# On the Mathematical Properties of Linguistic Theories[1]

C. Raymond Perrault

Artificial Intelligence Center
SRI International
Menlo Park, CA 94025
and
Center for the Study of Language and Information
Stanford University

**Metatheoretical findings regarding the decidability, generative capacity, and recognition complexity of several syntactic theories are surveyed. These include context-free, transformational, lexical-functional, generalized phrase structure, tree adjunct, and stratificational grammars. The paper concludes with a discussion of the implications of these results with respect to linguistic theory.**

## 1. Introduction

The development of new formalisms for expressing linguistic theories has been accompanied, at least since Chomsky and Miller's early work on context-free languages, by the study of their metatheory. In particular, numerous results on the decidability, generative capacity, and, more recently, the recognition complexity of these formalisms have been published (and rumored!). This paper surveys some of these results and discusses their significance for linguistic theory. However, we will avoid entirely the issue of whether one theory is more descriptively adequate than another. We will consider context-free, transformational, lexical-functional, generalized phrase structure, tree adjunct, and stratificational grammars.[2]

Although this paper focuses on metatheoretic results as arbiters among theories as models of human linguistic capacities, they may have other uses as well. Complexity results could be utilized for making decisions about the implementation of parsers as components of computer-based language-understanding systems. However, as Stanley Peters has pointed out, no one should underestimate the pleasure to be derived from ferreting out these results![3]

## 2. Preliminary Definitions

We assume that the reader is familiar with the basic definitions of regular, context-free (CF), context-sensitive (CS), recursive, and recursively enumerable (r.e.) languages, as well as with their acceptors (see Hopcroft and Ullman 1979). We will be much concerned with the problem of recognizing whether a string is contained in a given language (the **recognition problem**) and with that of

[1] This research was sponsored in part by the National Science and Engineering Research Council of Canada under Grant A9285. It was made possible in part by a gift from the Systems Development Foundation. An earlier version of this paper appeared in the *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, Cambridge, MA, June 1983.

I would like to thank Bob Berwick, Alex Borgida, Jim Hoover, Aravind Joshi, Lauri Karttunen, Fernando Pereira, Stanley Peters, Peter Sells, Hans Uszkoreit, and the referees for their suggestions.

[2] Although we will not examine them here, formal studies of other syntactic theories have been undertaken: e.g. Warren (1979) for Montague's PTQ (1973). Pereira and Shieber (1984) use techniques from the denotational semantics of programming languages to investigate the feature systems of several unification-based theories.

[3] It may be worth pointing out that the introduction of formal argumentation in linguistics has not always been beneficial. Some pseudoformal arguments against rival theories were unquestionably accepted by an audience that did not always have the mathematical sophistication to be critical. For example, Postal's claim (1964b) that two-level stratificational grammars generated only context-free languages was based on an imprecise definition by its proponents, as well as by the failure to see that among the more precise definitions were many very powerful ones.



0362-613X/84/030165-12$03.00

generating one (or all) derivations of the string (the **parsing problem**).

Some elementary definitions from complexity theory may be useful. Further details may be found in Aho et al. (1974). Complexity theory is the study of the resources required by algorithms, usually space and time. Let $f$(x) be a function, say, the recognition function for a language $L$. The most interesting results we could obtain regarding $f$ would be a lower bound on the resources needed to compute $f$ on a machine of a given architecture, say, a von Neumann computer or a parallel array of neurons. These results over whole classes of machines are very difficult to obtain, and none of any significance exist for parsing problems.

Restricting ourselves to a specific machine model and an algorithm $M$ for $f$, we can ask about the **cost** (e.g., in time or space) $c(x)$ of executing $M$ on a specific input $x$. Typically, $c$ is too fine-grained to be useful: what one studies instead is a function $c_w$ whose argument is an integer $n$ denoting the size of the input to $M$, and which gives some measure of the cost of processing inputs of length $n$. Complexity theorists have been most interested in the **asymptotic** behaviour of $c_w$, i.e., the behaviour of $c_w$ as $n$ gets large.

If one is interested in upper bounds on the behaviour of $M$, one usually defines $c_w(n)$ as the maximum of $c(x)$ over all inputs $x$ of size $n$. This is called the **worst-case complexity function** for $M$. Other definitions are possible: for example, one can define the **expected complexity function** $c_e(n)$ for $M$ as the average of $c(x)$ over all inputs of length $n$. $c_e$ might be more useful than $c_w$ if one had an idea as to the distribution of possible inputs to $M$. Not only are realistic distributions rarely available, but the introduction of probabilistic considerations makes the study of expected complexity technically more difficult than that of worst-case complexity. For a given problem, expected and worst-case measures may be quite different.[4]

It is quite difficult to get detailed descriptions of $c_w$; for many purposes, however, a cruder estimate is sufficient. The next abstraction involves ".lumping". classes of $c_w$ functions into simpler ones that demonstrate their asymptotic behaviour more clearly and are easier to manipulate. This is the purpose of O-notation (read "big-oh notation"). Let $f(n)$ and $g(n)$ be two functions. Function $f$ is said to be $O(g)$ if a constant multiple of $g$ is an upper bound for $f$, for all but a finite number of values of $n$. More precisely, $f$ is $O(g)$ if there is are constants $K$ and $n_0$ such that for all $n > n_0$, $f(n) < K * g(n)$.

Given an algorithm $M$, we will say that $M$ is $TIME(g)$ or, equivalently, that its worst-case time complexity is $O(g)$ if the worst-case time cost function $c_w(n)$ for $M$ is $O(g)$.[5] This merely says that almost all inputs to $M$ of size $n$ can be processed in time at most a constant times $g(n)$. It does *not* say that all inputs require $g(n)$ time, or Ruzzo machine that implements $f$. Also, if two algorithms $A_1$ and $A_2$ are available for a function $f$, and if their worst-case complexity can be given respectively as $O(g_1)$ and $O(g_2)$, and $g_1 \leq g_2$, it may still be true that for a large number of cases (maybe even all those likely to be encountered in practice), $A_2$ will be the preferable algorithm simply because the constant $K_1$ for $g_1$ may be much larger than is $K_2$ for $g_2$. A parsing-related example is given in Section 3.

In examining known results pertaining to the recognition complexity of various theories, it is useful to consider how robust they are in the face of changes in the machine model from which they were derived. These models can be divided into two classes: sequential and parallel. Sequential models (Aho et al. 1974) include the familiar single- and multitape Turing machines (TM) as well as random-access machines (RAM) and random-access stored-program machines (RASP). A RAM is like a TM except that its working memory is random-access rather than sequential. A RASP is like a RAM but stores its program in its memory. Of all these models, the RASP is most like a von Neumann computer.

All these sequential models can simulate one another in ways that do not require great changes in time complexity. For example, a $k$-tape Turing Machine that runs in time $O(t)$ can be simulated by a RAM in time $O(t \log t)$, conversely, a RAM running in $O(t)$ can be simulated by a $k$-tape TM in time $O(t^2)$.. In fact, all the familiar sequential models are **polynomially related:** they can simulate one another with at most a polynomial loss in efficiency.[6] Thus, if a syntactic model is known to have a difficult recognition problem when implemented on one sequential model, execution of an equivalent algorithm on another sequential machine will not be much easier.

Transforming a sequential algorithm to one on a parallel machine with a fixed number $K$ of processors provides at most a factor $K$ improvement in speed. More interesting results are obtained when the number of processors is allowed to grow with the size of the problem, e.g., with the length of the string to be parsed. These processors can be viewed as connected together in a circuit, with inputs entering at one end and outputs being produced at the other. The *depth* of the circuit, or the maximum number of processors that data must be passed through from input to output, corresponds to the parallel *time* required to complete the computation. A problem that has a solution on a *sequential* machine in polynomial time and in space $s$ will have a solution on a *parallel* machine with a polynomial number of processors and circuit depth (and hence parallel time) $O(s^2)$. This means that algorithms with sequential solutions requiring small space (such as deterministic CSLs) have fast parallel solutions.

[4] Hoare's Quicksort algorithm, for example, has expected time complexity of $O(n \log n)$ and worst-case complexity of $O(n^2)$, using notation defined in the next paragraph.

[5] Similarly, let $M$ be $SPACE(g)$ if the worst-case space complexity of $M$ is $O(g)$.

[6] RAMs and RASPs are allowed to store arbitrarily large numbers in their registers. These results assume that the cost of performing elementary operations on those numbers is proportional to their length, i.e. to their logarithm.

For a comprehensive survey of parallel computation, see Cook (1981).

## 3. Context-Free Languages

Recognition techniques for context-free languages are well known (Aho and Ullman 1972). The so-called CKY or "dynamic programming" method is attributed by Hays (1962) to J. Cocke; it was discovered independently by Kasami (1965) and Younger (1967), who showed it to be $O(n^3)$. It requires the grammar to be in Chomsky Normal Form, and putting an arbitrary grammar in CNF may square its size. Berwick and Weinberg (1982) point out that, since the complexity of parsing algorithms is generally at least linearly dependent on the size of the grammar, this requirement may make CKY less than optimal for parsing short sentences.

Earley's algorithm recognizes strings in arbitrary CFGs in time $O(n^3)$ and space $O(n^2)$, and in time $O(n^2)$ for unambiguous CFGs. Graham, Harrison, and Ruzzo (1980) offer an algorithm that unifies CKY and Earley's algorithm (1970), and discuss implementation details.

Valiant (1975) showed how to interpret the CKY algorithm as the finding of the transitive closure of a matrix and thus reduced CF recognition to matrix multiplication, for which subcubic algorithms exist. Because of the enormous constants of proportionality associated with this method, it is not likely to be of much practical use, either an implementation method or as a "psychologically realistic" model.

Ruzzo (1979) has shown how CFLs can be recognized by Boolean circuits of depth $O(\log(n)^2)$, and therefore that parallel recognition can be accomplished in time $O(\log(n)^2)$. The required circuit size is polynomial in $n$.

So as not to be mystified by the *upper bounds* on CF recognition, it is useful to remember that no known CFL requires more than linear time, nor is there even a nonconstructive proof of the existence of such a language.

This is also a good place to recall the difference between recognition and parsing: if parsing requires that distinct structures be produced for all parses, it will be $TIME(2^n)$, since in some grammars sentences of length $n$ may have $2^n$ parses (Church and Patil 1982). For an empirical comparison of various parsing methods, see Slocum (1981).

## 4. Transformational Grammar

From its earliest days, discussions of transformational grammar (TG) have included consideration of matters computational.

Peters and Ritchie (1973a) provided some the first nontrivial results regarding the generative power of TGs. Their model reflects the *Aspects* version quite faithfully, including transformations that move and add constituents, and delete them subject to recoverability. All transformations are obligatory, and applied cyclically from the bottom up. They show that every r.e. set can be generated by applying a set of transformations to a context-sensitive base. The proof is quite simple: the right-hand sides of the type-0 rules that generate the r.e. set are padded with a new "blank" symbol to make them at least as long as their left-hand sides. Rules are added to allow the blank symbols to commute with all others. These context-sensitive rules are then used as the base of a TG whose only transformation deletes the blank symbols.

Thus, if the transformational formalism itself is supposed to *characterize* the grammatical strings of possible natural languages, then the only languages being excluded by the formalism are those that are not enumerable under any model of computation. The characterization assumption is further discussed in Section 9.

At the expense of a considerably more intricate argument, the previous result can be strengthened (Peters and Ritchie 1971) to show that every r.e. set can be generated by a context-free based TG, as long as a **filter** – an intersection with a regular set – can be applied to the phrase-markers produced by the transformations. In fact, the base grammar can be *independent* of the language being generated. The proof involves the simulation of a TM by a TG. The transformations first generate an "input tape" for the TM being simulated, then apply the TM productions, one per cycle of the grammar. The filter ensures that the base grammar will generate just as many S nodes as necessary to generate the input string and do the simulation. In this case too, if the transformational formalism is supposed to characterize the possible natural languages, the **universal base hypothesis** (Peters and Ritchie 1969), according to which all natural languages can be generated from the same base grammar, is empirically vacuous: *any* recursively enumerable language can.

Following Peters and Ritchie's work, several attempts were made to find a restricted form of the transformational model that is descriptively adequate, yet whose generated languages are recursive (see, for example, LaPointe 1977). Since a key part of the proof in Peters and Ritchie (1971) involves the user of a filter on the final derivation trees, Peters and Ritchie (1973c) examined the consequences of forbidding final filtering. They show that, if $S$ is the recursive symbol in the CF base, the generated language $L$ is *predictably enumerable* and *exponentially bounded.* A language $L$ is **predictably enumerable** if there is an "easily" computable function $t(n)$ that gives an upper bound on the number of tape squares needed by its enumerating TM to enumerate the first $n$ elements of $L$. $L$ is **exponentially bounded** if there is a constant $K$ such that, for every string $x$ in $L$, there is another string $x'$ in $L$ whose length is at most $K$ times the length of $x$.

The class of nonfiltering languages is quite unusual, including all the CFLs (obviously), but also properly intersecting the CSLs, the recursive languages, and the r.e. languages.

The source of nonrecursivity in transformationally generated languages is that transformations can delete large parts of the tree, thus producing surface trees that are arbitrarily smaller than the deep structure trees they

were derived from. This is what Chomsky's "recoverability of deletions" condition was meant to avoid. In his thesis, Petrick (1965) defines the following condition on transformational derivations: a derivation satisfies the terminal-length-increasing condition if the length of the yield of any subtree $u$, resulting from the application of the transformational cycle to a subtree $t$, is greater than the length of the yield of any subtree $u'$ resulting from the application of the cycle to a subtree $t'$ of $t$.

Petrick shows that, if all recursion in the base grammar "passes through S" and all derivations satisfy the **terminal-length-increasing** condition, then the generated language is recursive. Using a slightly more restricted model of transformations Rounds (1973) strengthens this result by showing that the resulting languages are in fact context-sensitive.

In an unpublished paper, Myhill shows that, if Petrick's condition is weakened to terminal-length-nondecreasing, the resulting languages can be recognized in space that is at most *exponential* in the length of the input. This implies that recognition can be done in at most double-exponential time, but Rounds (1975) proves that not only can recognition be done in *exponential time,* but that every language recognizable in exponential time can be generated by a TG satisfying the terminal-length-nondecreasing condition and recoverability of deletions.

This is a very strong result, because of the closure properties of the class of exponential-time languages. To see why this is so requires a few more definitions.

Let **P** be the class of all languages that can be recognized in polynomial time on a deterministic TM, and **NP** the class of all languages that can be recognized in polynomial time on a nondeterministic TM. **P** is obviously contained in **NP**, but the converse is not known, although there is much evidence that it is false.

There is a class of problems, the so-called **NP-complete** problems, which are in NP and "as difficult" as any other problems in NP in the following sense: if any of them could be shown to be in P, all the problems in NP would also be in P. One way to show that a language $L$ is NP-complete is to show that $L$ is in NP and that every other language $L_0$ in NP can be **polynomially transformed** into $L$, – i.e., that there is a deterministic TM, operating in polynomial time, that will transform an input $w$ to $L$ into an input $w_0$ to $L_0$ such that $w$ is in $L$ if and only if $w_0$ is in $L_0$. In practice, to show that a language is NP-complete, one shows that it is in NP and that some already known NP-complete language can be polynomially transformed into it.

All the known NP-complete languages can be recognized in exponential time on a deterministic machine, and none have been shown to be recognizable in less than exponential time. Thus, since the restricted transformational languages of Rounds *characterize* the exponential languages, if all of them were to be in P, P would be equal to NP. Putting it another way, if P is not equal to NP, some transformational languages (even those satisfying the terminal-length-nonincreasing condition) have no "tractable" (i.e., polynomial-time) recognition procedures on any deterministic TM. It should be noted that this result also holds for all the other known sequential models of computation, as they are all polynomially related, and even for parallel machines with as many as a polynomial number of processors.

All the results outlined so far in this section are inspired by the model of transformational grammar presented in *Aspects*. More recent versions of the theory are substantially different, primarily in that most of the constructions handled in terms of deletions from the base trees are now handled using traces (i.e., constituents with no lexical material) indexed to other constituents. In his contribution to this issue (p. 189), Berwick presents a formalization of the theory of Government and Binding (GB) and some of its consequences. The formalization is unusual in that it reduces grammaticality to well-formedness conditions on what he calls annotated surface structures. From these conditions, two results follow. One is that for every GB grammar $G$ there is a constant $K$ such that for every string $w$ in $L(G)$ and for every annotated surface structure $s$ whose yield is $w$, the number of nodes in $s$ is bounded by $K*length(w)$. This, of course, ensures that the $L(G)$ is recursive. The second result is that GB languages all have the linear growth or arithmetic growth property: for every sufficiently long string $w$ in a GB language $L$ there is another string $w'$ in $L$ which is at most $K$ symbols shorter than $w$.

A few comments about Berwick's formalization and results are in order. To begin with, the formalization is clearly a quite radical simplification of current practice among GB practitioners, as it does not reflect D-structure, LF, or PF, nor case theory, the theta-criterion, and control theory. Thus, in its current form, the formalization does not include the machinery necessary to account for passives and raising. It also assumes that $X$-bar theory limits the base to trees generated by CFGs with no useless nonterminals and no cycles, except presumably through the $S$ and $NP$ nodes. This excludes accounts of stacked adjectives, as in *the white speckled shaggy Pekingese*, and of stacked relative clauses.

We suspect that most of these features could be added to the formalization without affecting either result, and that it is extremely useful to have even a first approximation of one to work with. Although Berwick is mute on the subject, we conjecture that recognition in the model he gives can be done in polynomial time. What is less clear is what will happen to recognition complexity under models that include the other constraints.

Berwick's result about the linear growth property has no immediate functional consequence for complexity or even for weak generative capacity. It is presented as a property that natural languages seem to have and thus that should be predicted by the linguistic model.

## 5. Lexical-Functional Grammar

In part, transformational grammar seeks to account for a range of constraints or dependencies within sentences. Of particular interest are subcategorization, predicate-argument dependencies, and long-distance dependencies, such as *wh*-movement. Several recent theories suggest different ways of accounting for these dependencies, but without making use of transformations. We examine three of these in the next several sections: lexical-functional grammar, generalized phrase structure grammar, and tree adjunct grammar.

In the lexical-functional grammar (LFG) of Kaplan and Bresnan (1982), two levels of syntactic structure are postulated: constituent and functional. All the work done previously by transformations is instead encoded both in the lexicon and in links established between nodes in the constituent and functional structures.

The languages generated by LFGs, or LFLs, are CSLs and properly include the CFLs (Kaplan and Bresnan 1982). Berwick (1982) shows that a set of strings whose recognition problem is known to be NP-complete, namely, the set of satisfiable Boolean formulas, is an LFL. Therefore, as was the case for Rounds's restricted class of TGs, if P is not equal to NP, then some languages generated by LFGs do not have polynomial-time recognition algorithms. Indeed only the "basic" parts of the LFG mechanism are necessary to the reduction. This includes mechanisms necessary for feature agreement, for forcing verbs to take certain cases, and for allowing lexical ambiguity. Thus, no simple change in the formalism is likely to avoid the combinatorial consequences of the full mechanism. It should be noted that the c-structures and f-structures necessary to make satisfiable Boolean formulas into an LFL are not much larger than the strings themselves; the complexity comes in finding the assignment of truth-values to the variables. In his paper in this issue (p. 189), Berwick argues that the complexity of LFLs stems from their ability to unify trees of arbitrary size, and that such a mechanism does not exist in GB. However, the recognition complexity of GB languages, as formalized in Berwick (1984) or in more "faithful" models, remains open, and may arise from other constraints.

Both Berwick and Roach have examined the relation between LFG and the class of languages generated by **indexed grammars** (Aho 1968), a class known to be a proper subset of the CSLs, but including some NP-complete languages (Rounds 1973). They claim (personal communication) that the indexed languages are a proper subset of the LFLs.

## 6. Generalized Phrase Structure Grammar

In a series of papers, Gerald Gazdar and his colleagues (1982) have argued for a joint account of syntax and semantics that is like LFG in eschewing the use of transformations, but unlike it in positing only one level of syntactic description. The syntactic apparatus is based on a nonstandard interpretation of phrase-structure rules and on the use of metarules. The formal consequences of both these devices have been investigated.

### 6.1. Node admissibility

There are two ways of interpreting the function of CF rules. The first, and most common, is to treat them as rules for *rewriting strings*. Derivation trees can then be seen as canonical representatives of classes of derivations producing the same string, differing only in the order in which the same productions are applied.

The second interpretation of CF rules is as *constraints* on derivation trees: a legal derivation tree is one in which each node is "admitted' by a rule, i.e., each node dominates a sequence of nodes in a manner sanctioned by a rule. For CF rules, the two interpretations obviously generate the same strings and the same set of trees.

Following a suggestion of McCawley's, Peters and Ritchie (1973b) showed that, if one considered context-sensitive rules from the node-admissibility point of view, the languages defined were still CF. Thus, for example, the use of CS rules in the base to impose subcategorization restrictions does not increase the weak generative capacity of the base component. (For some different restrictions of context-sensitive rules that guarantee that only CFLs will be generated, see Baker (1972).)

Rounds (1970b) gives a simpler proof of Peters and Ritchie's node admissibility result, using the techniques from tree-automata theory, a generalization to trees of finite state automata theory for strings. Just as a finite-state automaton (FSA) **accepts** a string by reading it one character at a time, changing its state at each transition, a finite-state tree automaton (FSTA) traverses trees, propagating states. The **top-down FSTA** "attaches" a starting state (from a finite set) to the root of the tree. Transitions are allowed by productions of the form

$$(q, a, n) => (q_1,...,q_n)$$

such that if state $q$ is being applied to a node labeled $a$ and dominating $n$ descendants, then state $q_i$ should be applied to its $i$th descendant. Acceptance occurs if all leaves of the tree end up labeled with states in the accepting subset. The **bottom-up FSTA** is similar: starting states are attached to the leaves of the tree and the productions are of the form

$$(a, n, (q_1,..., q_n) => q)$$

indicating that, if a node labeled $a$ dominates $n$ descendants, each labeled with states $q_1$ to $q_n$, then node $a$ gets labeled with state $q$. Acceptance occurs when the root is labeled by a state from the subset of accepting states.

As is the case with FSAs, FSTAs of both varieties can be either deterministic or nondeterministic. A set of trees is said to be **recognizable** if it is accepted by a nondeterministic bottom-up FSTA. Once again, as with FSAs, any set of trees accepted by a nondeterministic bottom-up FSTA is accepted by a deterministic bottom-up FSTA, but the result does not hold for top-down

FSTA, even though the recognizable sets are exactly the languages recognized by nondeterministic top-down FSTAs.

A set of trees is **local** if it is the set of derivation trees of a CF grammar. Clearly, every local set is recognizable by a one-state bottom-up FSTA that checks at each node to verify that it satisfies a CF production. Furthermore, the **yield** of a recognizable set of trees (the set of strings it generates) is CF. Not all recognizable sets are local: an example is the set of trees that satisfies the constraints of $X$-bar theory and the $\theta$-criterion. However, they can all be mapped into local sets by a simple homomorphic mapping.[7] Rounds's proof (1970a) that CS rules under node admissibility generate only CFLs involves showing that the set of trees accepted by the rules is recognizable – i.e., that there is a nondeterministic bottom-up FSTA that can check at each node that some node admissibility condition holds there. This requires confirming that the "strictly context-free" part of the rule holds and that a proper analysis of the tree passing through the node satisfies the "context-sensitive" part of the rule.

Joshi and Levy (1977) strengthened Peters and Ritchie's result by showing that the node admissibility conditions could also include arbitrary Boolean combinations of **dominance** conditions: a node could specify a bounded set of labels that must occur either immediately above it along a path to the root, or immediately below it on a path to the frontier.

In general, the CF grammars constructed in the proof of weak equivalence to the CS grammars under node admissibility are much larger than the original, and not useful for practical recognition. Joshi, Levy, and Yueh (1981), however, show how Earley's algorithm can be extended to a parser that uses the local constraints directly.

### 6.2. Metarules

The second important mechanism used by Gazdar (1982) is **metarules,** or rules that apply to rules to produce other rules. Using standard notation for CF rules, one example of a metarule that could replace the *Apects* transformation known as "particle movement" is

$$V \rightarrow V\ N\ Pt\ X \Rightarrow V \rightarrow V\ Pt\ N[\text{-}PRO]\ X$$

The symbol $X$ here behaves like variables in structural analyses of *Aspects* transformations. If such variables are restricted to being used as *abbreviations,* that is, if they are allowed to range only from a *finite* subset of strings over the vocabulary, then closing the grammar under the metarules produces only a finite set of derived rules; and thus the generative power of the formalism is not increased. If, on the other hand, $X$ is allowed to range over strings of *unbounded length,* as are the **essential variables** of transformational theory, then the consequences are less clear. It is well known, for example, that, if the right-hand sides of phrase structure rules are allowed to be arbitrary regular expressions, the generated languages are still context-free. Might something like this not be happening with essential variables in metarules? It turns out that such is not the case.

The formal consequences of the presence of essential variables in metarules depend on the presence of another device, the so-called **phantom categories.** It may be convenient in formulating metarules to allow, in the left-hand sides of rules, occurrences of syntactic categories that are never introduced by the grammar, i.e., that never appear in the right-hand sides of rules. In standard CFLs, these are called *useless categories;* rules containing them can simply be dropped, with no change in weak generative capacity. Not so with metarules: it is possible for metarules to be used to rewrite rules containing phantom categories into rules without them. Such a device was proposed at one time as a way to implement passives in the GPSG framework.

Uszkoreit and Peters (1983) have shown that essential variables in metarules are powerful devices indeed: CF grammars with metarules that use at most one essential variable and allow phantom categories can generate all recursively enumerable sets. Even if phantom categories are banned, some nonrecursive sets can be generated as long as the use of at least one essential variable is allowed.

Two constraints on metarules have been proposed to restrict the generative capacity of metarule systems. Gazdar (1982) has suggested replacing essential variables by abbreviative ones, i.e. variables that can only range over a finite set of (predetermined) alternatives. Shieber et al. (1983) argue that a generalization is lost in so doing, in the sense that the class of instantiations of the variable must be defined by extension rather than by intension. Given the alternative, this seems a small price to pay.

The other constraint, suggested by Gazdar and Pullum (1982), is *finite closure* of the metarule derivation process: no metarule is allowed to apply more than once in the derivation of a rule. Shieber et al. (1983) present several examples, namely the treatment of discontinuous noun phrases in Walpiri, adverb distribution in German, and causatives in Japanese, that cannot be handled under the finite closure constraint.

It should be noted that other ways of using one grammar to generate the rules of another have been proposed. VanWijngaarden (1969), for example, presented a scheme in which one grammar's *sentences* are the *rules* of another. Greibach (1974) gives some of its properties.

## 7. Tree Adjunct Grammar

The tree adjunct grammars (TAG) of Joshi and his colleagues (1982, 1984) provide a different way of accounting for syntactic dependencies. A TAG consists of two finite sets of finite trees, the **centre** trees and the **adjunct** trees.

[7] This mapping is a bottom-up finite-state tree transducer that simply labels each node with the state the recognizing bottom-up FSTA would have been in at that node.

The centre trees correspond to the surface structures of the "kernel" sentences of the languages. The root of the adjunct trees is labelled with a nonterminal symbol that also appears exactly once on the frontier of the tree. All other frontier nodes are labelled with terminal symbols. Derivations in TAGs are defined by repeated application of the **adjunction operation.** If $c$ is a centre tree containing an occurrence of a nonterminal $A$, and $a$ is an adjunct tree whose root (and one node $n$ on the frontier) is labelled $A$, then the adjunction of $a$ to $c$ is performed by "detaching" from $c$ the subtree $t$ rooted at $A$, attaching $a$ in its place, and reattaching $t$ at node $n$. Adjunction may then be seen as a tree analogue of a context-free derivation for strings (Rounds 1970a). The string languages obtained by taking the yields of the tree languages generated by TAGs are called **tree adjunct languages** (TAL).

In TAGs, all long-distance dependencies are the result of adjunctions separating nodes that at one point in the derivation were "close". Both crossing and noncrossing dependencies can be represented (Joshi 1983)). The formal properties of TALs are fully discussed by Joshi, Levy, and Takahashi (1975); Joshi and Levy (1982); and Yokomori and Joshi (to appear). Of particular interest are the following.

TALs properly contain the CFLs and are properly contained in the indexed languages, which in turn are properly contained in the CSLs. Although the indexed languages contain NP-complete languages, TALs are much better behaved: Joshi and Yokomori report (personal communication) an $O(n^4)$ recognition algorithm and conjecture that an $O(n^3)$ bound may be possible.

## 8. Stratificational Grammar

The constituent and functional structures of LFG, the metarules of GPSG, the constraints on deep and surface structures in TG, and the two-level grammars of van Wijngaarden are all different ways in which syntactic constraints can be distributed across more than one structure. The Stratificational Grammar (SG) of Lamb and Gleason (Lamb 1966, Gleason 1964) is yet another.

SG postulates the existence of several coupled components, known as **strata;** phonology, morphology, syntax, and semology are examples of linguistic strata. Each stratum specifies a set of correct structures, and an utterance has a representative structure at each stratum. The strata are linearly ordered and constrained by a **realization** relation.

Following Gleason's model, Borgida (1983) defines the realization relation so that it couples the application of specific pairs of *productions* (or sequences of productions) in the different grammars. Note that this is a generalization of the pairing of syntactic and semantic rules suggested by Montague, for example.

With any derivation in a rewrite grammar, one can associate a string of the productions used in the derivation. If a canonical order is imposed on the derivations – for example, that the leftmost nonterminal must be the next one to be expanded – a unique string of productions can be associated with each derivation tree. A **two-level stratificational grammar** consists of two rewrite grammars $G_1$ and $G_2$, called **tactics,** with sets of productions $P_1$ and $P_2$, respectively, and a realization relation $R$, which is a finite set of pairs, each consisting of a string of productions of $P_1$ and a string of productions of $P_2$. A derivation $D_1$ in $G_1$ is **realized** by a derivation $D_2$ in $G_2$ if the strings of productions $s_1$ and $s_2$ associated with $D_1$ and $D_2$ can be decomposed into substrings $s_1=u_1..u_n$ and $s_2=v_1...v_n$, respectively, such that $R(u_i,v_i)$, for all $i$ from 1 to $n$. The language generated by a two-level SG is the set of string generated derivations in $G_2$ that realize derivations in $G_1$.extended to more than two **strata.**

Because the realization relation binds *derivations,* it is the strong generative capacity of the tactics that determines the languages generated. Borgida (1983) studied the languages of two-level SGs as the strong generative capacity of the tactics is systematically varied. Some of his results are unexpected. All r.e. languages can be generated by two-level SGs with CF tactics. On the other hand, if the upper tactics are restricted to being right-recursive, only CFLs can be generated, even with type 0 lower tactics. If the grammars are restricted to have no length-decreasing rules, the languages describable by SGs lie in the class of *quasi-real time languages,* defined as recognizable by nondeterministic TMs in linear time.

The principal feature of SGs that accounts for high generative power is the presence of left recursion in the tactics: to escape from the regular languages, one needs left recursion on at least one stratum; to escape context-free languages, two non-right-recursive strata are needed. These results apply to SGs with arbitrary number of strata.

## 9. Seeking Significance

How, then, can metatheoretical results be useful in selecting among syntactic theories? The obvious route, of course, is to claim that the computationally most restrictive theory is preferable. However, this comparison is useful only if the theories to be compared rest on a number of shared assumptions and observations concerning the scope of the syntax, the computational properties of the human processor and the relation between the processor and the syntactic theory.

In this section, we first briefly consider the assumption of common syntactic coverage and the computational consequences of theory decomposition. We then ask how metatheoretical results can be used first as lower bounds and then as upper bounds on acceptable theories.

### 9.1. Coverage

Competing linguistic theories must obviously agree on the burden of their respective syntactic components. We consider here one example of a constraint for which two analyses have been presented, one purportedly completely syntactic, and the other partly semantic. The

problem at hand is the distribution of the so-called *polarity-sensitive* items, such as *any* and the metaphorical sense of *lift a finger.* Simply put, these terms need to appear within the scope of a **polarity reverser,** such as *not*, or *rarely.* The question is: how are scope and polarity reverser defined? In Linebarger's syntactic analysis (1980), the scope relation is defined on the logical forms of the government and binding theory (GB):

> An item is in the immediate scope of NOT if (1) it occurs only in the proposition which is the entire scope of NOT and (2) within the proposition there are no logical elements intervening between it and NOT.

In this analysis, *scope* and *intervening* must be defined configurationally, and one assumes that *logical element* is defined in the lexicon. Note that *not* is the only lexical element that can be a license. Linebarger assumes that

> all other cases are, strictly speaking, ill formed and salvaged only by the availability of an implicature which can be formalized to contain the polarity items in the appropriate relation to NOT. (Ladusaw 1983)

Ladusaw's analysis (1979), within the framework of Montague grammar, is in three parts:

1. A negative polarity item will be acceptable only if it is in the scope of a polarity-reversing expression.
2. For any two expressions $\alpha$ and $\beta$, constituents of a sentence S, $\alpha$ is in the scope of $\beta$ with respect to a composition structure of S, S′, iff the interpretation of $\alpha$ is used in the formulation of the argument of $\beta$'s interpretation in S′.
3. An expression D is a polarity reverser with respect to an interpretation function $\phi$ if and only if, for all expressions $X$ and $Y$,[8]

$$\phi(X) \subseteq \phi(Y) \Rightarrow \phi(d(Y)) \subseteq \phi(d(X))$$

In (1), "acceptable" is predicated of negative polarity items; these are clearly parts of surface structures, and thus syntactic objects. The condition on acceptability is in terms of *scope* and *polarity-reversing expression.* In (3), *polarity reverser* is applied to syntactic objects and defined in terms of their denotations. In (2) *$\alpha$ is in the scope of $\beta$.* is defined again of syntactic objects $\alpha$ and $\beta$, but in terms of the function that interprets the structure they occur in, not of their denotations. So the condition applies to syntactic structures, but is defined in terms of the denotations of parts of that structure and in terms of the interpretation function itself. Although it would be satisfying to do so, there appears to be no natural way to recast Ladusaw's constraint as one that is fully semantic, namely, by making the interpretation function partial (i.e., in a way that allows *John knows anything* to be grammatical but uninterpretable) because the definition of scope is in terms of the interpretation function, not the denotations themselves. We seem condemned to straddle the fence on this one.

Thus we have here one theory that deals, completely within the syntactic domain, only with the license *not*, and another that accounts for a much broader range of licenses by imposing on syntactic structures conditions defined in terms of their interpretations and of the interpretation function itself. They are computationally incomparable.

We close this section with an aside on the separation of constraints. Constraint separation can occur in two ways. In the case of polarity-sensitive items, it takes place across the syntax-semantics boundary. In several *syntactic* theories, such as GB and LFG, it can also occur within the syntactic theory itself: grammaticality in LFG, for example, is defined in terms of the existence of pairs of appropriately related constituent and functional structures.

In general, the class resulting from the intersection of the separated classes will be at least as large as either of them: e.g., the intersection of two CFLs is not always a CFL. More interesting is the fact that separation sometimes has beneficial computational effects. Consider, for example, the constraint in many programming languages that variables can only occur in the scope of a declaration for them. This constraint cannot be imposed by a CFG but can be by an indexed grammar, at the cost of a dramatic increase in recognition complexity. In practice, however, the requirement is simply not checked by the parser, which only recognizes CFLs. The declaration conditions are checked separately by a process that traverses the parse tree. In this case, the overall recognition complexity remains some low-order polynomial. It is not clear to me whether one wants to consider the declaration requirement syntactic or not. The point is that, in this case, the "unified account" is more general, and computationally more onerous, than the modular one. Some arguments of this kind can be found in Berwick and Weinberg (1982).

## 9.2. Metatheoretical results as lower bounds

The first use of formal results is to argue that a theory should be rejected if it is insufficiently powerful to account for observed constraints. Chomsky used this strategy initially against finite-state grammars[9] and then against CFGs. It obviously first requires extracting from empirical observation (and decisions about idealization) what the minimal generative capacity and recognition complexity of actual languages are. Several arguments have been made against the weak generative adequacy of CFGs. The best known of these are Bar-Hillel's claim (1961) based on the occurrence of *respectively* and Postal's (1964a) on nominalization in Mohawk. Higginbotham (1984) claims non-context-freeness for English

[8] Following Fauconnier, Ladusaw's denotation functions take as their values sets, ordered as usual. Sentences, for example, get as denotations the set of all worlds in which they are true.

[9] There has always been interest in finite-state grammars to account for some perceptual constraints on sentence recognition, such as the difficulty of center-embedded sentences – e.g., "The rat that the cat that the dog chase ate died" (Langendoen 1975, Church 1981, Langendoen and Langsam 1984). They have also provided useful models in morphology (Kay 1983, Koskenniemi 1983) and phonology (Church 1983),

on the basis of sentences containing *such that*. Postal and Langendoen (this issue, p. 177) do so with cases of sluicing. Pullum and Gazdar (1982) (convincingly, I believe) refute the first two cases by claiming that the constraints on which they are based do not in fact hold. Similarly, Pullum (this issue, p. 182) argues against Postal and Langendoen, and against Higginbotham, again on the basis of the linguistic facts. Pullum and Gazdar also consider the case of verb and noun-phrase ordering in Dutch; although they show that no evidence has been given suggesting that the weak generative capacity of Dutch is greater than context-free, the phrase structure trees generated by their fragment are not obviously adequate for a compositional semantic analysis. This point is also made by Bresnan et al. (1982).

The most convincing evidence so far against the weak context-freeness of natural languages comes from Swiss-German. Shieber (1984) shows that, like Dutch, Swiss-German allows cross-serial order in subordinate clauses but also requires that objects be marked for case, as in German. Given that the verb *hälfed* 'help' takes a dative object while *aastriiche* 'paint' and *lönd* 'let' take accusative objects, we get the following subordinate clauses, which can be made into complete sentences by prefixing them with *Jan säit das* 'Jan says that'.

. . . mer em Hans es huus hälfed aastriiche
. . . we Hans-DAT the house-ACC helped paint
. . . we helped Hans paint the house

. . . *me em Hans es huus lönd aastriiche
. . . we Hans-DAT the house-ACC let paint
. . . we let Hans paint the house

. . . mer d'chind em Hans es huus lönd hälfed aastriiche
. . . we the children-ACC Hans-DAT the house-ACC let help paint
. . . we let the children help Hans paint the house

. . . *mer d'chind de Hans es huus lönd hälfed aastriiche
. . . we the children-ACC Hans-ACC the house-ACC let help paint
. . . we let the children help Hans paint the house

The proof that Swiss-German (SG) is not context-free is classic: intersect SG with the following regular language:

Jan säit das mer (d'chind)*(em Hans)*
es huus händ wele (laa)*(hälfe)* aastriche.

With some care, Shieber argues from the data that $SG \cap L$ is the language

Jan säit das mer (d'chind)$^m$ (em Hans)$^m$
es huus händ wele (laa)$^m$ (hälfe)$^m$ aastriche.

which is not context-free. Since context-free languages are closed under intersection with regular languages, Swiss-German is not context-free either.

Hintikka (1977) claims that English is not recursive, let alone context-free, based on the distribution of the words *any* and *every*. His account of why *John knows everything* is grammatical while *John knows anything* is not, is that *any* can appear only in contexts where replacing it with *every* changes the meaning. If equivalence of meaning is taken to be logical equivalence, this means that grammaticality is dependent on the determination of equivalence of logical formulas, an undecidable problem.

Several responses could be made to Hintikka's claim. One is to argue, as did Ladusaw (1979), that the constraint is semantic, not syntactic. Another route, followed by Chomsky (1980), is to claim that a simpler solution is available, namely, one that replaces logical equivalence with syntactic identity of some kind of logical form. This is the basis for Linebarger's analysis.

### 9.3. Metatheoretical results as upper bounds

In the preceding section, we discussed ways in which formal results about syntactic theories can be used against them on the grounds that they show them to be insufficiently powerful to account for the observed data. Now, given a theory that is powerful enough, can its formal properties be used against it on the basis that it fails to *exclude* impossible languages?

The classic case of an argument of this form is Peters and Ritchie's argument against the TG model, discussed in Section 4.

More generally, the premises are the following:

1. The possible languages are decidable.
2. The correct syntactic theory must generate exactly the possible languages.
3. The correct syntactic theory is T.
4. The class of languages C generated by T is a priori too large to be the class of possible languages.

One conclusion from this argument is that theory T is incorrect, i.e., that assumption (3) fails. Chomsky rejects assumption (1) instead, insisting that the possible languages are those that can be *learned*.[10]

Although Chomsky also claims that the class of possible languages is *finite*,[11] the crucial concern here is that, finite or not, the class of possible languages could contain languages that are not recursive, or even not recursively enumerable. For example, let $L$ be a non-recursive language and $L'$ its complement (also non-recursive). Let $s$ be some string of L and $s'$ some string of $L'$. The procedure by which the subject chooses $L$ if $s$ is encountered before $s'$ and $L'$ otherwise will learn one of $L$ or $L'$.

[10] Learning algorithms can be compared along several dimensions. For a mathematical framework for learnability theory, see Osherson et al. (1983).

[11] Actually, finiteness is claimed for the class of core grammars, from which the possible languages are assumed to be derived. Core languages and possible languages would be the same only "under idealized conditions that are never realized in fact in the real world of heterogeneous speech communities. . . . Each actual 'language' will incorporate a periphery of borrowings, historical residues, inventions, and so on, which we can hardly expect to – and indeed would not want to – incorporate within a principled theory of UG." (Chomsky 1981: 8)

Chomsky (1980) argues convincingly that there is no case for natural languages being necessarily recursive. Nevertheless, languages might just *happen* to be recursive. Putnam (1961) gives three reasons he claims "point in this direction":

1. "Speakers can presumably classify sentences as acceptable or unacceptable, deviant or nondeviant, et cetera, without reliance on extra-linguistic contexts. There are of course exceptions to this rule...",
2. Grammaticality judgments can be made for nonsense sentences,
3. Grammars can be learned.

The first reason is most puzzling. The reference to "extra-linguistic context" is irrelevant; without it, reason (1) seems to be asserting that acceptability can be decided except where it cannot be. With respect to the second reason, the fact that grammaticality judgments could be made for *some* nonsense sentences in no way affects the question of whether they can be made for all *grammatical* sentences. Finally, languages could be learnable without being recursive, as it is possible that all the rules that need to be acquired could be on the basis of sentences for which the recognition procedure succeeds.

Peters and Ritchie (1973a) contains a suggestive but hardly conclusive case for contingent recursivity:

1. Every TG has an exponentially bounded cycling function, and thus generates only recursive languages,
2. Every natural language has a descriptively adequate TG, and
3. The complexity of languages investigated so far is typical of the class.

If learnability rather than recognizability is the defining characteristic of possible languages, no claim refuting a theory on the grounds that it allows difficult languages will bear any weight, unless it can also be shown that possible languages are in fact easier to recognize than the recognizability theory predicts them to be. However, our everyday experience with language understanding leads us to think that syntactic recognition is a computationally efficient process – an observation, of course, that is the basis for Marcus's claim (1980) that a large part of it can be done in linear time, if not in real time. How are we to reconcile this with the $O(g)$-results we have for most theories, where $g$ is at least quadratic?[12]

These intuitive conclusions are based on observations (1) of "everyday" sentences, (2) where some nonsyntactic processing is done in parallel, (3) by the human processor. Each of these points is important.

[12] It has already been pointed out that $O(g)$ results are upper bounds, and showing that a recognition problem, for example, is $O(g)$ does not mean that, for any language, it is necessary to reach the upper-bound. Better upper-bounds can be achieved by tighter proofs, not just by better algorithms.

Although recognition may appear to be done in real time for most sentences encountered day to day, the $O$-results are asymptotic worst-case measures. It is therefore essential to obtain measures of recognition times for a variety of strings of words, whether sentences or not, and especially see if there are short, difficult ones. There are at least two cases of interest here. The first is that of garden-path sentences such as *The horse raced past the barn fell* and *Have the students who failed the exam take the supplementary,* which are globally unambiguous but locally ambiguous. These appear to be psychologically difficult. Another case is that of sentences that, in most grammars, are ambiguous because of attachment choices, such as those discussed by Church and Patil (1982). Finding one parse of these sentences is easy, but finding them all may be exponentially difficult. Psychological measures show these sentences *not* to be difficult, suggesting that not all parses are constructed or that they can all be examined in parallel.

$O$-results depend on some underlying machine model, and most of the results known for language recognition have been obtained on RAMs. Can implementation changes improve things on relevant range? As mentioned above, the sequential models are all polynomially related, and no problem not having a polynomial time solution on a sequential machine is likely to have one on a parallel machine limited to at most a polynomial number of processors, at least if P is not equal to NP.

Both these results restrict the improvement one can obtain by changing implementation, but are of little use in comparing algorithms of low complexity. Berwick and Weinberg (1982) give examples of how algorithms of low complexity may have different implementations differing by large constant factors. In particular, changes in the form of the grammar and in its representation may have this effect.

It is well-known that implementation of machines with infinite storage on finite devices leads to a change in specification. A context-free parser implemented on a machine with finite memory will have a bounded stack and therefore recognize only finite-state languages. The language recognized by the implemented machine could therefore be recognized by another machine in linear time. Although one would rarely use this strategy as a design principle, a variant of it is more plausible: use a restriction of the general method for a subset of the inputs and revert to the general method when the special case fails. Marcus's parser (1980) with its bounded look-ahead is a good example. Sentences parsable within the allowed look-ahead have "quick" parses, but some grammatical sentences, such as "garden path" sentences cannot be recognized without an extension to the mechanism that would distort the complexity measures. A consequence of the possibility of implementation of this character is that observations of their operation ought to show "discontinuities" in the processing time, depending on whether an input is in or out of the restricted subset.

There is obviously much more of this story to be told. Allow me to speculate as to how it might go. We may end up with a space of linguistic theories, differing in the idealization of the data they assume, in the way they decompose constraints, and in the procedural specifications they postulate. I take it that two theories may differ in that the second simply provides more detail than the first as to how constraints specified by the first are to be used. Our observations, in particular our measurements of necessary resources, are drawn from the "ultimate implementation", but this does not mean that the "ultimately low-level theory" is necessarily the most informative, or that less procedural theories are not useful stepping stones to more procedural ones.

It is also not clear that theories of different computational power may not be useful as descriptions of different parts of the syntactic apparatus. For example, it may be easier to learn statements of constraints within the framework of a general machine. The constraints once learned might then be subjected to transformation to produce more efficient special-purpose processors also imposing resource limitations.

Whatever we decide to make of existing formal results, it is clear that continuing contact with the complexity community is important. The driving problems there are the $P = NP$ question, the determination of lower bounds, the study of time-space tradeoffs, and the complexity of parallel computations. We still have some methodological house-cleaning to do, but I don't see how we can avoid being affected by the outcome of their investigations.

## References


Aho, A.V. 1968 Indexed Grammars: An Extension of the Context-Free Grammars. *JACM* 15: 647-671.

Aho, A.V.; Hopcroft, J.E.; and Ullman, J.D. 1974 *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Massachusetts.

Aho, A.V. and Ullman, J.D. 1972 *The Theory of Parsing, Translation, and Compiling.* Prentice Hall, Englewood Cliffs, New Jersey.

Baker, B.S. 1972 Arbitrary Grammars Generating Context-Free Languages. Center for Research in Computing Technology, Harvard University.

Bar-Hillel, Y.; Perlis, M.; and Shamir, E. 1961 On Formal Properties of Simple Phrase Structure Grammars. *Z. Phonetik, Sprach. Komm.* 14: 143-172.

Berwick, R.C. and Weinberg, A. 1982 Parsing Efficiency, Computational Complexity, and the Evaluation of Grammatical Theories. *Linguistic Inquiry* 13: 165-191.

Berwick, R.C. 1982 Computational Complexity and Lexical Functional Grammar. *American Journal of Computational Linguistics* 8(3-4): 97-109.

Berwick, R.C. 1984 Strong Generative Capacity, Weak Generative Capacity and Modern Linguistic Theories. *CL* 10(3-4):, 189-203.

Borgida, A.T. 1983 Some Formal Results about Stratificational Grammars and their Relevance to Linguistics. *Math. Sys. Th.* 16: 29-56.

Bresnan, J.; Kaplan, R.M.; Peters, P.S.; and Zaenen, A. 1982 Cross-serial Dependencies in Dutch. *Ling. Inq.* 13.

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

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

Church, K. 1981 On Memory Limitations in Natural Language Processing. Master Th., M.I.T.

Church, K. 1983 A Finite-State Parser for Use in Speech Recognition. *Proceedings of 21st Annual Meeting of the ACL.* Cambridge, Massachusetts: 91-97.

Church, K. and Patil, R. 1982 Coping with Syntactic Ambiguity or How to Put the Block on the Table. *American Journal of Computational Linguistics* 8(3-4): 139-149.

Cook, S.A. 1981 Towards a Complexity Theory of Synchronous Parallel Computation. *L'Enseignement Mathématique* 27: 99-124.

Earley, J. 1970 An Efficient Context-Free Parsing Algorithm. *Communications of ACM* 13: 94-102.

Gazdar, G. 1982 Phrase Structure Grammar. In Jacobson, P. and Pullum, G., Eds., *The Nature of Syntactic Representation.* Reidel, Dordrecht.

Gazdar, G. and Pullum, G. 1982 Generalized Phrase Structure Grammar: A Theoretical Synopsis. Indiana Univ. Linguistic Club.

Gleason, H.A. Jr. 1964 The Organization of Language: a Stratificational View. In *Monograph Series on Language and Linguistics, no. 21.* Georgetown University Press, Washington.

Graham, S.L.; Harrison, M.A.; and Ruzzo, W.L. 1980 An Improved Context-Free Recognizer. *ACM Trans. on Prog. Lang. and Systems* 2: 415-462.

Greibach, S.A. 1974 Some Restrictions on W-Grammars. Int. J. of Comp. and Info. Sc. 3: 415-462.

Hays, D.G. 1962 *Automatic Language Data Processing.* Prentice Hall, Englewood Cliffs, New Jersey.

Higginbotham, J. 1984 English Is Not a Context-Free Language. *Ling. Inq.* 15: 225-234.

Hintikka, J.K.K. 1977 Quantifiers in Natural Language: Some Logical Problems. II. *Linguistics and Philosophy* 2: 153-172.

Hopcroft, J.E. and Ullman, J. 1979 *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, Reading, Massachusetts.

Joshi, A.K. 1983 Factoring Recursion and Dependencies: an Aspect of Tree Adjoining Grammars and a Comparison of Some Formal Properties of TAGs. *Proceedings of 21st Annual Meeting of the ACL.* Cambridge, Massachusetts: 7-15.

Joshi, A.K. 1984 How Much Context-Sensitivity Is Required to Provide Reasonable Structual Descriptions: Tree Adjoining Grammars. In *Natural Language Processing: Psycholinguistic, Computational and Theoretical Properties.* Cambridge University Press, New York, New York.

Joshi, A.K. and Levy, L.S. 1977 Constraints on Structural Descriptions: Local Transformation. *SIAM J. on Computing* 6: 272-284.

Joshi, A.K. and Levy, L.S. 1982 Phrase Structure Trees Bear More Fruit than You Would Have Thought. *American Journal of Computational Linguistics* 8(1): 1-11.

Joshi, A.K.; Levy, L.S.; and Takahashi, M. 1975 Tree Adjunct Grammars. *J. Comp. and Sys. Sc.* 10: 136-163.

Joshi, A.K.; Levy, L.S.; and Yueh, K. 1980 Local Constraints on Programming Languages, Part 1: Syntax. *Th. Comp. Sc.* 12: 265-290.

Kaplan R. and Bresnan, J. 1982 Lexical-Functional Grammar: a Formal System for Grammatical Representation. In Bresnan, J., Ed., *The Mental Representation of Grammatical Relations.* MIT Press, Cambridge, Massachusetts: 173-281.

Kasami, T. 1965 An Efficient Recognition and Syntax Algorithm for Context-Free Languages. AF-CRL-65-758, Air Force Cambridge Research Laboratory, Bedford, Massachusetts.

Kay, M. 1983 When Meta-Rules Are Not Meta-Rules. In Sparck-Jones, K. and Wilks, Y., Eds., *Automatic Natural Language Parsing.* John Wiley, New York.

Koskenniemi, K. 1983 Two-Level Model for Morphological Analysis. Ph.D. Th., Univ. of Helsinki.

Ladusaw, W. 1979 Polarity Sensitivity as Inherent Scope Relations. Ph.D. Th., University of Texas at Austin.

Ladusaw, W. 1983 Logical Forms and Conditions on Grammaticality. *Ling. and Phil.* 6: 373-392.

Lamb, S. 1966 *Outline of Stratificational Grammar.* Georgetown University Press, Washington, DC.

Langendoen, D.T. 1975 Finite-State Parsing of Phrase-Structure Languages and the Status of Readjustment Rules in Grammar. *Ling. Inq.* 6(4): 533-554.

Langendoen, D.T. and Langsam, Y. 1984 The Representation of Constituent Structures for Finite-State Parsing. *Proceedings of COLING-84.* Stanford, California: 24-27.

LaPointe, S. 1977 Recursiveness and Deletion. *Ling. Anal.* 3: 227-265.

Linebarger, M. 1980 The Grammar of Negative Polarity. Ph.D. Th., MIT.

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

Montague, R. 1973 The Proper Treatment of Quantification in Ordinary English. In Hintikka, J.K.K.; Moravcsik, J.; and Suppes, P., Eds., *Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics.* Reidel, Dordrecht: 221-242.

Osherśon, D.N.; Stob, M.; and Weinstein, S. 1983 Formal Theories of Language Acquisition: Practical and Theoretical Perspectives. *Proceedings of IJCAI-83:* 566-572.

Pereira, F.C.N. and Shieber, S. 1984 The Semantics of Grammar Formalisms Seen as Computer Languages. *Proceedings of COLING-84:* 123-129.

Peters, P.S. and Ritchie, R.W. 1969 A Note on the Universal Base Hypothesis. *Ling. and Phil.* 5: 150-152.

Peters, P.S. and Ritchie, R.W. 1971 On Restricting the Base Component of Transformational Grammars. *Inf. and Control* 18: 483-501.

Peters, P.S. and Ritchie, R.W. 1973a On the Generative Power of Transformational Grammars. *Inf. Sc.* 6: 49-83.

Peters, P.S. and Ritchie, R.W. 1973b Context-Sensitive Immediate Constituent Analysis – Context-Free Languages. *Math. Sys. Theory* 6: 324-333.

Peters, P.S. and Ritchie, R.W. 1973c Non-Filtering and Local Filtering Grammars. In Hintikka, J.K.K.; Moravcsik, J.; and Suppes, P., Eds., *Approaches to Natural Language.* Reidel, Dordrecht: 180-194.

Petrick, S.R. 1965 Recognition Procedure for Transformational Grammars. Ph.D. Th., MIT.

Postal, P.M. 1964a Limitations of phrase-structure grammars. In *The Structure of Language: Readings in the Philosophy of Language.* Prentice Hall, Engĺewood Cliffs, New Jersey: 137-151.

Postal, P.M. 1964b Constituent Structure: A Study of Contemporary Models of Syntactic Structure. *Int. J. of Amer. Ling.* 3.

Postal, P.M. and Langendoen, D.T. 1984 English and the Class of Context-Free Languages. *CL* 10(3-4): 177-181.

Pullum, G.K. 1984 On Two Recent Attempts to Show that English Is Not a CFL. *CL* 10(3-4): 182-186.

Pullum, G.K. and Gazdar, G. 1982 Natural and Context-Free Languages. *Ling. and Phil.* 4: 471-504.

Putnam, H. 1961 Some Issues in the Theory of Grammar. *Proceedings, American Math. Soc.*

Rounds, W.C. 1970a Mappings and Grammars on Trees. *Math. Sys. Th.* 4(3): 257-287.

Rounds, W.C. 1970b Tree-Oriented Proofs of Some Theorems on Context-Free and Indexed Languages. *Second Symp. on Th. Comp. Sc., ACM:* 109-116.

Rounds, W.C. 1973 Complexity of Recognition in Intermediate-Level Languages. *Symp. on Sw. & Aut. Th., IEEE:* 145-158.

Rounds, W.C. 1975 A Grammatical Characterization of Exponential-Time Language. *Symp. on Found. of Comp. Sc., IEEE:* 135-143.

Ruzzo, W.L. 1979 Uniform Circuit Complexity. *Proceedings of 20th Annual ACM Symp. on Found. of Comp. Sc.:* 312-318.

Shieber, S.M. 1984 Evidence Against the Context-Freeness of Natural Language. TN-330, SRI International, Menlo Park, California. To appear in *Linguistics and Philosophy.*

Shieber, S.M.; Stucky, S.U.; Uszkoreit, H.; and Robinson, J.J. 1983 Formal Constraints on Metarules. *Proceedings of 21st Annual Meeting of the ACL.* Cambridge, Massachusetts: 22-27.

Slocum, J. 1981 A Practical Comparison of Parsing Strategies. *Proceedings of the 19th Annual Meeting of the ACL.* Stanford, California: 1-6.

Uszkoreit, H. and Peters, P.S. 1983 Essential Variables in Metarules. Technical Note 305, SRI International, Menlo Park, California.

Valiant, L. 1975 General Context-Free Recognition in Less Than Cubic Time. *J. Comp. and Sys. Sc.* 10: 308-315.

Van Wijngaarden, A. 1969 Report on the Algorithmic Language ALGOL 68. *Numerische Mathematik* 14: 79-218.

Warren, D.S. 1979 Syntax and Semantics of Parsing: An Application to Montague Grammar. Ph.D. Th., University of Michigan.

Yokomori, T. and Joshi, A.K. to appear Semi-linearity, Parikh-boundedness and Tree Adjunct Languages. *Inf. Pr. Letters.*

Younger, D.H. 1967 Recognition and Parsing of Context-Free Languages in Time $n^3$. *Inf. and Control* 14: 189-208.