# THEORETICAL LINGUISTICS

Vol. 17 1991 No. 1/2/3

Offprint

WALTER DE GRUYTER W DE G BERLIN · NEW YORK

ROBERT C. BERWICK

# COMPUTATIONAL COMPLEXITY THEORY AND NATURAL LANGUAGE: A PARADOX RESOLVED

This article surveys currently known results on the application of computational complexity theory to natural language theories and the practical implications of these results. Because they contain rich feature systems and variables, all current descriptively adequate linguistic theories evidently embed computationally intractable problems, in the class *NP* (Nondeterministic Polynomial time) or worse. We can resolve this apparent complexity paradox by noting that natural grammars contain substantive constraints on complexity-producing processess like the number of agreement features, or that the average-case problems encountered in practice are truncated just enough to make processing efficient by a deterministic computational device. Taken together, this evidence suggests that natural languages are constrained to avoid the intractability of *NP* problems and that people do not have the unlimited computational resources to process languages as if they were nondeterministic or fully parallel computers.

## 1 Introduction: complexity theory as an analytical tool for natural language

In computer science modern computational complexity theory has become one of the most powerful techniques known for analyzing the inherent structural complexity of information-processing problems. By posing linguistic analyses as such problems we can apply the same techniques to linguistic theories as well. Complexity theory provides a unique approach to analyzing the structure of linguistic models in a theory- and algorithm-independent way. This neutrality is an enormous advantage, since it is often quite difficult to see how to compare quite different linguistic theories in terms of weak generative capacity (the languages they generate) – two theories may often diverge widely in what linguistic phenomena they cover and what component (syntax, semantics, etc.) they ascribe a particular phenomenon to. In this paper we show how complexity analysis can sidestep the ofen contentious issue of syntax vs. semantics. Then too, since we know relatively little about the algorithms the brain might use to compute linguistic analyses, an algorithm-independent approach would seem to be an appropriate conservative research program.

This paper surveys what is known about the computational complexity of many current linguistic theories, shows why all such theories seem to have roughly the same high degree of complexity, and then examines the *practical* significance of these intractability results – the potential difference between the formal *computational* complexity of linguistic theories and the observed *cognitive* complexity of actual language processing. A *priori* there are three possible comparative outcomes: either (1) the computational complexity of a linguistic formalism matches observed cognitive complexity; or (2) computational complexity is greater than observed cognitive complexity; (3) cognitive complexity exceeds the computational complexity of the theory on the same linguistic phenomena.

In case (1), the complexity analysis fully accounts for observed cognitive complexity; in case (2), further restrictions must be forthcoming: either the linguistic theory must be restricted (by, e.g., substantive constraints) or else the range of inputs the cognitive system must handle is restricted (possibly truncated) in some way, put another way, there must be a substantive theory of linguistic performance; in case (3), the theory is descriptively inadequate. Note that in each case the complexity analysis can tell us something, irrespective of how the results turn out. Thus, our aim in doing computational complexity analysis is *not* to show that one theory is better or worse than another – often the aim of a weak generative capacity analysis – but rather to diagnose in an invariant way the sources of complexity in all linguistic theories.

A second aim of this paper is to resolve a basic paradox mentioned in the previous paragraph: most linguistic theories are computationally intractable, more precisely, at least $\mathcal{NP}$-hard (solvable only in Non-deterministic Polynomial time or worse, with no non-exponential time algorithms known). But at a minimum, modulo the three cases above, we would expect natural language processing to be polynomial-time computable, running in time proportional to $n^j$ for some integer $j$ fixed in advance; in fact, perhaps, linear time (with $j = 1$) or, even faster, real time (allowing only a bounded number of processing steps between each input token read, a condition that can be violated by linear time processing).[1] Indeed, given the number of results in hand (see table I below), it now seems reasonable to put forth as a general thesis, as Ristad (1990) does, that any adequate linguistic theory is bounded *from below* by the class $NP$. Let us call this the $NP$ *thesis*.

How can this paradox be resolved? $NP$-hard problems are considered computationally intractable, yet people evidently solve these problems every day even as we speak – just as computer scientists must solve classic $NP$-hard problems like $k$-GRAPH COLORABILITY – a generalization of the 4-color map problem – all the time. Therefore, we need constraints that will make these $NP$-hard problems tractable. To understand what constraints we need, we must carefully examine the *source* of computational intractibility in cases where it arises. This is just a detailed analysis of case (2) described above: when the complexity analysis of a language model does not mesh with observed performance, then something substantive remains to be said, beyond what the model claims.

We shall see that by examining more carefully the complexity of intractable problems like propositional satisfiability (SAT) or $k$-GRAPH COLORABILITY and how they relate to natural languages, one can determine that features or variables begin to cause processing difficulty as soon as their number exceeds 4 or 5. This is quite striking, because the number of

---

[1] Indeed, some current linguistic models – GPSG (as defined in Gazdar, Klein, Pullum, and Sag (1985); any model that includes vp-ellipsis by copy-and-bind lambda abstraction are $PSPACE$-hard (polynomial space hard) or worse. While at first glance this seems incompatible with such results as the polynomial-time parsability of, say, tree adjoining grammar (TAGs) (Vijay-Shanker and Joshi (1985)) we note that the TAG results generally omit the complexity effects of feature-checking. Following Ristad and Berwick (1989) we can show *any* formalism embedding agreement features is at least $NP$-hard; see also Schabes and Joshi (1988) which raises the issue of augmenting tags with feature-checking unification machinery but does not discuss its complexity-theoretic implications.

[2] It is also possible to argue (as Ristad (1990) does) that any adequate linguistic theory ought to be bounded *above* by the class $NP$, e. g., ought *not* to be in the class $PSPACE$. Why? Ristad's reasoning is to note that $NP$ problems have efficient *witnesses* (solutions that can be rapidly verified) but $PSPACE$ problems do not (see section 2.1). If we assume that witness examples always exist – any linguistic output implies that an (ideal, competence-based) speaker started from *some* witness to the parsing problem, then $PSPACE$ systems are ruled out. Not however that there are auxiliary assumptions that call this argument into question in the usual way: the relationship between competence and performance models, for one, suggests that there is no *necessary* reason that the speaker, even an ideal one, ought to have in hand an efficient witness.

syntactic agreement features across the world's languages, and within a given language does not exceed this number (being restricted to person, number, gender, exclusively, dual; here we must assume, e. g., that kinship terms or Bantu long/short classifiers do not do matching and are arguably like color-term systems, not an agreement system at all). Similarly, the number of independent vowel harmony processes in any natural language evidently does not exceed three. Evidence like this *supports* the *NP* thesis and the complexity analysis approach, because the complexity analysis is assumed in order to explain otherwise mysterious facts about natural language. Thus, contrary to what has sometimes been asserted (e. g., in Koskenniemi and Church (1988)), the observation that natural languages are restricted so as to sidestep *NP* problems actually buttresses, not refutes, the *NP* thesis.

A third aim of this paper is to suggest how computational complexity can be used to probe the notion of *modularity* in linguistic theories. For example, we shall see that the intractability of autosegmental phonology arises from allowing variables (features) to interact across distinct autosegmental tiers. Thus the autonomy promised by autosegmental theory may not be so innocently autonomous at all.

Note that many current linguistic models, e. g., the *principles and parameters* model of Chomsky (1986), are generally *not* modular in the computer science sense of that term even though it's commonly said that they are: they violate the notions of context-independence and non-interference because principles are *designed* to interact with one another (large deductive consequences are to follow from a small set of modules, or, as Fodor (1983: 27) notes, "some capacities surely arise from the *interaction* of underlying causes, in fact, the more of these, the merrier for the theorist, since his goal is to get the maximum amount of psychological explanation out of the smallest possible inventory of postulated causal mechanisms."[3]

Finally, it may be that restricting ourselves to the problems that occur *in practice* make the *average* complexity quite better than the worst-case analysis. This point is sometimes raised as an objection to complexity analysis generally. While SAT and $k$-GRAPH COLORABILITY are hard in the worst case, it is widely believed that such hard examples are quite sparse, hence unlikely to arise in practice. We shall see that the emerging theory of average-case complexity provides some insight into this question, since the break-point for difficulty again comes at $k = 4$ or 5. We can then relate this

[3] This observation and citation is due to S. Fong (personal communication).

result back to the linguistic problem of pronoun antecedence, which is modeled by *k*-GRAPH COLORABILITY: Berwick (1989) shows that if the number of antecedents (the parameter *k*) is fixed, then there *is* a polynomial time algorithm for disjoint reference, as in the sentence

(1) John told Bill that he should leave

where *he* may be coreferential with either *John, Bill,* or some arbitrary person (drawn from the discourse). The average-case complexity analysis tells us that as soon as the number of antecedents exceeds 4 or 5, then determining pronoun antecedence becomes computationally intractable (under this set of assumptions).[4] Much foundational work remains to be done with average-case analysis however, since average-case complexity theory naturally depends on as yet unexplored distributional assumptions about linguistic constructions. (See also Kasper and Rounds (1990) for a related view on the complexity of unification in language.)

In any case, the existence of such constraints in natural languages, which seem designed to sidestep the computational intractabilities of *NP* problems, provides additional evidence that human cognitive processors do not have sufficient computational resources at their command to simulate general nondeterministic polynomial time computations, say by unlimited backtracking or unlimited parallelism. Rather, this kind of search process is severely limited.

The rest of this article is organized as follows. Section 2 briefly reviews the terminology and foundations of complexity-based accounts of linguistic processing. Section 2.1 outlines the major properties of the computational complexity probe used: the fundamental distinction between deterministic and nondeterministic Turing machine computations, the method of problem reduction, the key litmus test problem of 3-SATISFIABILITY (3-SAT), a simple example of the complexity of pronoun binding as related to the problem of *k*-GRAPH COLORABILITY. Section 2.2 summarizes most of the known complexity-theoretic classifications of linguistic problems and discusses Ristad's *NP* thesis.

Section 3 turns to a case study of the computational complexity of a different linguistic phenomena: nonlinear (autosegmental) phonology. (The result was first presented in Berwick (1986) at the University of

[4] For alternative formulations of the pronoun antecedence problem that make other assumptions and obtain somewhat different results, see Giorgi, Pianesi, and Satta (1990).

Pennsylvania, and substantially refined by Ristad (1990).) The case study shows that the autosegmental formalism can encode a computationally intractable problem, but, more importantly, it pinpoints *where* the intractability comes from: an unexpected nonmodularity in the linguistic representation. Given this diagnosis, Section 4 shows how to resolve the apparent paradox of computational intractability in linguistic systems, showing how either constraints on linguistic features and nonmodular components or input truncation can yield efficient (polynomial time) language processing.

## 2 Complexity theory: basic terminology and known results about natural language

Computational complexity theory measures the intrinsic difficulty of solving a problem no matter how its solution is obtained – it abstracts away from algorithms and machine implementation details altogether. This section reviews the basic terminology and methods of complexity analysis, and then turns to a survey of known results about the computational complexity of current linguistic models.

### 2.1 *Complexity theory: basic terminology and methods*

Let us review the key features of computational complexity theory as they pertain to linguistic analysis; for further details, refer to Barton, Berwick, and Ristad (1987) or Garey and Johnson (1979).

Computational complexity theory classifies *problems*, e.g., the mapping from phonetic form (PF) or an input sentence to a parse tree or S-structure, according to the amount of time needed to solve these problems on some abstract computer model, conventionally a deterministic Turing machine (TM). Recall that a TM has three parts, a *finite-state control* that is effectively the program of the machine; an *input tape* bounded on the left but extending arbitrarily far to the right, divided into some number of discrete *cells* or *tape squares*; and a *read/write tape head* that looks at one tape cell at a time, and, given what the finite-state control says, in a single move (i) changes state, (ii) possibly writes one of a finite number of symbols on the tape cell it is currently scanning; and (iii) moves its read/write head one square right or left (a *step* or *move* of the TM). The TM starts with the input written on the leftmost *n* cells and scans the very leftmost cell. Problems are encoded on the TM's input tape, in some standard fashion that does not obscure the complexity of the original problem.[5]

Crucially, the TM is *deterministic*: at each step, given the symbol that the read/write head is scanning, the finite-state control dictates that there is only *one* possible move the TM can make. A TM is *nondeterministic* if there is more than one such move. We shall see that this distinction between deterministic and nondeterministic TMs plays a key role in what follows.

We use deterministic TMs to measure the computational resources, generally time or space, used by algorithms in the following way. Intuitively, we measure time by counting the number of moves and space by counting the number of tape squares used during a computation, considering the worst possible case. More precisely:

**Definition 1:** Given some TM $M$, if for every input of length $n$, denoted $|n|$, $M$ makes at most $T(n)$ moves before halting, then $M$ is of *time complexity* $T(n)$. (If $M$ does not halt its time complexity is either undefined or infinite.)
**Definition 2:** Given some TM $M$, if for every input of length $n$, $M$ scans at most $S(n)$ tape squares before halting, then $M$ is of *space complexity* $S(n)$. (If $M$ does not halt its space complexity is either undefined or infinite.) The time (resp. space) complexity of a problem is the minimum value of $T(n)$ (resp. $S(n)$) as we range over all possible TM programs.

In contrast to deterministic TMs, the time or space complexity of a nondeterministic TM is calculated in a slightly different way. We may imagine the computation sequence of a nondeterministic TM as a branching *or*-tree: starting at the root of the tree, at each step we may have some finite number of possible next moves to make; then, for each of these possibilities,

[5] For example, we might code a problem about graphs by encoding the graph as a matrix of 1's and 0's, written out as a set of 1-dimensional vectors. Typically the problem is coded so that it becomes a *decision problem* with a yes/no answer rather than actually outputting the solution to the problem, but this ordinarily causes no change in the complexity classification of a problem. Problems also cannot be encoded in some baroque way that pads out their length so greatly that the complexity of the problem becomes easier than it should be. For example, if a problem takes exponential time as a function of its input length, time $2^{|n|}$, then we could simply puff up the input length with dummy symbols to that size, ignore those dummy symbols, and the time complexity would become a linear function of input length. For this reason, encodings are usually restricted to be "reasonable," meaning at most polynomially larger than the original problem statement. The discussion in the next makes clear why this is a reasonable definition of "reasonable"; see Garey and Johnson (1979) for additional discussion.

there may be further branches, and so forth. Some of these possible paths will succeed in finding a solution to the problem, some may not, and some paths may never halt. The time used by such a nondeterministic TM is the *shortest* such path that leads to a successful solution, if one exists. Thus the failing paths do not count in the complexity analysis of a nondeterministic TM. Intuitively, given some fixed input $n$, the machine can "guess" possible solutions for free. We shall see below how this guessing power enters into the fundamental distinction between computationally tractable and intractable problems.

How does complexity theory tie this TM model to computational problems and linguistic analysis? Complexity theory can play a particularly valuable role. First of all, computational complexity theory studies the inherent complexity of problem structure – strong generative capacity in the case of linguistic theories since the grammar is typically included in problem statements. We include the grammar in the problem statement because we want the complexity analysis to tell us something about the internal structure of linguistic theories, not just languages, so that we can discover where a theory is too complex. In other words, even though any single natural language might be described by just one grammar drawn from a grammatical framework, in order to study the complexity of the framework and not just single grammars we will often explicitly include the grammar in the complexity analysis. In Barton, Berwick, and Ristad (1987) this approach was dubbed the *Universal Recognition Problem* (URP) as opposed to *Fixed Recognition Problem* (FRP), but it is probably more apt to call it an *intensional* or *grammatical framework analysis* as opposed to an *extensional* or *language analysis.* (The distinction between a strong generative capacity analysis and a weak generative capacity analysis also comes to mind.)

This tack is in keeping with the spirit of standard computational complexity theory that incorporates in the problem statement (hence in the encoded input $n$) the important parameters of interest, for to exclude these parameters is to eliminate them from complexity analysis altogether, and we certainly wish to analyze the complexity of entire grammatical theories, not just languages. Fixing the grammar generally allows precomputations and optimizations that, while important for special-case processing (as we shall discuss in section 4 below), often hide the real complexity of a problem. For example, in order to analyze the complexity of games like chess or go, complexity theorists standardly generalize board size to $n \times n$, because otherwise, for a fixed board size, akin to a fixed grammar, the complexity is essentially zero: one can simply compute all the possible moves in advance and look up answers in a table in effectively constant time.

Second, complexity classifications are invariant across a wide range of primitive machine models, representations, algorithms, and actual implementations. We can contrast this approach to those that posit a particular algorithm, machine implementation combination, such as Marcus (1980) or Berwick and Weinberg (1984). These analyses depend on assuming that, for example, the language processor uses a certain kind of language processor, one with a pushdown stack machine and a lookahead buffer. The results change if the machine changes. In contrast, one can show that complexity classifications are stable under fairly broad changes to the underlying computer model, e. g., if we used a more "reasonable" machine with a random access memory instead of a lumbering TM, or even any physically realizable parallel computer, nothing substantial would change in the analyses that follow.

It is important to see how powerful this invariance is. Any change in the problem representation that preserves the essential features of the original representation – preserving solutions to the original problem, in effect its descriptive adequacy – can have no effect on its complexity classification as efficiently or inefficiently solvable. This robustness makes complexity theory ideally suited for studying cognition. While we do know something about the abstract computational problems the brain solves, we know correspondingly little about the algorithms and hardware involved.

This invariance also lets us sidestep another common difficulty with formal analyses of linguistic theories. Note that the complexity of a problem abstracts away from the particular algorithm or implementation used to solve it, since we range over all algorithms (and assume invariance over computer models). Thus the complexity analysis holds over representational shifts. This is a big advantage if one does not know whether a particular phenomenon is to be captured in "syntax" or "semantics" or in any other component of a grammatical system, because the complexity analysis does not make any assumptions about this kind of division either. Thus, if a problem is computationally intractable, it is hard no matter what (for almost all purposes). The results cannot be dismissed by simply appealing to another component of the grammar. We discuss this point further immediately below, after introducing the method of problem reduction.

We next briefly review the basic categorizations obtained by computational complexity theory and its key division of problems into *tractable* and *intractable* classes; see figure 1. *P* is the class of problems solvable

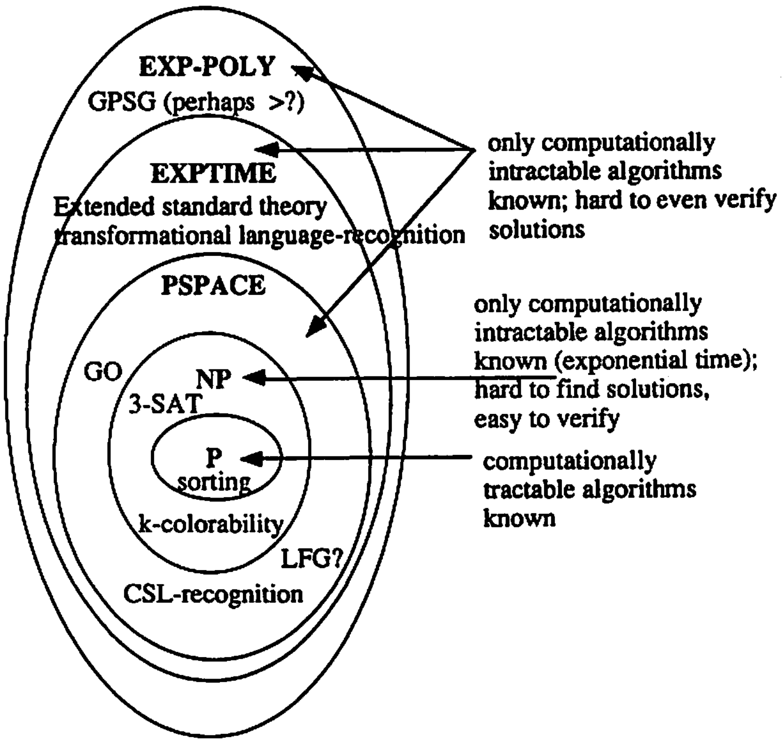
EXP-POLY
GPSG (perhaps >?)
EXPTIME
Extended standard theory
transformational language-recognition
PSPACE
GO
NP
3-SAT
P
sorting
k-colorability
LFG?
CSL-recognition
only computationally intractable algorithms known; hard to even verify solutions
only computationally intractable algorithms known (exponential time); hard to find solutions, easy to verify
computationally tractable algorithms known

Fig. 1. The standard hierarchy of time and space classes. All containments are assumed proper, under the usual assumptions that $P \neq NP$, etc. No known deterministic polynomial time solution algorithms are known for problems in *NP*, while problems in *P* are known to have deterministic polynomial time solution algorithms. Some representative problems in each class are named, including some linguistic examples. For example, context-sensitive language recognition is in *PSPACE*, while the older extended standard transformational theory, under suitable restrictions, described exactly the languages recognizable in expontential time, or *EXPTIME*.

in Polynomial time on a *deterministic* Turing machine (TM), that is, in the worst case, time $n^j$ for some integer $j$ where $n$ as before is the length of an encoding of the problem to be solved. Such problems are considered to be computationally tractable. For example, sorting a list of $n$ names takes time $n \log n$ in the worst case by a variety of algorithms, and so is efficiently solvable (in *P*).

*NP* is the class of problems solvable in *N*ondeterministic *P*olynomial time. Such problems have only known exponential time solution algorithms and are dubbed computationally intractable. Informally, a problem is in this class if one can guess the answer to the problem (that is the nondeterministic part) and then verify its correctness rapidly, in polynominal time. In other words, one of the computation sequences leading to a solution is of polynomial length. Put another way, we say then that *NP* problems have efficient *witnesses*, namely, the answers to the problems that can be rapidly verified.

Some problems have no known algorithms for their solution in *P* but are easily seen to be in *NP*. For example, the problem of deciding whether a whole number *i* is composite is in *NP* because it can be solved by guessing a pair of potential divisors and then quickly checking if their product equals *i*. (This is the efficient witness for the problem.) But the only known algorithms for finding composites on real computers take at least time exponential in the worst case, proportional to $2^n$. Obviously the class *P* is contained in *NP*. But it is strongly hypothesized that this containment is proper because there are no efficient algorithms known for a wide variety of problems in *NP*, and because of the techniques of problem reduction, described below, demonstrating that *if* certain problems in *NP* were efficiently solvable *then* all problems in *NP* would be efficiently solvable (see figure 3). But no such efficient algorithms are known, hence the suspected proper containment of *P* in *NP* (figure 1).

Following standard definitions, a problem is called *NP-hard* if it is at least as hard as any problem in *NP*. A problem is *NP-complete* if it is both in *NP* and *NP*-hard. In effect then, an *NP*-hard problem serves as a proxy for the entire class *NP*. An *instance* of a problem is just a particular example of a problem with all its parameters filled in – for example, an instance of the sorting problem is some particular list of, say, twenty names to alphabetize. We give more precise definitions of these concepts just below.

The other complexity classes illustrated in figure 1 are presumably even broader, as shown, and include *PSPACE*, *EXPTIME* and *EXP-POLY*:

The class of problems solvable in polynominal space (context-sensitive language restriction is *PSPACE*-complete); exponential time; and exponential-polynomial time (the union over all time classes $d^{f(n)}$ where *d* is some constant and $f(n)$ is some polynomial). Again, it is widely assumed that *NP* is properly contained in *PSPACE*, which is in turn assumed to be properly contained in *EXPTIME*, and in turn *EXP-POLY* (see figure 1). Note that a problem in PSPACE cannot even be verified in polynomial time – that is, even if we have its solution in hand, it will take more than

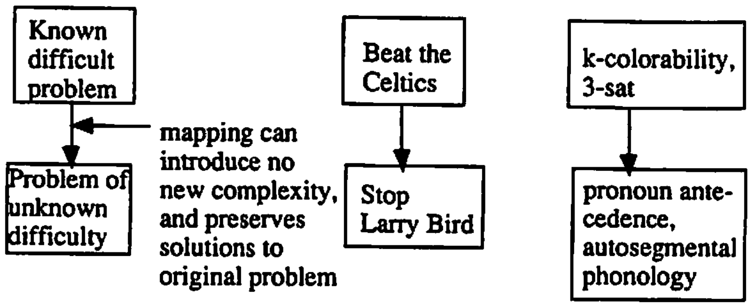
Known difficult problem
Problem of unknown difficulty
mapping can introduce no new complexity, and preserves solutions to original problem
Beat the Celtics
Stop Larry Bird
k-colorability, 3-sat
pronoun antecedence, autosegmental phonology

Fig. 2. Computational complexity theory shows that one problem is as hard as another by using the method of reduction. If a fast algorithm were discovered for solving the problem $S$, then we could now solve problem $T$ rapidly by (rapidly) transforming instances of $T$ to instances of $S$ and solving $S$. Clearly, the transformation must preserve solutions to the original problem.

polynomial time to check it. $PSPACE$ problems do not have (known) efficient witnesses.

Complexity classifications are established mainly via the proof technique of *reduction* (figure 2). A reduction converts instances of a problem $T$ of known complexity into instances of a problem $S$ whose complexity we wish to determine. The reduction must operate in polynomial time or less in order for it to introduce no spurious complexity, and it must preserve solutions to the original problem. Note how the logic of reduction works: *if* we had a deterministic polynomial-time algorithm for solving $S$, given the reduction *then* we could now solve $T$ in polynomial time, using $S$ as a subroutine, simply by converting instances of $T$ into instances of $S$, and then solving $S$ rapidly (note that this relies on the fact that functional composition preserves polynomial time, i.e., if $f$ and $g$ are polynomial time then so is $f \circ g$ because $n^j \cdot n^k = n^{j+k}$). If $T$ is $NP$-hard, this leads to a contradiction; in this case $S$ must be at least as hard as $T$. We can now redefine the notion of a problem being *hard* or *complete* in a class $C$ in terms of reducibility:

**Definition 3:** A problem $T$ is *hard for* $C$ if every problem in $C$ is polynomialtime reducible to $T$.
**Definition 4:** If a problem $T$ is hard for $C$ and in addition $T \in C$, then $T$ is *complete* for $C$.

To take a more nonmathematical and properly Bostonian analogy, suppose we know that beating the Celtics is intractable, and that every

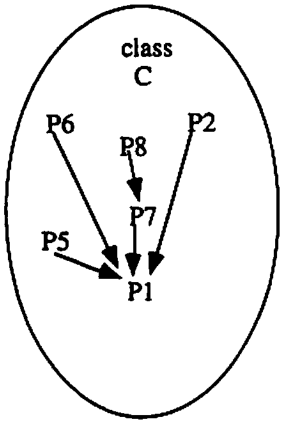

Fig. 3. The method of reduction can show that some problems are *complete* for a complexity class. Here, all problems in the class *C* can be quickly transformed into instances of problem *P* 1, so as not to take them outside the original class *C*. *P* 1 is therefore the *hardest* problem in *C*.

instance of beating the Celtics may be transformed into an instance of stopping Larry Bird (the example is a bit dated but something mysterious stays the hand from updating it, to say, the Chicago Bulls and Michael Jordan). Then stopping Bird must be at least as hard as beating the Celtics, for it stopping Bird were easy, then we could also beat the Celtics easily, a contradiction, by stopping Bird. (Note the transformation cannot introduce any spurious complexity, say, triple-teaming).

**Example.** The problem of determining whether there exists a satisfying assignment to the variables of a Boolean formula in 3-conjunctive normal form, i. e., an assignment of *true* or *false* to the Boolean variables such that a conjunction of 3-literal (negated or unnegated variable) disjuncts evaluates to *true*, is *NP*-complete (3-SAT). Here is an istance of such a problem:

$$(\bar{x} \vee y \vee \bar{z}) \wedge (y \vee z \vee u) \wedge (x \vee z \vee \bar{u}) \wedge (\bar{x} \vee y \vee u)$$

In effect, 3-SAT can simulate *any* nondeterministic polynomial time computation. Informally, it can be difficult to figure the solution to a 3-SAT problem out without guessing; the best one can do is to explore every combinatorial possibility, taking exponential time in the worst case. The reason is that the variables on the surface give no clues as to whether they should have the values *true* or *false*. In addition, variable assignments must be globally consistent; they interact, like a jigsaw puzzle: by choosing $x$ to be *true* in the first clause, it must be *false* in the third, so then either $z$ or $\bar{u}$ must be assigned *true*. (On the other hand, as section 4 shows, these "hard" 3-SAT problems are remarkably sparse and so unlikely to turn up "in practice", a point of some importance examined in section 4.)

**Example.** The problem of determining whether a map with $n$ countries can be colored with $k$ colors such that no two adjacent countries have the same color ($k$-COLORABILITY) is $NP$-complete (see also figure 4 for a linguistic example that can be related to this problem).

**A linguistics example: pronoun binding and obviation.** With this terminology in hand, we can give an example of a linguistic problem that is $NP$-hard and what the invariance of complexity theory means for the question of syntax and semantics. The problem is that of INTRASENTENTIAL PRONOUN ANTECEDENCE. An instance of the problem is a sentence of English, with $k$ possible antecedents (names), and $n$ pronouns; the problem is to determine a linking between the pronouns and names that satisfies the conditions known to govern linking, namely, that a pronoun and an antecedent cannot be too close (generally within the same S(entence) domain), or what is sometimes called *local obviation*; and that the antecedent and pronoun must agree in person, number, and gender. This problem can be shown to be $NP$-hard (Ristad (1990)) by a straightforward reduction from $k$-graph colorability, where the names are the colors and the pronouns the countries; note that the obviation and agreement conditions induce a graph that corresponds to a $k$-colorability problem, as with this example sentence:

(2) Before Bill$_a$, Tom$_b$, and Jack$_c$ were friends,
[he$_1$ wanted him$_2$ to introduce him$_3$ to him$_4$]

In this case the colors are the three names and the vertices of the graph are the four pronouns; it is easy to see that sentence (2) induces the graph in figure 4. Note that introspectively at least, determining the linking relations is cognitively difficult and becomes rapidly more difficult as $k$ and $n$ exceed 3 or 4.

In the linguistic theory considered by Ristad, the linking constraints are carried out at a level of logical form (LF), a modest transduction of syntactic structure, and these are syntactic constraints. In contrast, other linguistic theories (e. g., the one proposed in Gazdar, Klein, Pullum, and Sag (1985)) do not state constraints such as obviation at a syntactic level, but

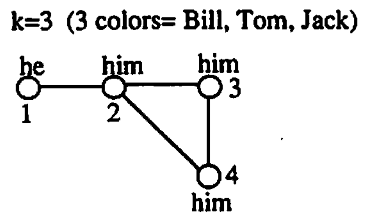

Fig. 4. A sentence with pronouns and antecedents has an associated graph coloring problem. The example is from Ristad (1990).

rather leave such constraints "for semantics." It is here that the power of a complexity analysis comes to the fore. We might consider the "linking" of *he* and the three *hims* to be carried out by some other component in some other theory – say by a theory that does not even use a level of LF. Can this make any difference about the difficulty of the pronoun antecedence problem? As long as we accept the basic *descriptive* facts about pronoun-antecedent distribution, the answer is, in all likelihood, No. The reason follows from the reduction argument: *if* it were the case that pronoun antecedence was easy to compute, by whatever component, call it $M$, then we could use this component to solve the pronoun antecedence problem using the syntactic (LF) based representation, assuming that there were a polynomial-time (easy to compute) transformation between LF and $M$. Then the original $\mathcal{NP}$-complete problem used would be easy to solve, a contradiction.

Of course, one could always avoid this possibility by assuming that syntactic structure (or LF) and $M$ were related by a nonpolynomial computation, say even an arbitrarily complex recursive function. In other words, $M$ would not be related to the level of syntactic structure in any computationally simple way. An arbitrary relation is of course possible, but does not seem to hold of current linguistic formalisms, which for the most part relate syntax and semantics via simple pushdown-stack transductions (see, e.g., Borgida (1983) or Plátek and Sgall (1978)). In addition, the "decoupled" relation of syntactic structure and $M$ would be problematic. If this reasoning is correct, then it becomes irrelevant whether we believe that it is "syntax" or "semantics" that is doing the work for us. The point is that the computational work must get done in *some* component of the grammatical system. By using complexity theory in this way, we can sidestep complicated border disputes about the boundaries between syntax and semantics; we can even ignore those labels entirely. In this way the complexity results are strongly theory-neutral, holding as long as one simply adheres to the basic linguistic facts about obviation themselves. This is an additional advantage of the complexity theory approach.

In section 4 we shall examine the cognitive aspects of this complexity result and attempt to resolve the paradox of pronoun antecedence intractability.

## 2.2 *A review of known complexity results about natural language*

We next review many of the known results about the computational complexity of modern linguistic theories. We then turn in section 3 to *where* this complexity comes from.

Table I surveys known results about the *Universal* Recognition Problems (URPs) for linguistic theories and their subcomponents, with question marks indicating unconfirmed conjectures. (That is, we explicitly include the grammar as a parameter in the complexity analysis, as mentioned earlier. See Barton, Berwick, and Ristad (1987) for the source of many of these results and Ristad (1990) for the results on the *Barriers* model and VP ellipsis.) Note that *every* linguistic theory is bounded from below by *NP* – it is at least *NP*-hard. The general explanation for this is simple: it can be easily shown (Ristad and Berwick (1989)) that any descriptively adequate linguistic theory that embeds the simplest facts about agreement and word category ambiguity, what they call AGREEMENT GRAMMAR RECOGNITION, is *NP*-hard. It is quite easy to see how agreement phenomena can model 3-SAT: whether a lexical item is a noun or a verb mirrors whether a variable is true or false;

Table 1. A summary of known complexity results about some linguistic theories and their subcomponents. Question marks indicate conjectured relationships.

| Theory/component | Complexity lower bound (at *least* this hard) | Complexity upper bound (at *most* this hard) |
|---|---|---|
| Autosegmental phonology | *NP* | *NP* |
| 2-level morphology ("Kimmo" systems) (no surface null elts.) | *NP* | *NP* |

Table 1. (continued)

| Theory/component | Complexity lower bound (at *least* this hard) | Complexity upper bound (at *most* this hard) |
|---|---|---|
| 3-level morphology | All recursively enumerable (r.e.) sets | All r.e. sets |
| LFG | *NP* | ?*PSPACE* |
| GPSG (1985 Gazdar *et al.*) | *EXP-POLY* | ?*EXP-POLY* |
| Revised GPSG (R-GPSG) (Ristad 1986) | *NP* | *NP* |
| Agreement Grammars | *NP* | *NP* |
| Tree Adjoining Grammars (with agreement) | *NP* | ?*NP* |
| Functional Unification Grammars unification component *only* | *NP* | *NP* |
| Older Extended Standard Transformational Theory | *EXPTIME* | *EXPTIME* |
| *Barriers* transformational theory (including vp ellipsis) | *NP* | *PSPACE* |
| Lasnik & Saito 1984 transformational theory (incl. vp ellipsis) | *NP* | *PSPACE* |
| Intrasentential pronoun antecedence (pa) | *NP* | *NP* |
| pa + vp ellipsis copy + bind | *PSPACE* | *PSPACE* |

the syntactic structure of the clauses is easy to generate with a context-free grammar (CFG); one can force at least one word per clause to have the value true with a CFG; and the global consistency of truth assignments can be mirrored by feature agreement.

On the surface, then, an agreement-and-ambiguity sentence similar to *police police police* could contain no clues at all as to whether each word was a noun or a verb, and at the same times, agreement processes could force, say, the first occurrence of *police*, corresponding to the literal $x$, to agree with the third. This is exactly like a 3-SAT problem. (Note that the English example is not *exactly* like such a problem because word order in English gives some clues as to whether a token is a noun or a verb.) It is interesting that it is *features* rather than *rules* that lead to the intractability. Finally, note that AGREEMENT GRAMMAR RECOGNITION is in $\mathcal{NP}$, neatly illustrating the difference between problems whose solutions are hard to find but easy to verify: once one is given the "solution" – the parse tree and agreement feature values – to a sentence like *police police police*, it is trivial to check that the sentence is grammatical. But it is very difficult to find that solution in the first place without resorting to exhaustive search. Again, it is striking that introspectively at least this becomes difficult after the number of interacting variables reaches 3 or 4.

Summarizing, the plain fact is that *any* descriptively adequate linguistic theory will have $\mathcal{NP}$ as a lower bound on its complexity, what we might call, following Ristad (1990), the $\mathcal{NP}$ *thesis* for linguistic theories. This is already an important result, since it means that one cannot take the efficient processing of language for granted, as is usually done. In section 4 we shall see how to possibly resolve this paradox.

In addition, the $\mathcal{NP}$-hardness (or worse) of the remaining subcomponents of the linguistic theories shown in table I may be established by other straightforward reductions. (See section 3 below for details on autosegmental phonology; Kasper and Rounds (1990) for details on the extended notion of unification deployed in Functional Unification Grammar that includes disjunctive feature specification. The result on the older extended standard transformational theory is due to Rounds (1975).) As for the worst-case complexity conjectures in the table, it may be possible in lexical-functional grammar (LFG) to mirror the existential and universal quantifiers to yield a reduction from the standard $\mathcal{PSPACE}$-complete problem QUANTIFIED BOOLEAN FORMULAS (QBF), but an actual proof remains to be found.

This result would be consistent with the suggestion in Kaplan and

Bresnan (1982) that LFG can generate only strictly context-sensitive languages. It seems similarly unlikely that generalized phrase structure grammar would be more complex than the class *EXP-POLY*, which includes the union of all exponential functions with polynomial exponents. As for tree adjoining grammars with agreement features (TAGs), a reasonable speculation is that their corresponding URPs are in *NP*, since the fixed recognition problem for these grammars is in *P* and they bear a close relation to context-free grammars, lying just beyond them in terms of weak generative capacity (see Schabes and Joshi (1988) or Vijay-Shanker and Joshi (1985)).

## 3 The source of computational complexity in natural language: a case study

In this section we examine in some detail a result that illustrates the method of reduction and the use of computational complexity theory to discover the source of unwanted complexity in a linguistic theory. In particular, we shall look at phonological theory and the so-called ***autosegmental model*** (see, e. g., Halle and Vergnaud (1987)), since it has been widely cited as somehow more restricted than the older theory of Chomsky and Halle (1968). However, we shall see that in fact the autosegmental framework admits computational intractability as well. The source of the complexity is an unexpected violation of modularity. Then, in section 4 we shall explore several ways to eliminate that complexity.

### *3.1 Autosegmental phonology and computational complexity*

At first glance, autosegmental or metrical phonology seems ideally designed to *avoid* the problems of computational intractability. It aims to be a modular, restricted theory: formerly long-distant contraints which were accounted for by means of string-variables in Chomsky and Halle's (1968) *The Sound Patterns of English* (SPE), for instance, suprasegmental properties like stress or vowel harmony, are dealt with by parceling out these constraint onto separate "planes" on which the relevant predicates can be stated in terms of strict adjacency. Note that even in a restricted interpretation of SPE as using just context-sensitive rewrite rules we would have a *PSPACE*-hard system, since context-sensitive language recognition is *PSPACE*-complete. Intuitively, SPE's globally matching string variables, as with earlier transformational theories, are quite powerful. In contrast, the autosegmental planes are all supposed to interact without using string variables by projecting to a surface form that represents a set of timing slots, marked by *X*'s because their features are underspecified. It is this constraint that leads to the hope for improved computational performance.

In Halle's memorable image, the planes are like the leaves of a spiral notebook, while the spine of the book is the surface form that emerges by projecting the features of the planes to the spine. As is familiar, this is the way that one can model the intercalated CV patterns of Semitic morphology, studied by McCarthy (1979), McCarthy and Prince (1990), and many others. Its descriptive success and constraint leads to the hope that it might be immune from computational intractability.

But is the autosegmental theory really complexity immune? Berwick (1986) first showed in a computer science colloquium at the University of Pennsylvania that the autosegmental model is not as modular as it looks, because under certain conditions feature variables could apply across global domains to simulate the effects of 3-SAT. Thus the framework is still computationally intractable.

The reduction from 3-SAT to the autosegmental framework is fairly direct; see figure 5. Given an arbitrary 3-SAT formula, we construct an autosegmental recognition problem, namely, a sequency of underspecified segments (one for each literal in the original formula) that appear "on the surface" as it where, and whose feature values we must determine, along with a particular autosegmental system of tiers (planes) and a fixed set of syllabic possibilities; the segmental sequence has a valid autosegmental feature assignment if and only if the original formula is satisfiable. To do this, each Boolean variable can be represented as a separate autosegmental tier, in effect encoding one harmony process per variable and ensuring consistency of whatever features are assigned on that tier – whether the variable is consistently assigned to be *true* or *false*. The segments on the surface are underspecified for these values so we cannot know what their full feature specifications are simply by examining their surface form alone – just as in the agreement and ambiguity example described in the previous section where we did not know whether *police* was a noun or a verb, we don't know whether any particular segment *X* is *true* or *false*. Finally, the 3-SAT structure may be duplicated by a metrical-syllabic structure of three segments per syllable, on a separate Consonant-Vowel plane, with at least one vowel per syllable mirroring at least one *true* literal assigned per triple. The same features in the Consonant-Vowel tier that force at least one *true* per triple of literals must of course also appear on the variable tiers (since they represent the same true-assignment variables). Note that we may adopt a fairly restricted version of autosegmental theory where the *association lines* from any one tier to the surface segments do not cross (the lines from any one plane of elements to the spiral notebook center line do not cross).

Figure 5 shows the details of the reduction used by Ristad (1990), illustrating a sample reduction from the formula $(x \vee \bar{y} \vee z) \wedge (\bar{z} \vee y \vee z)$. Each variable corresponds to a (distinct) place of articulation in phonology like *back* or *high*; no association lines need cross. The input is just the original

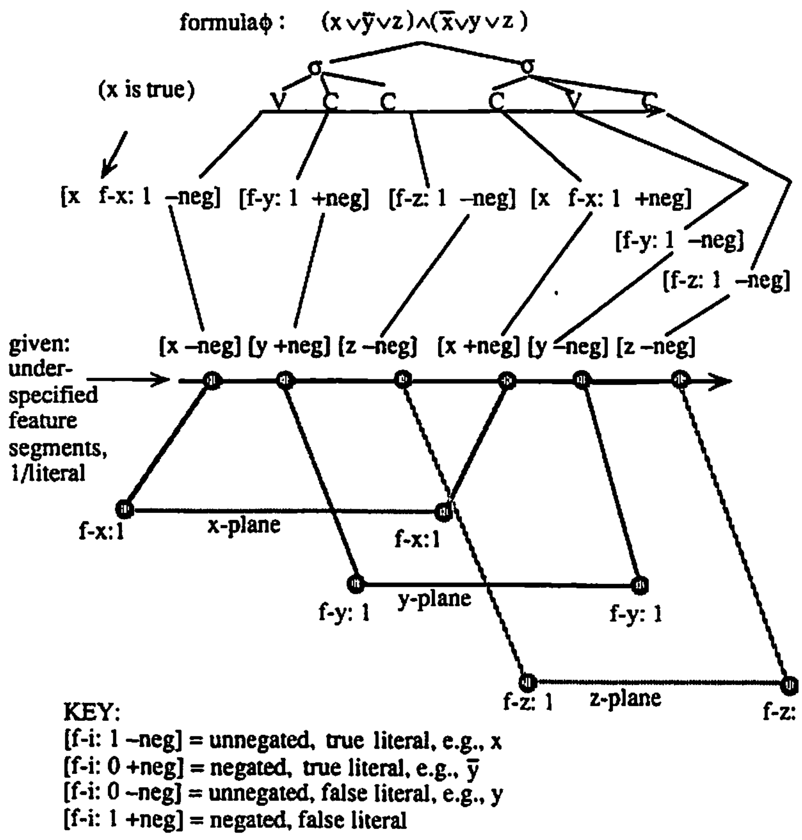
formulaφ : (x∨ȳ∨z)∧(x̄∨y∨z)
σ
σ
(x is true)
V C C C V C
[x f-x: 1 –neg] [f-y: 1 +neg] [f-z: 1 –neg] [x f-x: 1 +neg]
[f-y: 1 –neg]
[f-z: 1 –neg]
given: under-specified feature segments, 1/literal
[x –neg] [y +neg] [z –neg] [x +neg] [y –neg] [z –neg]
f-x:1
x-plane
f-x:1
f-y: 1
y-plane
f-y: 1
f-z: 1
z-plane
f-z: 1
KEY:
[f-i: 1 –neg] = unnegated, true literal, e.g., x
[f-i: 0 +neg] = negated, true literal, e.g., ȳ
[f-i: 0 –neg] = unnegated, false literal, e.g., y
[f-i: 1 +neg] = negated, false literal

Fig. 5. Standard autosegmental theory is $\mathcal{NP}$-hard, even though tier-based theory looks modular and even under constraints like no crossing association lines. This figure shows how an example 3-SAT formula instance can be reduced to an autosegmental recognition problem, modeled after Ristad (1990).

formula, with underspecified feature complexes $x$, $y$, etc. that are given minimally the values – neg (if the literal is unnegated, like $x$) or + neg (if the literal is negated, like $\bar{y}$). A "vowel" or true literal corresponds to one of two feature combinations: – neg along with a 1 value for the place of articulation assigned to the feature variable ($x, y, z, \ldots$ projected from the underspecified input) or + neg along with a 0 value for the place of articulation feature corresponding to a negated literal like $\bar{x}$. (The "consonants," or false literals, have the dual feature values: either – neg and 0, because then the corresponding unnegated variable must have the true assignment value *false*, or + neg and 1 because the negated literal has been assigned the value *false* and hence the corresponding unnegated value for that variable is *true*). In fact, the entire proof is exactly the same as for the proof that lexical-functional grammar recognition is $NP$-hard, in Berwick (1982). The end result is that the surface segments form a permissible phonological representation iff the original formula is satisfiable. The construction can

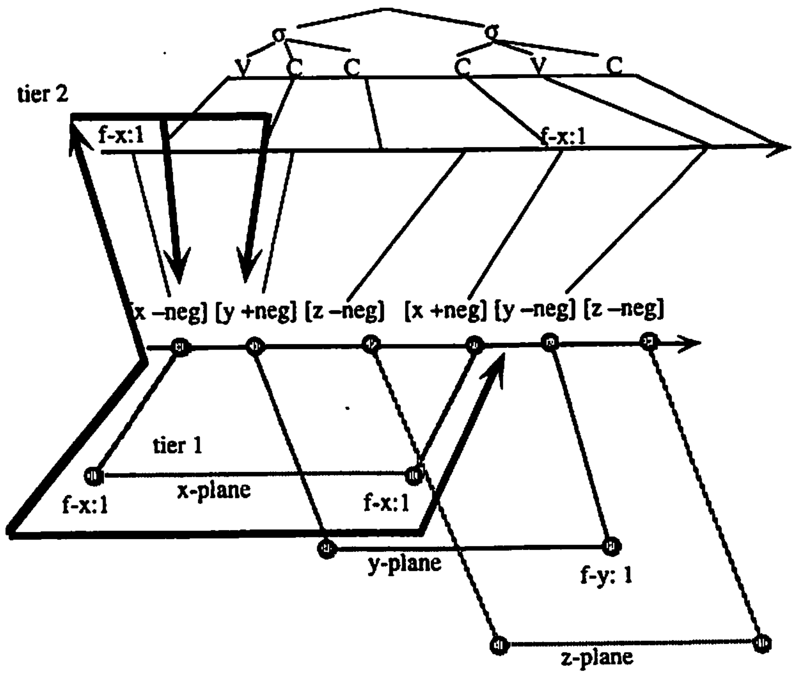

Fig. 6. The appearance of the same feature-value combinations on different tiers destroys the modularity of the autosegmental tiers. In effect, global variables are admitted into the system.

clearly be done in polynomial time, so autosegmental recognition is *NP*-hard.[6]

Why is the autosegmental framework computationally intractable when it at first appears so promising? Figure 6 tells the story. The problem is with the variables in the tiers. Because the same features (the truth-assignment feature for each variable) may appear on both the metrical tier and the vowel harmony or variables tiers, the variables planes are really not autonomous or modular even though they look like they are: the feature value we pick for the place of articulation in the *x*-plane in fact interacts with what we pick for the *y*-plane, just as with 3-SAT: if we pick, say, a place value of 1 for *x*, then we have one vowel in the first clause and *y* is free to have a place value of 1 or 0. So the tiers are not really independent after all. We shall take a look at potential remedies for this problem in section 4.

## 4 Escaping from computational intractability: restrictions on natural languages

Computational intractability, then, lurks everywhere in grammatical frameworks. How can one escape from the shackles of *NP*-hard intractability in grammatical theories? The symptoms dictate the cure. To sidestep intractability we have essentially three choices: (1) truncate the number of feature variables, showing that an unlimited number of variables/features are not needed descriptively in the linguistic theory; (2) change the representation, so that nonmodular interactions cannot occur, or (3) show that the hard examples do not arise "in practice" (alternatively, on the average). As it turns out, each of these solutions seems to be found in natural languages. (Of course, in any given situation a particular choice may not prove sufficient; the examples must be examined on a case-by-case basis.) One by one, let us see how these constraints can help.

6 It is also *NP*-complete, as Ristad (1990) shows, since it is easy to demonstrate that, given a sequence of surface segments and a syllabic system, one can simply guess all the possible segmental feature specifications, and then check them in deterministic polynomial time.

## 4.1 *Substantive linguistic constraints and feature truncation*

In the autosegmental model, the satisfiability simulation relies on an unbounded number of harmony processes. This is also true of the two-level morphology reduction described in Barton, Berwick, and Ristad (1987). However, natural languages do not ever seem to exhibit more than 3 independent harmony processes (Halle, personal communication; Koskenniemi and Church (1988)). This is an intriguing fact. Why should the number be three or four rather than, say, 17 or 20? Indeed, three seems to be a "breakpoint" for computational intractability: there is suggestive evidence that systems with three or fewer processes will generally (but not always) be easy to compute, while those with more will be difficult. Thus, a substantive linguistic constraint to three or fewer harmony processes may have some computational underpinnings (and supports the view that unlimited or nearly costless backtracking, required to simulate nondeterminism, is unavailable to people). Let us call this the *truncation* thesis.

We can establish the truncation thesis empirically. There are at least three striking, independent sources of evidence for it. First, we can carry out experiments on randomly constructed SAT formulas, in an attempt to discover the "hard" examples of SAT and the nondeterminism (as measured by backtracking) as one varies parameters like formula length, number of variables, etc. (see Purdom (1983); Goldberg, Purdom, and Brown (1982), though these results ordinarily assume an arbitrary number of variables).[7] In general, this is difficult to do because one must naturally make distributional assumptions to calculate expected running times, but some results are known, which we describe in more detail below. Roughly, if we assume that SAT formulas are uniformly distributed, and then vary the number of variables, the expected time (measured by amount of backtracking) to solve a SAT problem by, say, the usual Davis-Putnam procedure (1962) looks like the graph in figure 7.[8]

[7] Interestingly, these results don't seem to vary very much if one uses "smarter" backtracking procedures. The best known SAT algorithm currently known to the author is from Van Gelder (1988), which has a worst case time of $O(2^{n/8})$.

[8] This procedure is a backtracking search with these steps: (1) if there are no clauses, the formula is satisfiable; (2) if any clause is empty, the formula is unsatisfiable; (3) if there is any clause consisting of a single literal, select the variable in this clause, and set it so that the clause is true, then simplify and recurse; (4) if any variable appears only negated or unnegated, select it as the next variable, and set it so that its literals are true, proceed as in step 3; (5) select any variable and form two formulas by setting the variable to *true* in one and *false* in the other; recurse on each subproblem.

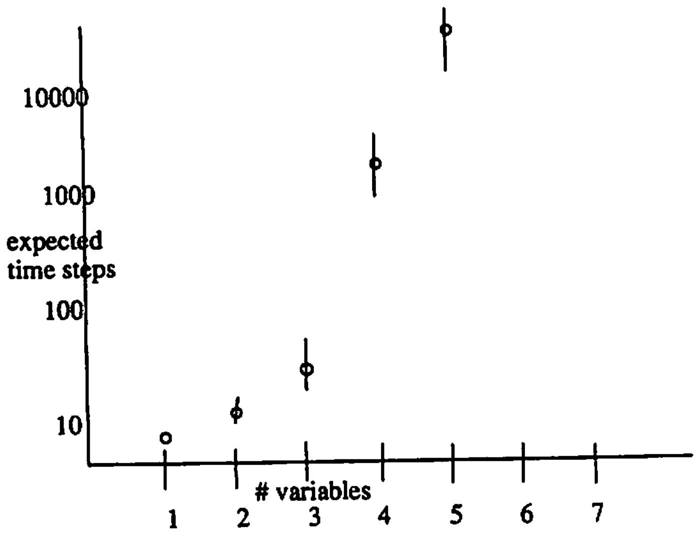

Fig. 7. This graph shows the increase in expected solution time (measured in terms of number of backtrackings) for randomly constructed 3-SATISFIABILITY formulas as the number of variables is increased. There is a sharp bend upwards at $n = 4$.

The graph in figure 7 shows that there is a breakpoint between $n = 3$ and $n = 4$ variables. (There are of course a fixed number of such formulas for any $n$). Beyond this inflection point, the expected solution time rises enormously. (Below we shall see that this breakpoint result is also confirmed by independent experiments with $k$-GRAPH COLORABILITY.) Returning now to natural grammars, this shows that, practically speaking, if all other things are held equal, then limiting harmony processes to three or less makes the resulting autosegmental problem tractable. Note that from a certain perspective this empirical result *confirms* the validity of the complexity analysis because by assuming that natural grammars are designed to avoid intractability we can arrive at an explanation of why harmony processes are limited to three or fewer: if there were more, then computational intractability could arise very easily.

Second, as another example of the power of truncation, Berwick (1989) shows that if the number of antecedent names is bounded, then pronoun antecedence is in $P$. In fact there is some evidence that the number of names must be small, again just 3 or 4, to gain tractability. Introspectively this is true, and this result is indirectly supported by the $k$-GRAPH COLORABILITY simulations discussed below.

Third, additional support for truncation position comes from an examination of agreement processes in natural grammars. Agreement and ambiguity systems are NP-*hard*, in general. However, the agreement features actually *used* in any natural language (indeed, across all natural languages) are limited quite strikingly in exactly the same way as harmony processes. Natural languages use roughly 3 or 4 agreement features, and no more: e.g., person, number, exclusivity, or gender.[9] Again this small number is an otherwise a mysterious fact, particularly coupled with the observed restrictions on harmony processes and the comparatively large number of, say, phonemes or rule processes in natural languages. This fact becomes less mysterious if we assume that computationally intractable systems are (sometimes) avoided by natural grammatical systems.

Truncation facts like these argue that people cannot have the unlimited computational resources required to simulate a nondeterministic Turing machine (e.g., unlimited parallelism or cost-free backtracking). If they did, then there would be no apparent reason to limit natural grammars in any of these ways, at least on grounds of computational complexity. (There might of course be some other reason for the restriction.)

## 4.2 *Changing the linguistic representations*

Rather than substantively restrict the *number* of processes in a linguistic framework, we next consider more fundamental representational changes in grammatical systems that would remove intractability. The basic idea is to increase the modularity of the system, using the traditional sense of that term.

In the autosegmental case, the complexity reduction can be blocked if distinct tiers cannot share the same features. Note that this *automatically* isolates each variable to a strictly contained, autonomous domain, as originally desired in the autosegmental theory. This constraint has been sometimes proposed on independent linguistic grounds, but we can see now that this constraint has an independent, complexity-theoretic motivation.

Similarly, if agreement processes could be contained within strictly local domains so that *no* variable information can be passed from one domain to another, then the 3-SAT reduction used for agreement and ambiguity

[9] As mentioned earlier, we take systems like the Bantu long/short classifiers to be something else, not used for agreement or matching; similarly for kinship terms.

grammars would not work. In linguistic terms, this would amount to limiting agreement relations to, for example, single S domains. This restriction seems too strong as stated. For instance, one would not be able to link the agreement features in one NP, say, *the guy* to the agreement features of NPs arbitrarily far away. But pronoun linking seems to involve just such distances. A closer approximation to the needed restriction involves information encapsulation or modularity. If we can ensure that the agreement between, e. g., a subject NP and a verb does not interact with some *other* local agreement process, then the reduction is blocked (recall that in 3-SAT the choices of a truth-assignment value for $x$ interact with choices for its neighbors).

The same question of modularity arises in analyzing the computational complexity of a particular version of the modern principles and parameters transformational theory (the *Barriers* government binding theory, Chomsky (1986)). Putting aside the matter of agreement and ambiguity, Table I indicated that this theory, too, is $\mathcal{NP}$-hard (a result of Ristad (1990)). The reason why is instructive. In this theory, local agreement processes, indicated by the coindexing of a Subject NP and the inflectional element (the *specifier* of the sentence and the *head* of the sentence), are allowed to interact with global movement processes, indicated by the same kind of coindexing between an NP and the underlying location where that NP is to be interpreted in terms of thematic structure. For example, an NP in object position might have to be moved to subject position in a passivetype construction, and then agree with the verb in person and number. Ristad shows that because the theory uses the *same* coindexing machinery to encode both kinds of grammatical relations, we can effectively transmit information just as in the autosegmental case. Indeed, the two constructions are topologically identical.

This suggests the same remedy for breaking the complexity of the *Barriers* theory: Use different machinery for agreement coindexing, head and complement selection, and movement coindexing. Then this particular reduction does not go through, because we cannot get the conflation of complement selection, specifier-head agreement, and movement to work together as one. If these elements cannot be conflated, we cannot get variable truth assignment to work together with this additional SAT constraint. In short, a more modular account of indexing, with a different indexing system for each "tier" of syntactic representation within phrase structure, is needed.

## *4.3 Complexity in practice: average-case complexity and efficient processing*

Finally, while complexity theory's abstraction away from particular algorithms or implementations is quite valuable, the theory also assumes "worst case" results. But do these arise in practice? It may be that human cognitive processors never encounter the hard examples, just as they never have to process deeply center-embedded sentences because they are not produced by speakers. Note that this is again a kind of truncation effect, this time on the distribution of input problem examples.

What we need, then, is some measure of the *average-case* complexity of linguistic problems, as they arise in practice. One can do this by direct sampling experiments, or one can use the developing theory of average case complexity. Both methods suggest that computationally intractable problems arise infrequently enough to solve the paradox of natural language computational complexity in many cases. Let us consider four examples that point in this direction, one sampling experiment and three average case complexity results.

**Sampling experiments.** One can carry out empirical experiments on the two-level morpholy system that decomposes surface form words like *tries* into their underlying morphemes like *try* + *s* (plural), taking into account spelling change rules. Table I observes that this problem is *NP*-hard, as proved in Barton, Berwick, and Ristad (1987). But this is a worst-case analysis. What about in practice? Similar to the 3-SAT experiments in figure 7, we can plot the amount of backtracking against word lenght, which directly measures the amount of nondeterminism in the system and abstracts away from certain details of machine running time.[10]

A good test language is Warlpiri, which exhibits significant harmony and reduplication effects, but certainly fewer than two independent harmony processes. The results are as expected. For example, below is a trace of the system recognizing the surface form *pinyi* (the hyphens are inserted for readability only). Recall that the system consists of two sets of finite-state automata, one that checks for each surface – underlying form pairing type, and other that checks for possible co-occurrences of stems and roots. In the

10 We must also test the system on more than unambiguous words/underlying letter trees, for otherwise this would just be like testing the Earley algorithm on unambiguous sentences; of course the time will be linear in the length of the input in such cases.

trace below, each backtracking point is numbered in square brackets and refers to that state to which the system must return and proceed from. In the example that follows, there are 9 such backtracks.

Recognizing surface form "pi-nyi".

1. (0) (y.0) → automaton NYJ Orthography blocks from state 1.
2. (0) + (p.p) → (1 1 1 1 1 1 1).
3. (1) (i.i) → (1 1 1 1 1 1 2).
4. (2) Nothing to do.
5. (1) [3] (⟨u1⟩.i) → (1 1 1 2 1 1 2).
6. (2) Entry |p⟨u1⟩| ends → new lexicon] V3STEM, config (1 1 1 2 1 1 2).
7. (3) (–.0) → (1 3 1 2 1 1 2).
8. (4) Nothing to do.
9. (3) [7] (–.–) → (1 2 1 2 1 1 2).
10. (4) (n.n) → (1 2 2 3 1 1 2).
11. (5) (y.0) → automaton ⟨ui⟩ Assimilation blocks from state 3.
12. (5) + (y.y) → (1 2 1 1 1 1 2).
13. (6) (i.i) → (1 2 1 1 1 1 2).
14. (7) Entry |-nyi| ends → new lexicon] END, config (1 2 1 1 1 1 2).
15. (8) End → result is ("p⟨ui⟩-nyi" (ODAMAGE V NPST)).
16. (7) [14] Entry |-nyi| ends → new lexicon DIRENCLITIC, config (1 2 1 1 1 1 2) 17. (8) (–.0) → automaton Hyphen Realization blocks from state 2.
18. (7) [14] Entry |-nyi| ends → new lexicon] WORD, config (1 2 1 1 1 1 2)
19. (8) (#.0) → automaton Hyphen Realization blocks from state 2.
20. (7) [14] Entry |-nyi| ends → new lexicon SUBJ, config (1 2 1 1 1 1 2)
21. (8) (–.0) → automaton Hyphen Realization blocks from state 2.
22. (7) [14] Entry |-nyi| ends → new lexicon OBJ, config (1 2 1 1 1 1 2)
23. (8) (–.0) → automaton Hyphen Realization blocks from state 2.
24. (7) [14] Entry |-nyi| ends → new lexicon RLA, config (1 2 1 1 1 1 2)
25. (8) (–.0) → automaton Hyphen Realization blocks from state 2.
26. (7) [14] Entry |-nyi| ends → mew lexicon], config (1 2 1 1 1 1 2)
27. (8) (–.0) → automaton Hyphen Realization blocks from state 2.
28. (1) [3] (I.i) → (1 1 1 1 1 2 2).
29. (2) Nothing to do.

"pi-nyi" ⇒ ("p⟨u1⟩-nyi" (ODAMAGE V NPST))

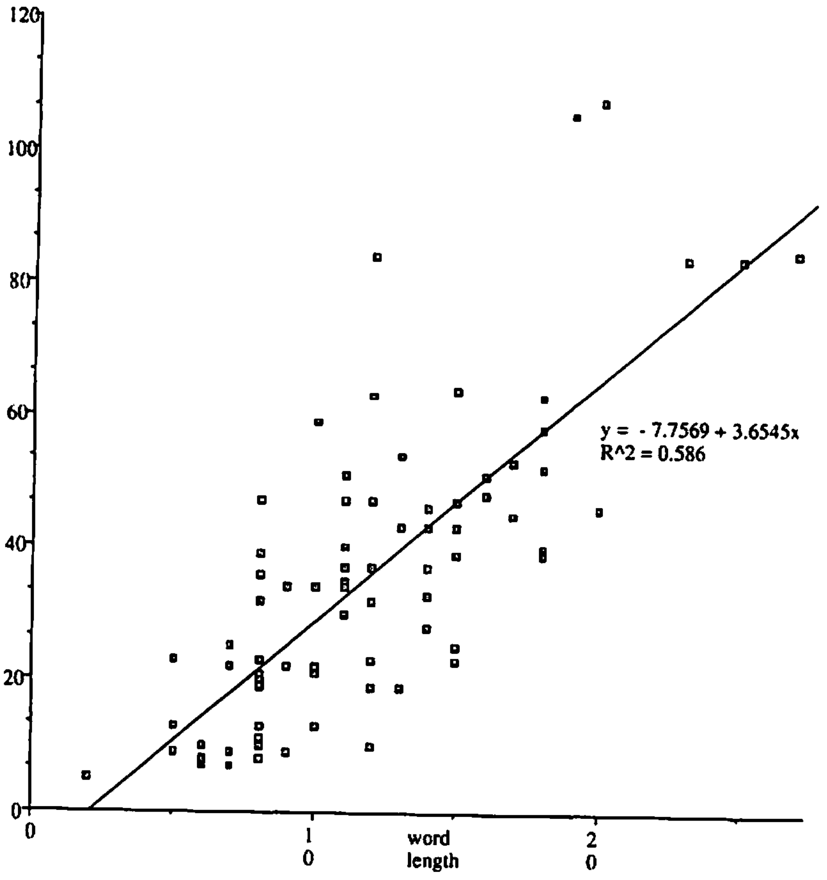

Fig. 8. A graph of the backtracking done by the two-level morphological analyzer for 81 Warlpiri words. The amount of backtracking is linear in word length, reflecting the relative sparsity of "hard" problems along with the nonambiguity of some underlying word decompositions.

Figure 8 displays the resulting graph of word length vs. backtracking for a distribution of over 80 Warlpiri words. Note that the two-level system does substantial backtracking: the amount of backtracking grows linearly with input length. Thus, in this sample, the amount of backtracking is not the *best* it could be – it is *not* a constant amount of backtracking, which would result in a linear time algorithm and which is claimed, evidently incorrectly, by Koskenniemi (1983) for the two-level both worst-case and average-case analyses – but neither is it the ***worst*** the two-level formalism allows – an exponential amount of backtracking (since the problem is $\mathcal{NP}$-hard).

Average-case behavior of 3-SAT. In short, "hard" two-level problems don't seem to arise in practice, even for examples that involve just two or three harmony processes. In fact this is a general result, not just for linguistic examples. "Hard" cases of the problems used for the linguistic reductions are hard to come by. But if the hard cases of a problem are rare, then expected running time for the problem will, on average, be easy. Since 3-SAT was used for most of the reductions mentioned, it is important to note that 3-SAT is considered to be average-time polynomial, although the distribution assumptions under which this holds are not clear (see Johnson (1984) for additional discussion). What this means is that *most* 3-SAT problems are in fact easy, and that improved backtracking methods (see note 7) work in expected polynomial time (see Purdom (1983) for a review).

It is a challenge to transfer mathematical results about average-case complexity to the linguistic domain. For an *average-case complexity* reduction to hold, not only must the reduction be done quickly, it must preserve properties of the original distribution assumed for the known problem we started with. In particular, for the reductions we have used, both 3-SAT formulas and linguistic examples must have roughly the same actual distributions. But this is by no means clear; we do not even know what the linguistic distributions look like. Still, sampling experiments like those described immediately above suggest that this good average time behavior carries over to linguistic examples. This is an area that remains to be investigated in depth.[11]

**Average-time behavior of extended unification.** To buttress this finding, we note that Kasper and Rounds (1990) have investigated the complexity of extended unification, including disjunctions, which is required in linguistic frameworks such as functional unification grammar. They point out that the algorithms for solving such extended agreement problems often have good average-time behavior, and that in fact Kasper's (1987) algorithm for the solving the consistency of a disjunctive feature specification runs in cubic time in many practical instances.

[11] Alternatively, we could attempt to find a problem that can "simulate" all the distributions in a class (i.e., is average-case complete for $\mathcal{NP}$) but such problems are rare (see Gurevich (1989) for a review) and no such reductions have been carried out for lingustic problems that this author is aware of.

**Pronoun antecedence and average-time behavior of *k*-GRAPH COLORABILITY.** Recall from section 2 and table I Ristad's (1990) demonstration that INTRASENTENTIAL PRONOUN ANTECEDENCE is $\mathcal{NP}$-complete. That reduction hinges on *k*-GRAPH COLORABILITY: finding a valid assignment of *n* pronouns to *k* antecedents within a sentence can be as hard as coloring a graph with *n* vertices with *k* colors such that no two connected vertices have the same color. This latter problem is $\mathcal{NP}$-complete. But, as is widely known (Johnson (1984)) most such problems are easy because the answer is almost always No. Take for example 3-colorability. A random graph will almost always contain a 4-clique – a totally connected subgraph with 4 vertices – and hence, not be 3-colorable.

More generally, if the probability of selecting a graph is fixed, standard backtrack search solves the problem in constant time – it takes roughly 197 steps on average (Wilf (1985)). However, here again there is a breakpoint at $n = 4$ or $n = 5$; the search tree for 5-colorability contains 750,000 nodes. Thus, we would expect backtrack search to perform quite poorly past this point, and it does (Wilf (1985)).

This result is completely consistent with the truncation effect in pronoun antecedence suggested earlier and figure 7, and strengthens the truncation thesis. If human cognitive systems use simple backtracking search, and if they are essentially deterministic engines – so that they do not have unlimited parallel processors or memory at their disposal to simulate a nondeterministic Turing machine – then we would expect pronoun antecedence computation to become difficult with four antecedents. This in fact seems to be the case. Simple backtracking can work, because people need to solve only small or simple problems.

Summarizing, there is no paradox between the computational intractability of linguistic theories and the observed efficiency with which people process language. We need not even invoke the *deus ex machina* of unlimited parallelism, whatever other merits parallel processing may have. Natural languages avoid intractability by adopting truly autonomous, modular representations; by incorporating substantive constraints that truncate computationally troublesome problems before they arise; by mirroring problems whose average-time behavior is good; or by handling only simple, small inputs. Evidently a delicate balance is maintained between the immense computational sophistication of language and its actual processing by people with limited, deterministic computational resources.

## ACKNOWLEDGEMENTS

This paper is in large part drawn from work by my graduate students S. Fong and E. Ristad. I would like to thank them for many stimulating discussions. This research is supported in part by a grant from the National Science Foundation under a Presidential Young Investigator Award and by the Kapor Family Foundation. Any residual errors are of course my responsibility alone.

## REFERENCES

Barton, E., Berwick R., Ristad E. (1987), Computational Complexity and Natural Language. Cambridge, MA: MIT Press.

Berwick, R. (1982), Computational complexity and lexical-functional grammar. American Journal of Computational Linguistics 8, 97–109.

Berwick, R. (1986), Computational complexity and the information structure of natural language. Paper presented at the University of Pennsylvania Cognitive Science Colloquium, Philadelphia, PA, March, 1986.

Berwick, R. (1989), Natural language, computational complexity, and generative capacity. Computers and Artificial Intelligence 8, 423–441.

Berwick, R., Weinberg A. (1984), The Grammatical Basis of Linguistic Performance. Cambridge, MA: MIT Press.

Borgida, A. (1983), Some formal results about stratificational grammar and their relevance to linguistic theory. Mathematical Systems Theory 16, pp. 29–56.

Chomsky, N., Halle, M. (1968), The Sound Pattern of English. Cambridge, MA: MIT Press.

Chomsky, N. (1986), Barriers. Cambridge, MA: MIT Press.

Davis, M., Logemann, G., Loveland, D. (1962), A machine program for theorem proving. Communications of the Association for Computing Machinery, 5, 394–397.

Fodor, J. A. (1983), The Modularity of Mind. Cambridge, MA: MIT Press.

Garey, M., Johnson, D. (1979), Computers and Intractability. San Francisco: W. H. Freeman and Company.

Gazdar, G., Klein, E., Pullum G., Sag, I. (1985) Generalized Phrase Structure Grammar. London: Basil Blackwell.

Giorgi, A., Pianesi, F., Satta, G. (1990), Computational aspects of operator-variable structures. Geneva, Switzerland: Workshop on GB Parsing, June 15–16, unpublished m. s.

Goldberg, A., Purdom, P., Brown, C. (1982), Average time analyses of simplified Davis-Putnam procedures. Information Processing Letters 15, 72–75.

Gurevich, Y. (1989), Average case completeness. Ann Arbor, MI: University of Michigan Electrical Engineering and Computer Science Department, unpublished m.s.

Halle, M., Vergnaud, J.-R. (1987), An Essay on Stress. Cambridge, MA: MIT Press.

Johnson, D. (1984), The NP-completeness column: an ongoing guide. Journal of Algorithms 5, 284–299.

Kaplan, R., Bresnan, J. (1982), Lexical-functional grammar: a formal system for grammatical representations. pp. 173–281 in Bresnan, J. (Ed.), The Mental Representation of Grammatical Relation. Cambridge, MA: MIT Press.

Kaspar, R. (1987), Feature structures: a logical theory with applications to natural language analysis. Ann Arbor, MI: Ph.D. dissertation, University of Michigan Electrical Engineering and Computer Science Department.

Kasper, R., Rounds, W. (1990), The logic of unification in grammar. Linguistics and Philosophy 13, 35–58.

Koskenniemi, K. (1983), Two-level morphology: a general computational model for word-form recognition and production. Helsinki, Finland: University of Helsinki Department of General Linguistics publication number 11.

Koskenniemi, K., Church, K. (1988), Complexity, two-level morphology, and Finnish. pp. 335–340 in: Proceedings of Coling–88, Budapest, Hungary.

Lasnik, H., Saito, M. (1984), On the nature of proper government. Linguistic Inqüiry 15, 235–289.

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

McCarthy, J. (1979), Semitic morphology. Cambridge, MA: Ph.D. dissertation, MIT Department of Linguistics and Philosophy.

McCarthy, J., Prince A. (1990), Foot and word in prosodic morphology: the Arabic broken plural. Natural Language and Linguistic Theory 8, 177–208.

Plátek, M., Sgall, P. (1978), A scale of context-sensitive languages: applications to natural languages. Information and Control 38, 1–20.

Purdom, P. (1983), Search rearrangement backtracking and polynomial average time. Artificial Intelligence 21, 117–133.

Ristad, E., Berwick, R. (1989), Computational consequences of agreement and ambiguity in natural language. Journal of Mathematical Psychology 33, 379–396.

Ristad, E. (1986), Complexity of linguistic models: a computational analysis and reconstruction of generalized phrase structure grammar. Cambridge, MA: M.S. dissertation, MIT of Electrical Engineering and Computer Science.

Ristad, E. (1990), Computational structure of human language. Cambridge, MA: Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science.

Rounds, W. (1975), A grammatical characterization of the exponential time language. Proceedings of the 16th Annual Symposium on Switching Theory and Automata, New York: IEEE Computer Society, 135–143.

SCHABES, Y., JOSHI, A. (1988), An Earley-type parsing algorithm for tree adjoining grammars. Philadelphia, PA: University of Pennsylvania Department of Computer and Information Sciences Report MS-CIS-88-36.

VAN GELDER, A. (1988), A satisfiability tester for non-clausal propositional calculus. Information and Computation 79, 1–21.

VIJAY-SHANKER, JOSHI, A. (1985), A CKY algorithm for parsing tree adjoining grammars. Philadelphia, PA: University of Pennsylvania Department of Computer and Information Sciences Report MS-CIS-85-24.

WILF, H. (1985), Some examples of combinatorial averaging. American Mathematical Monthly 92, 250–261.