature to get filled in *every* term. Since the "F" lexical entry does not have a Truth-assignment value, if *all* the X_i's in a term triple select the F_i path (all the literals are "F"), then no Truth-assignment feature is ever picked up from the lexical entries, and that term never gets a value for the Truth-assignment feature. This violates what the predicate template demands, and so the whole analysis is thrown out. (The ill-formedness is exactly analogous to the case where a transitive verb never gets an Object.) Since this condition is applied to each term, we have now guaranteed that *each* term must have at least *one* literal below it that selects the "T" path – just as desired. To actually add the new predicate template, one simply adds a new (but dummy) branch to each term E_i, with the appropriate predicate constraint attached to it. See Figure 6.

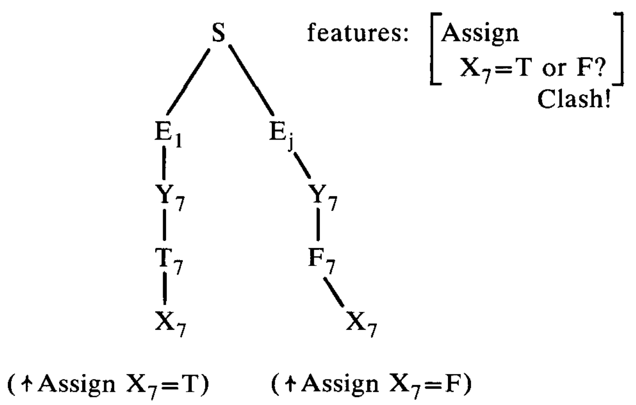

Figure 5. The feature compatibility machinery of LFG can force assignments to be co-ordinated across terms.

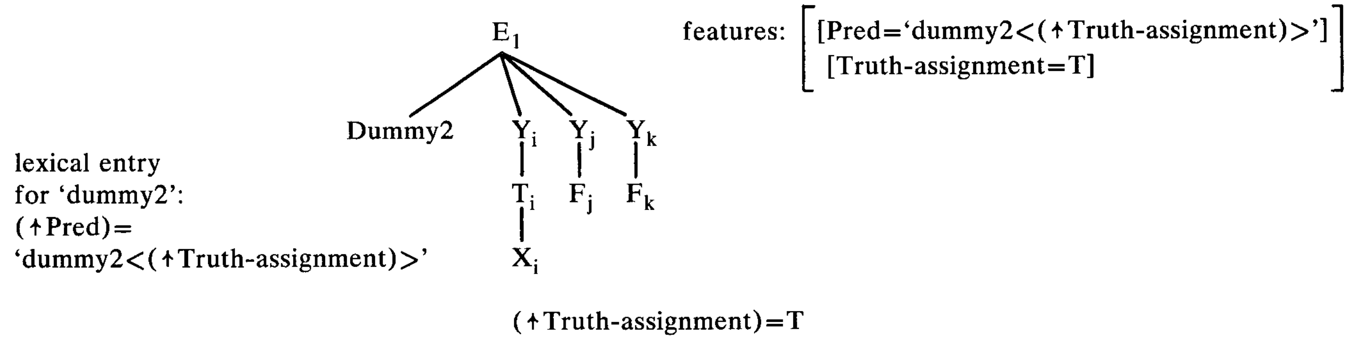
E1
features: [Pred='dummy2<(↑Truth-assignment)>'] [Truth-assignment=T]
Dummy2
Yi Yj Yk
Ti Fj Fk
Xi
lexical entry for 'dummy2': (↑Pred)= 'dummy2<(↑Truth-assignment)>'
(↑Truth-assignment)=T

Figure 6. Predicates can be used to force at least one "T" per term.

There is one final subtle point here: one must also *prevent* the Pred and Truth-assignment features for each term from being passed up to the head "S" node. The reason is that if these features were passed up, then, since the LFG machinery automatically *merges* the values of any features with the same name at the topmost node of the parse tree, the LFG machinery would force the union of the feature values for Pred and Truth-assignment over *all* terms in the analysis tree. The result would be that if *any* term had at least one "T" (hence satisfying the Truth-assignment predicate template in at least one term), then the Pred and Truth-assignment features would get filled in at the topmost node as well. The string below would be well-formed if at least one term were "T", and this would amount to a disjunction of disjunctions (an "OR" of "OR"s), not quite what is sought. To eliminate this possibility, one must add a final trick: *each* term E_1 is given separate Pred, Truth-assignment, and Assign features, but *only* the Assign feature is propagated to the highest node in the parse tree as such. In contrast, the Pred and Truth-assignment features for each term are kept "protected" from merger by storing them under *separate* feature headings labeled $E_1,...,E_n$. The means by which just the Assign feature bundle is lifted out is the LFG analogue of the natural language phenomenon of Subject or Object "control", whereby *just* the features of the Subject or Object of a lower clause are lifted out of the lower clause to become the Subject of Object of a matrix sentence; the remaining features stay unmergeable because they stay protected behind the individually labeled terms.

To actually "implement" this in an LFG, one can add two new branches to each term expansion in the base context-free grammar, as well as two "control" equation specifications that do the actual work of lifting the features from a lower clause to the matrix sentence. A natural language example of this phenomenon is the following (from Kaplan and Bresnan 1981, pp. 43-45):

The girl persuaded the baby to go.

(part of the)
lexical entry for
persuaded: V (↑Vcomp Subject)=(↑Object)

According to this lexical entry, the Object feature structure of a root sentence containing a verb like persuade is to be the same as the feature structure of the Subject of the Complement of persuade – a "control" equation. Since this Subject is the baby, this means that the features associated with the NP the baby are shared with the features of the Object of the matrix sentence.

The satisfiability analogue of this machinery is quite similar to this; see Figure 7.

As Figure 7 shows, a "control equation" should be attached to the A_i node that forces the Assign feature bundle from the C_i side to be lifted up and ultimately merged into the Assign feature bundle of the E_1 node (and then, in turn, to become merged at the topmost node of the tree by the usual full copy up-and-down arrows):

(↑C_iAssign)=↑Assign)

The satisfiability analogue is just like the sharing of the Subject features of a Verb Complement with the Object position of a matrix clause.

To finish off the reduction argument, it must be shown that, given any 3-CNF formula, the corresponding LFG grammar and string as just described can be constructed in a time that is a polynomial function of the length of the original input formula. This is not a difficult task, and only an informal sketch of how it can be done will be given. All one has to do is scan the original formula from left to right, outputting an appropriate cluster of base rules as each triple of literals is scanned: E_i=>A_iC_i; C_i=>Dummy2 $Y_iY_jY_k$; Y_i=>T_i | F_i (similarly for Y_j and Y_k); T_i=>X_i, F_i=>X_i (similarly for T_j and T_k). Note that for each triple of literals in the original input formula the appropriate grammar rules can be output in an amount of time that is just a constant times *n*. In addition, one must also maintain a counter to keep track of the

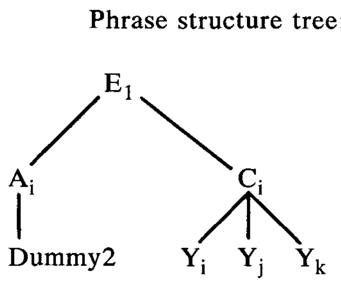

Figure 7. Phrase structure tree required to implement control . equations in CNF analogue of natural language case.

number of triples so far encountered. This adds at most a logarithmic factor, to do the actual counting. At the end of processing the input formula, one must also output the rule $S=>E_1E_2,...,E_m$, where m is the number of triples in the CNF formula. Since *m* is less than *n*, this procedure too is easily seen to take time that is a polynomial function of the length of the original input formula. Finally, one must also construct the lexical entry for each X_i and $\overline{X}_i$. This too can be done as the input formula is scanned left to right. The only difficulty here is that one must check to see if the entry for each X_i has been previously constructed. In the worst case, this involves rescanning the list of lexical entries built so far. Since there are at most *n* such entries, and since the time to actually output a single entry is constant, at worst the time spent constructing a single lexical entry could be proportional to *n*. Thus for *n* entries the total time spent in construction could be at most of order n^2. Since the time to construct the entire grammar is just the sum of the times spent in constructing its production rules and its lexicon, the total time to transform the input formula is bounded above by some constant times n^2.

4. Relevance of Complexity Results and Conclusions

The demonstration of the previous section shows that LFGs have enough power to "simulate" a probably computationally intractable problem. But what are we to make of this result? On the positive side, a complexity result such as this one places the LFG theory more precisely in the complexity hierarchy. If one conjectures, as seems reasonable, that LFG language recognition is actually in the class NP (that is, LFG recognition can be done by a non-deterministic Turing machine in polynomial time), then the LFG languages are NP-complete. This is a plausible conjecture because a non-deterministic Turing machine should be able to "guess" all candidate feature propagation solutions using its non-deterministic power, including any "long-distance" binding solutions (an LFG device not discussed in this paper). Since *checking* candidate solutions is quite rapid – it can be done in n^2 time or less, as described by Kaplan and Bresnan 1981 – recognition should be possible in polynomial time on such a machine. Comparing this result to other known language classes, note that context-sensitive language recognition is in the class of polynomial *space* ("PSPACE"), since (non-deterministic) linear space bounded automata generate exactly the class of context-sensitive languages. (As is well known, for polynomial space the deterministic and non-deterministic classes collapse together because of Savitch's results (see Hopcroft and Ullman 1979) that any function computable in non-deterministic space N can be computed in deterministic space N^2.) Furthermore, the class NP is clearly a subset of PSPACE (since if a function uses space N it must use at least time N) and it is suspected, but not known for certain, that NP is a proper subset of PSPACE (this being the P=NP question once again). Our conclusion is that it is likely that LFGs can generate only a *proper* subset of the context-sensitive languages. This speculation is highly suggestive, in that several other "natural" extensions of the context-free languages – notably the class of languages generated by the so-called "indexed grammars" – also generate strict subsets of the context-sensitive languages, including those strictly context-sensitive languages shown to be generable by LFGs by Kaplan and Bresnan 1981. The class of indexed languages is also known to be NP-complete (see Rounds 1973). Indeed, a cursory look at the power of indexed grammars suggests that they might subsume the machinery of the LFG theory.[13] On the other side of the coin, how might one restrict the LFG theory further so as to avoid potential intractability? Several escape hatches come to mind; these will simply be listed here. Note that all of these "fixes" have the effect of supplying additional constraints to further restrict the LFG theory. In this respect, the LFG complexity demonstration presented here plays the same role as, say Peters and Ritchie's earlier result about Transformational Grammars: it shows that the theory

[13] For a formal discussion of this possibility, see Berwick 1981. Note added in proof: This can be shown to be false, however; LFGs can generate non-indexed languages.

must be tightened *if* one wants to avoid computational intractability.

1. Rule out "worst case" languages as linguistically (that is, empirically) irrelevant.

The probable computational intractability of LFG recognition arises because co-occurrence restrictions (compatible X_i assignments) can be expressed across arbitrary stretches of the terminal string *in conjunction with* potential categorial ambiguity for each terminal item. If some device can be found in natural languages that always filters out or removes such ambiguity locally (so that the choice of whether an item is "T" or "F" never depends on other items arbitrarily far away in the terminal string), or if natural languages never employ such kinds of co-occurrence restrictions, then the reduction is theoretically valid but linguistically irrelevant. Note that such a finding would be a positive result, since one would be able to further narrow the LFG theory in its attempt to characterize all and only the natural languages. This discovery would be on a par with, for example, Peters and Ritchie's observation that although the context-sensitive phrase structure rules formally advanced in linguistic theory have the power to generate non-context-free languages, this power has apparently never been used in the grammars that linguists have designed (see Peters and Ritchie 1973b).

2. Add "locality principles" for recognition (or parsing).

One could simply stipulate that LFG languages must meet additional conditions known to ensure efficient parsability, e.g., Knuth's LR(k) restriction, suitably extended to handle the LFG case. This approach is typified by Marcus's 1980 work, which hypothesized that people normally construct only a single derivation for any given sentence, and proposed other conditions that turn out to guarantee that Knuth's LR(k) restriction will hold. (See Berwick 1982 for further discussion.)

3. Restrict the lexicon.

The reduction argument crucially depends upon having an infinite stock of lexical items and an infinite number of features with which to label them. This is necessary because as CNF formulas grow larger and larger, the number of distinct literals can grow arbitrarily large, and one requires an arbitrarily large number of distinct X_i features to check for co-occurrence conditions. If, for whatever reason, the stock of lexical items or feature labels is finite, then the reduction method works for only finite-sized problems. This restriction seems *ad hoc* in the case of lexical items (why can't there be an infinite number of words?) but less so in the case of features. (If features required "grounding" in terms of other sub-systems of knowledge, e.g, if a feature had to be in the spanning set of a finite number of some hypothetical cognitive or sensory-motor basis elements, then the total number of features would be finite.)[14]

Of course, constraints may be drawn from all three of these general classes in order to make the LFG theory computationally tractable. Even then, it remains to be seen what additional constraints would be required in order to guarantee that LFG recognition takes only a *small* amount of polynomial time – e.g, cubic time or less, as for context-free language recognition. Here it may well turn out to be the case that something like the LR(k) restrictions suffice.

Acknowledgments

I would like to thank Ron Kaplan, Ray Perrault, Christos Papadimitriou, and Stanley Peters for various discussions about the contents of this paper.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Office of Naval Research under ONR contract N00014-80C-0505.

References

Aho, A. 1968 Indexed grammars – an extension of context-free grammars. *Journal of the ACM* **15:4** 647-671.

Berwick, R. 1981 The formal language theory of Lexical-Functional Grammar. Talk given at the Linguistic Society of America Annual Meeting, December 1981.

Berwick, R. 1982 Locality Principles and the Acquisition of Syntactic Knowledge. Ph.D. dissertation. Cambridge, MA: MIT Department of Computer Science and Electrical Engineering.

Berwick, R. and Weinberg, A. 1982 Parsing efficiency, computational complexity, and the evaluation of grammatical theories. *Linguistic Inquiry* **13:1** 165-191.

Berwick R. and Weinberg A. 1983 The role of grammars in models of language use. *Cognition,* 13:1, 1-61.

Berwick, R. and Weinberg, A. forthcoming *The Grammatical Basis of Linguistic Performance: Language Use and Acquisition.* Cambridge, MA: MIT Press.

Chomsky, N. 1965 *Aspects of the Theory of Syntax.* Cambridge, MA: MIT Press.

Chomsky, N. 1980 *Rules and Representations.* New York: Columbia University Press.

Chomsky, N. 1981 *Lectures on Government and Binding.* Dordrecht: Foris Publications.

Fodor, J., Bever, T., and Garrett, M. 1974 *The Psychology of Language.* New York: McGraw-Hill.

Hopcroft, J. and Ullman, J. 1979 *Introduction to Automata Theory, Languages, and Computation.* Reading, MA: Addison-Wesley.

Kaplan, R. and Bresnan, J. 1981 Lexical-functional Grammar: A Formal System for Grammatical Representation. Cambridge, MA: MIT Center for Cognitive Science Occasional Paper #13. (Also forthcoming in Bresnan, J., ed., *The Mental Representation of Grammatical Relations.* Cambridge, MA: MIT Press.)

[14] See for example Pinker 1980, where this claim is made for Lexical-Functional Grammar. Still, it is most natural to assume that there is a *potentially* unbounded number of lexical items – all that is required for the reduction.

Knuth, D. 1965 On the translation of languages from left to right. *Information and Control* **8** 607-639.

Marcus, M. 1980 *A Theory of Syntactic Recognition for Natural Language.* Cambridge, MA: MIT Press.

Peters, S. and Ritchie, R. 1973a On the generative power of transformational grammars. *Information Sciences* **6** 49-83.

Peters, S. and Ritchie, R. 1973b Context-sensitive immediate constituent analysis: context-free languages revisited. *Mathematical Systems Theory* **6:4** 324-333.

Petrick, S. 1965 A Recognition Procedure for Transformational Grammar. Ph.D. dissertation. Cambridge, MA: MIT Department of Linguistics.

Pinker, S. 1980 A Theory of the Acquisition of Lexical-Interpretive Grammars. Cambridge, MA: MIT Center for Cognitive Science Occasional Paper #6. (Also forthcoming in Bresnan, J., ed., *The Mental Representation of Grammatical Relations.* Cambridge, MA: MIT Press.

Rounds, W. 1973 Complexity of recognition in intermediate-level languages. *Proceedings of the 14th Annual Symposium on Switching Theory and Automata.* 145-158.

Rounds, W. 1975 A grammatical characterization of exponential-time languages. *Proceedings of the 16th Annual Symposium on Switching Theory and Automata.* 135-143.