Reprinted from JOURNAL OF MATHEMATICAL PSYCHOLOGY
All Rigts Reserved by Academic Press, New York and London

Vol. 33, No. 4, December 1989
Printed in Belgium

Computational Consequences of Agreement and Ambiguity in Natural Language

ERIC SVEN RISTAD AND ROBERT C. BERWICK

Massachusetts Institute of Technology

We argue that the modern computer science technique of computational complexity analysis can provide powerful insights into the algorithm-neutral analysis of information-processing tasks. In particular, we show that a simple, theory-neutral linguistic model of syntactic agreement and lexical ambiguity demonstrates that natural language parsing may be computationally intractable, extending the classic work of Chomsky and Miller (1963). Significantly, we show that it may be *syntactic features* rather than complex *rules* that can cause this difficulty. Informally, human languages and the computationally intractable satisfiability problem (SAT) share two costly computational mechanisms: both enforce agreement among terminal symbols across unbounded distances and both allow terminal symbol ambiguity. In natural languages, lexical elements may be required to agree (or disagree) on such features as person, number, and gender (e.g., subject/verb agreement in English); in SAT, agreement ensures the consistency of variable truth assignments. Lexical ambiguity can appear freely in natural language utterances (*can* may be a noun, verb, or auxiliary), while a variable in a SAT formula may be either true of false. When coupled with a deterministic performance model, this complexity result explains a subtle psycholinguistic distinction between *discovering* and *verifying* the grammaticality of an utterance. Finally, the applicability of computational complexity analysis to other cognitive faculties such as vision is discussed. © 1989 Academic Press, Inc.

1. INTRODUCTION

What is language? On one account, it is our ability to pair sound and meaning, ultimately an information-processing task. In this paper, we argue that modern computational complexity theory can provide powerful insights into the structure of this problem by providing an algorithm-neutral analysis of information-processing structure.

Specifically we show two things. First, contrary to what is commonly assumed,

Reprint requests should be addressed to Eric Sven Ristad, Artificial Intelligence Lab, Massachusetts Instituted Technology, 545 Technology Square, 805, Cambridge, Massachusetts 02139.

We thank G. Edward Barton for discussions that inspired this work, and anonymous reviewers for the *Journal of Mathematical Psychology* for comments that greatly improved it. This paper describes research done at the Artificial Intelligence Laboratory at the Massachusetts Institute of Technology. Support for this research has been provided by an IBM Fellowship to Eric Sven Ristad, by National Science Foundation Grant No. DCR-8552543 to Professor Robert C. Berwick under a Presidential Young Investigator Award, and by National Science Foundation Grant IRI-8511531.

0022-2496/89 $3.00

most, perhaps all, natural languages are *not* easy to parse: some grammatical sentences are too complex to be understood by person or machine. Second, computational complexity theory's distinction between the difficulty of *finding* a solution and of *verifying* a solution has a precise analog in the domain of natural language processing. In brief, we demonstrate formally that sentences combining syntactic agreement with syntactically ambiguous words can quickly become too difficult to parse, although their well-formedness may be verified easily once a paraphrased "solution" is provided. Since possibly all natural languages exhibit ambiguity and agreement, such as subject–verb agreement and noun/verb homophones like *block* in English (see Section 4), this result provides a robust, modern counterpart to Miller and Chomsky's classic distinction between abstract knowledge of language—linguistic competence—and how that knowledge is put to use—performance. Our result moves beyond Miller and Chomsky's in four ways: its application of computational complexity theory; its formal specification of the syntactic phenomena of agreement and ambiguity as a precise model that we call *agreement grammars*; its prediction of a specific class of sentences that are difficult to analyze but easy to check for well-formedness in retrospect, as a consequence of a sentence processor's purely *deterministic* operation rather than simply its *finite* characterization; and its broad applicability to most, perhaps all, natural languages.

The remainder of this paper is organized as follows. Section 2 outlines our approach to applying complexity theory in the language processing domain, reviewing the essential terminology of computational complexity theory that will be used in the sequel. Section 3 formalizes the purely syntactic phenomena of agreement and ambiguity in terms of *agreement grammars*. It then outlines a proof that any natural language containing syntactically ambiguous elements and agreement constraints will contain sentences that are computationally intractable to parse. Section 4 discusses the implications of this result, indicating how the distinction between solution and verification is reflected in human sentence processing. An Appendix provides formal details of our proofs.

2. Complexity Theory and Psychological Models

Following Marr (1980), we assume the scientific explanation of any complex biological information-processing system demands at least three distinct theoretical levels: (1) a *computational theory*, explaining what is computed and why, including algorithm-neutral representations for the input and output of the process; (2) an *algorithmic theory* that can account for the transformation of input to output; and (3) a (hardware) *implementation theory*, or the device in which the representation and algorithm are physically realized. Accordingly, the study of linguistic knowledge divides into the study of competence and performance. A theory of *competence* corresponds to Marr's topmost level of computational theory, explaining what information structures are computed and why, while abstracting away from algorithmic details, memory limitations, shifts of attention or interest, and errors.

Marr's remaining levels belong to the theory of *performance*, that proposes a representation, algorithm, implementation triple to account for actual language use.

Once we understand the topmost of Marr's levels—the computational theory of an information-processing problem—we can understand more about the other levels as well:

> Although algorithms and mechanisms are empirically more accessible, it is the top level, the level of computational theory, which is critically important from an information-processing point of view. The reason for this is that the nature of the computations that underlie perception depends more upon the computational problems that have to be solved than upon the particular hardware in which their solutions are implemented. To phrase the matter another way, an algorithm is likely to be understood more readily by understanding the nature of the problem being solved than by examining the mechanisms (and the hardware) in which it is embodied. (Marr, 1980, p. 27.)

What then is the role of complexity theory in scientific explanation? Computational complexity theory measures the intrinsic difficulty of solving an (information-processing) problem no matter how its solution is obtained; for example, the problem of arranging a list of n names into alphabetic order. Inherently then, complexity theory studies *problem structure*: it classifies problems according to the amount of computational resources (for example, time or space) needed to solve them on some abstract computer model, typically a deterministic Turing machine. Complexity classifications are invariant across a wide range of primitive machine models, all choices of representation, algorithm, and actual implementation, and even the resource measure itself.

It is important to see how powerful this invariance is. *Any* change in the problem representation that preserves the essential features of the original problem (preserving solutions to the original problem, in effect, its descriptive adequacy) can have no effect on its complexity classification. The robustness of these classifications makes complexity theory ideally suited for studying cognition: while we do know something about the abstract problems the brain solves, we do not know much about the representations, algorithms, or hardware involved. "If we believe that the aim of information-processing studies is to formulate and understand particular information-processing problems, then the structure of those problems is central..." (Marr, 1980, p. 347.)

The two complexity classes we distinguish below are $\mathscr{P}$ and $\mathscr{NP}$. $\mathscr{P}$ is the natural and important class of problems solvable in deterministic $\mathscr{P}$olynomial time, that is, on a deterministic Turing machine in time n^j for some integer j, where n denotes the size of the problem to be solved.[1] $\mathscr{P}$ is considered to be the class of problems that can be solved efficiently. For example, sorting takes $n \cdot \log n$ time in the worst case using a variety of algorithms, and therefore is efficiently solvable.

$\mathscr{NP}$ is the class of all problems solvable in $\mathscr{N}$ondeterministic $\mathscr{P}$olynomial time in

[1] Problems must be encoded in a "reasonable" way for a size measure to make sense: for discussion, see Garey and Johnson (1979).

the worst case. Informally, a problem is in $\mathcal{NP}$ if one can guess an answer to the problem and then verify its correctness in polynomial time. Such problems have no known polynomial-time (efficient) solution algorithms. For example, the problem of deciding whether a whole number i is composite is in $\mathcal{NP}$ because it can be solved by guessing a pair of potential divisors, and then quickly checking if their product equals i.

A problem T is *NP-hard* if it is at least as hard computationally as any problem in the class $\mathcal{NP}$: if we had a subroutine that solved T in polynomial time, then we could write a program to solve any problem in $\mathcal{NP}$ in polynomial time (essentially by efficiently transforming the problem in $\mathcal{NP}$ to T and then solving T with the fast subroutine: the Appendix gives a more detailed account of how this procedure, known as *problem reduction,* works). Note that T need not be in $\mathcal{NP}$ to be NP-hard. A problem is *NP-complete* if it is both in $\mathcal{NP}$ and NP-hard.

NP-complete problems can be solved only by methods too slow for even the fastest computers.[2] Since it is widely believed, though not yet proved, that no faster methods of solution can ever be found for these problems, NP-complete problems are considered computationally intractable. A famous NP-complete problem is the traveling salesman problem: to find the shortest route for a traveling salesman who must visit a number of cities and return to the city started at. For additional details, the reader may refer to Garey and Johnson (1979); Lewis and Papadimitriou (1978); or Barton, Berwick, and Ristad (1987). This last work explores further the relationship between computational complexity and natural language.

3. Modeling Agreement and Ambiguity

Having reviewed the basic terminology of complexity theory, we now turn to problem of formally modeling agreement and ambiguity in natural languages.

Syntactic agreement and ambiguity are widespread in human languages. Agreement can be morphological (word based) or structural, and can hold across unbounded distances and among unlimited sets of elements. This is quite easy to demonstrate. For example, in nearly all languages, predicates must agree with their arguments. In English, morphological agreement includes subject–verb agreement on person, number, gender, animacy, humanity, abstractness, quantity, and other features. Agreement occurs at the intra-morpheme level in some languages, for example, in Turkish where a suffix ending such as *ü* forces agreement in vowel qualities with preceding vowels; this phenomenon occurs widely in such diverse languages as Finnish, Arabic, Hebrew, and the Australian language Warlpiri.

Case marking is another form of agreement that surfaces both morphologically and syntactically in natural languages. Noun forms such as epithets, pronouns, and

[2] However, some NP-complete problems have good *average*-time behavior, that is, the instances that occur most often can be efficiently solved. We discuss such behavior in relation to our NP-completeness result below in Section 4.

anaphora may be required to agree of disagree with other noun forms in person, number, gender, and so forth, as in, *Reagan, the fool, believed he could appoint justices himself.* Typically such agreement can occur over an unbounded number of words or phrases. In short, syntactic agreement is a widespread phenomenon of natural languages generally, perhaps found in all natural languages.

Ambiguity is equally common in natural languages. Syntactic homonyms are typical: in English, the word *block* may be a noun or a verb. Ambiguity in quantifier scope and reference are equally common; for example, the dual meaning of *Everyone loves someone.*

Descriptively adequate linguistic theories must therefore describe these two phenomena, and, in fact, all major linguistic theories do so, using three devices: (1) *distinctive features* to represent the dimensions of agreement; (2) an *agreement enforcement* mechanism; and (3) provision for lexical and structural (syntactic) *ambiguity.*

While different theories work out the details of these three devices in different ways, one can abstract away from these variations in order to model just agreement and ambiguity and study their computational complexity. We introduce agreement grammars here as a simple formal linguistic model with exactly these three devices. Agreement grammars are not natural language grammars. For one thing, agreement grammars are too simple to completely model any natural language. They can also generate infinitely many unnatural languages, such as Σ^*, or any finite, regular, or context-free language. But while our results apply only to this simplified formal model of agreement and ambiguity, it is nonetheless true that all current linguistic theories that attempt to describe natural grammars readily embed the agreement grammar problem (see Barton, Berwick, & Ristad 1987). In this respect, our approach follows that of Kirousis and Papadimitriou (1985), who study the complexity of a formal model of scene recognition known as line-labeling. While line-labeling is *not* the same as scene recognition, it may be construed as a simple formal model embedded in the full-scale problem of scene recognition.

We begin with an informal introduction, and follow that with a formal specification of agreement grammars.

3.1. *Defining agreement grammars*

Following conventional notation, we first recall that a context-free grammar G is a 4-tuple,

$$G = \langle V_N, V_T, P, S \rangle,$$

where V_N is a finite set of *nonterminal symbols*, V_T a finite set of *terminal symbols*, P a finite set of *productions* of the form $A \rightarrow \gamma$, where $A \in V_N$ and $\gamma \in (V_N \cup V_T)^*$, and S is a distinguished *start* symbol. If P contains a production $A \rightarrow \gamma$, then for any α, $\beta \in (V_n \cup V_T)^*$, we write $\alpha A \beta \Rightarrow \alpha\gamma\beta$ and say that $\alpha A \beta$ *derives* $\alpha\gamma\beta$ with respect to G. We let $\overset{*}{\Rightarrow}$ be the reflexive transitive closure of $\Rightarrow$, dropping the clause "with respect to G" where it is understood from context. The *language* $L(G)$

generated by a context-free grammar is the set of all strictly terminal strings that can be derived from the start symbol with respect to G, that is,

$$L(G)=\{x: x\in V_T^* \text{ and } S\overset{*}{\Rightarrow} x\}$$

We extend context-free grammars to obtain agreement grammars (AG's) by adding nonterminals that are sets of features and by imposing an agreement condition on the derivation relation.

A feature is a [*feature-name feature-value*] pair. For example, [PER 1] is a possible feature, denoting first-person. Some features may be designated *agreement features,* and required to match other features (see below). For instance, an AG nonterminal labeling the first person pronoun *I* could be written {[CAT N], [PLU −], [PER 1]}, while the singular verb *sleeps* could be labeled with the AG nonterminal features {[CAT V], [PLU −], [PER 3]}.

More formally, we define the set of nonterminals in the following way. The set of nonterminals in an agreement grammar is characterized by a *specification* $\langle F, A, \rho\rangle$, where F is a finite set of feature names and A is a finite set of feature values. ρ is a function from feature names to permissible feature values; that is, $\rho: F\rightarrow 2^A$. $\langle F, A, \rho\rangle$ *specifies* a finite set V_N of nonterminals, where a nonterminal may also be thought of as a partial function from feature-names to feature-values:

$$V_N=\{C\in A^{(F)}: \forall f\in \textsc{dom}(C)[C(f)\in\rho(f)]\}$$

Here $Y^{(X)}$ is the set of all partial functions from X to Y. $\textsc{dom}(C)$ is the domain of C, that is the set $\{x: \exists y[\langle x, y\rangle\in C]\}$. A category C' *extends* a category C (written $C'\sqsupseteq C$) if and only if $\forall f\in\textsc{dom}(C)$, $[C'(f)=C(f)]$, that is, C' is a superset of C. For example, the category {[PER 1], [NUM 1]} extends the category {[PER 1]}.

An *agreement grammar* (AG) is a 5-tuple,

$$G=\langle\langle F, A, \rho\rangle, V_T, F_A, P, S\rangle,$$

whose first element specifies a set V_N of syntactic categories and where V_T is a finite terminal alphabet. F_A is the set of agreement feature names, $F_A\subseteq F$. S is the distinguished starting symbol, $S\in V_N$. P is a finite set of the usual context-free productions, each member taking one of the forms

1. $C\rightarrow a$, where $C\in V_N$ and $a\in V_T$,
2. $C_0\rightarrow C_1\cdots C_n$, where each $C_i\in V_N$.

No so-called null productions or epsilon transitions are permitted: each production must have at least one non-null element on its righthand side.

To complete our definition, we modify the *derives* relation to incorporate agreement. We say that a production $C_0'\rightarrow C_1'\cdots C_n'$ *extends* a production $C_0\rightarrow C_1\cdots C_n$ if and only if C_i' extends C_i for every i and the mother's agreement features appear on every daughter:

1. $\forall i,\ 0 \leqslant i \leqslant n,\ [C'_i \sqsupseteq C_i]$, and
2. $\forall f \in (\mathrm{DOM}(C'_0) \cap F_A),\ \forall i,\ 1 \leqslant i \leqslant n,\ [(f \in \mathrm{DOM}(C'_i)) \wedge (C'_i(f) = C'_0(f))]$

The last condition (the *agreement convention*) ensures that all agreement features on the mother are also found on all daughters.

We may now define the language generated by an agreement grammar. If P contains a production $A \to \gamma$ with an extension $A' \to \gamma'$, then for any $\alpha, \beta \in (V_N \cup V_T)^*$, we write $\alpha A' \beta \Rightarrow \alpha \gamma' \beta$. Let $\overset{*}{\Rightarrow}$ be the reflexive transitive closure of $\Rightarrow$ in the given grammar G. The language $L(G)$ generated by G contains all terminal strings that can be derived from any extension of the start category:

$$L(G) = \{x : x \in V_T^* \text{ and } \exists S',\ [S' \sqsupseteq S, \text{ and } S' \overset{*}{\Rightarrow} x]\}.$$

A NATURAL LANGUAGE EXAMPLE. The following artificial agreement grammar G_1 models subject–verb agreement for person and number in English.

1. G_1 includes the set F of feature names $\{\text{CAT, PLU, PER}\}$ and the function ρ defined by

$$\rho(\text{CAT}) = \{\text{S, VP, NP, V, N}\}$$

$$\rho(\text{PER}) = \{1, 2, 3\}$$

$$\rho(\text{PLU}) = \{+, -\}.$$

The start category S is $\{[\text{CAT S}]\}$, and the set of agreement feature names $F_A = \{\text{PER, PLU}\}$. The feature CAT encodes the syntactic category of the nonterminal (sentence, noun phrase, and so forth). PER encodes person (first, second, or third), and PLU encodes number ($[\text{PLU} +]$ is plural, $[\text{PLU} -]$ is singular).

2. The terminal vocabulary of G_1 is

$$V_T = \{I, men, John, sleep, sleeps\}.$$

3. G_1 contains the following 9 productions:

$$\{[\text{CAT S}]\} \to \{[\text{CAT NP}]\}\{[\text{CAT VP}]\}$$

$$\{[\text{CAT VP}]\} \to \{[\text{CAT V}]\}$$

$$\{[\text{CAT NP}]\} \to \{[\text{CAT N}]\}$$

$$\{[\text{CAT NP}], [\text{PLU} -], [\text{PER } 1]\} \to I$$

$$\{[\text{CAT N}], [\text{PLU} +]\} \to \textit{men}$$

$$\{[\text{CAT NP}], [\text{PLU} -], [\text{PER } 3]\} \to \textit{John}$$

$$\{[\text{CAT V}], [\text{PLU} +]\} \to \textit{sleep}$$

$$\{[\text{CAT V}], [\text{PLU} -], [\text{PER } 1]\} \to \textit{sleep}$$

$$\{[\text{CAT V}], [\text{PLU} -], [\text{PER } 3]\} \to \textit{sleeps}.$$

The sample grammar generates exactly the following sentences:

a. I sleep (= {[CAT S], [PER 1], [PLU −]})
b. men sleep (= {[CAT S], [PLU +]})
c. John sleeps (= {[CAT S], [PER 3], [plu −]}).

We next turn to the computational complexity of recognizing sentences generated by an arbitrary agreement grammar.

3.2. *The computational complexity of agreement grammar recognition*

Given an arbitrary agreement grammar, how hard is it to parse using the agreement features of that grammar? Computational complexity theory gives us a precise answer to this question. We may state the recognition problem for agreement grammars as follows:

> Given an arbitrary agreement grammar AG and a string x, is $x \in L(\mathrm{AG})$?

This problem is NP-complete. Intuitively, feature agreement let us "simulate" the problem of finding out whether there exists an assignment of truth-values to variables that satisfies an arbitrary Boolean formula in 3-conjunctive normal form, that is, a formula such as

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

where there are exactly three disjoined variables per clause, and each clause is conjoined with the next. This problem is called 3SAT (for "three-satisfiability"); the appendix provides a formal definition of this problem.[3] Feature agreement simulates the assignment of truth-values: if y is given the value *true* in one clause, then it must be *true* in all other clauses (and $\bar{y}$ must be *false*). Syntactic category ambiguity simulates the fact that we must "guess" whether y is to have the value *true* or *false*, just as we must sometimes guess whether *block* is a noun or a verb. Finally, ordinary context-free productions may be used to guarantee that there is at least one *true* variable per clause, as is demanded for there to be a satisfying truth-assignment. This simulation, formally called a reduction, establishes that AG recognition is NP-hard. To establish inclusion in $\mathcal{NP}$ we use the impossibility of null transitions in AG's to derive a polynomial bound on the length of a shortest derivation. Given this, it is easy to show that a nondeterministic program can

[3] The possibility that the agreement grammar recognition problem might be NP-complete and a general ideal of how to prove it arose out of a discussion between the authors and E. Barton.

"guess" membership of x in $L(\mathrm{AG})$ in polynomial time. The proof follows that in Barton, Berwick, and Ristad (1987) and is spelled out in the Appendix.

It is important to note that this complexity result is a function of both input sentence length *and* grammar size. At first glance, this might seem unreasonable. A child learning a language might be able to discover a more compact, highly efficient grammar to use. Similarly, people appear to use *one* grammar to process sentences, not a family of grammars. If the grammar were fixed, then it would not be part of the input to the problem, and a polynomial time recognition algorithm might exist.

But factoring out grammar size has many problems, as discussed in Barton, Berwick, and Ristad (1987). To summarize these: (1) Complexity analysis should consider all relevant inputs; grammar size is an important, direct component of recognition algorithms, and therefore it is wrong to ignore this dominant element of recognition time. This is especially true for natural languages, where grammar size is much larger than expected sentence length (typically by a factor of 10^3 or more). (2) Known preprocessing steps for agreement grammars all fail, because they expand the grammar size exponentially, which acts as a huge constant factor of $2^{|G|}$ multiplying the recognition time. For example, a full grammar with 10,000 rules could require time $2^{10{,}000}\cdot n^3$ to parse—polynomial time in a strict sense, but impossibly long in practical terms.

What this NP-completeness result means is that there is no known algorithm for determining membership in the language of an arbitrary agreement grammar that does not in effect exhaustively check an exponential number of possible feature combinations. Further, there is no known reasonable representational recasting of the AG recognition problem that would do better. Interestingly, this NP-completeness result does not rely on the context-free power of the AG model. The agreement grammar used in the reduction generates a regular language, and essentially the same reduction would apply to an agreement grammar whose language was finite. The reduction relies only on the combinatorial possibilities that arise from nonlocal agreement and ambiguity.

Put another way, natural languages that incorporate the minimal machinery of agreement and ambiguity are *inherently* asymptotically intractable. This intractability arises from the interaction of agreement and ambiguity. Informally, human languages and the NP-complete satisfiability problem (SAT) share two costly computational mechanisms: both enforce agreement among terminal symbols across unbounded distances and both allow terminal symbol ambiguity. In natural language, lexical elements may be required to agree (or disagree) on such features as person, number, gender, case, count, category, reference, thematic role, tense, and abstractness (subject/verb agreement in English, for example); in SAT, agreement ensures the consistency of variable truth assignments. Lexical ambiguity can appear freely in natural language utterances (is *can* a noun, verb, or auxiliary verb?), while a variable in a SAT formula may be either true or false. Thus, the linguistic mechanisms for agreement and ambiguity are exactly those needed to simulate satisfiability—any linguistic theory that uses them, as any descriptively adequate theory must, will be computationally intractable.

4. Intractability and Linguistic Performance

Having established the inherent computational intractability of descriptively adequate linguistic theories, we turn next to the implications of this result for models of human sentence processing. We show that the fundamental difference between *finding* and *verifying* a result surfaces in the agreement grammar case, and in the associated natural language examples.

Following Miller and Chomsky (1963), let us imagine a linguistic performance model M (a "parser") that is fundamentally deterministic and assigns structural descriptions to utterances in real time. Refining their discussion, by "deterministic" we mean that M may have limited parallelism and cannot guess correct answers. This machine model is thought to include all physically realizable computing machines, from the fastest digital computers to the brain.

A consequence of the NP-completeness results of the previous section is that M will not be able to analyze certain constructions involving both ambiguity and long-distance agreement. This result does not dispute that short, unambiguous, or structurally simple utterances can be processed efficiently.[4] More importantly, given the apparent speed of ordinary language use, the result suggests that actual biological recognizers may be both fast and occasionally inaccurate. M will, however, be able to efficiently "verify" (in a sense to be clarified below) many of the constructions it fails to analyze. The choice of a deterministic performance model, when coupled with the AG model of competence, indicates that some performance limitations will arise out of the *deterministic* nature of processing (see Berwick & Weinberg, 1982) rather than from the finite nature of human cognitive capacity (see Miller & Chomsky, 1963).

As evidence of such a performance limitation, consider examples that exhibit excessive lexical and structural ambiguity, as in Sentence (1) below, where *buffalo* can be one or many shaggy beasts, a city, or a transitive verb that means *fool*. The sentences in (2) demonstrate the same effect with the elaborate agreement processes found in consecutive constituent coordination, discontinuous constituent coordination, rightward movement out of coordinate structures, and gapping. Sentence (1) has an array of possible interpretations, ranging from the simple interpretation suggested by the parallel sentence "Boston buffalo fool Boston buffalo" to more elaborate ones with relative clauses, for example "[Buffalo that buffalo fool] can fool buffalo."[5]

buffalo buffalo buffalo buffalo buffalo (1)

a. John owned and then sold hundreds of late model cars
to us that he waxed all the time.

[4] Thus, the fact that many NP-complete problems have good "average time" solutions does not contradict our result. In fact, given the preponderant distribution of short utterances, it reinforces our result, as the discussion below makes clear.

[5] Equivalent sentences can be constructed out of any word whose plural noun form is morphologically identical to its plural verb form: *police police police police police*, *french french french french french*, etc.

b. John liked and wanted to tease Sue and Bill, Mary.

c. John owned and then sold hundreds of late model cars to us and Bill, trucks. (2)

d. John owned and then sold hundreds of late model cars to us and to Bill, trucks.

Examples combining the two phenomena become even worse: *Buffalo buffalo buffalo and buffalo buffalo buffalo of buffalo buffalo to buffalo buffalo buffalo buffalo and buffalo, buffalo buffalo.*

Linguistic agreement and ambiguity may cause intractability in other languages as well. Free word order languages such as Warlpiri, a central Australian aborigine language, have special morphology for verbs and for norminal arguments that make sentences such as the *buffalo* examples easy to understand when they are directly translated. But the morphological processes in these languages typically allow other highly ambiguous constructions that are difficult to understand. For example, Warlpiri fails to distinguish adjectives and nouns either morphologically or configurationally (as in English), making the direct translation of such trivial English sentences as *John flushed the Air Force space shuttle toilet* computationally analogous to the intractable "buffalo" sentences of English. We conclude that there is, in fact, a class of grammatical sentences whose recognition complexity can grow exponentially faster than their length, and therefore, contrary to common belief, natural language may not be efficiently parsable in general.

Significantly, the preceding natural language examples have the computational character of NP-complete problems: solutions may be hard to find, but they are easy to verify. This is a nontrivial result because there is no a priori reason why solutions to a problem should be easy to verify. Thus, if full natural language understanding was harder than NP-complete, as has been suggested by Chomsky (1980), then some grammatical sentences could never be understood, even with extensive priming and prompting.

The reader's first attempt to understand the *buffalo* sentence is likely to fail completely. However, it is generally easy to *check* the paraphrased "solution." The curious nature of this phenomenon confirms the predictions made by the model M, since it is a property of a deterministic machine operating under polynomial time constraints that it will be unable to find an analysis of the agreement-type sentences. On the other hand, M should be able to verify an agreement sentence, since such NP-hard problems are, by definition, verifiable in polynomial time by a deterministic machine. This result also argues that neither agreement nor ambiguity should be bounded by the competence model.

An explanation of the psycholinguistic dichotomy between solving and verifying relies critically on the competence/performance distinction. The *possibility* of understanding (verifying) the utterances at all is explained by a competence theory that does not bound agreement or ambiguity. On the other hand, the *difficulty* of understanding (solving) the utterances is best explained by the deterministic nature of the performance model.

Agreement and ambiguity, if permitted to operate without bound in the speaker, will quickly generate utterances that exceed the (deterministic) perceptual capabilities of hearers. These sentences, being too difficult for the hearer to understand, will not be used due to the *fidelity criterion* of communication systems (see Chomsky & Miller, 1963, p. 273). The fidelity criterion states that the receiver establishes the criterion of acceptability of a communication system: if the receiver cannot process a signal, then the fidelity of the communication channel is wasted. Simply put, unacceptably ambiguous sentences, being difficult for the hearer, are not used in practice, "just as many other proliferations of syntactic devices that produce well-formed sentences will never actually be found," (Miller & Chomsky, 1963, p. 471).

Nearly all utterances evince both agreement processes and ambiguity to varying degrees. Therefore, there is no reason to expect that occasional unacceptability introduced by excessive agreement and ambiguity will cause those processes to disappear from language over the course of time. In fact, all known natural languages employ these mechanisms. It would be reasonable to expect, however, that natural language processing systems might develop techniques to efficiently process the "easy" cases and approximately process the "hard" ones.

It remains to develop a complete theory of approximate processing for hard problems, but complexity theory again suggests some possible answers. One approach is what we advance above: hard sentences are not in fact solved, but only verified for grammaticality upon paraphrase. Another approach is to simply restrict the domain of problems solved: only short agreement sentences will be analyzed, and analysis of those exceeding a set resource limit will be aborted.

More generally, on this analysis, whenever the computational cost of a task matches its observed cognitive cost, we know that scientific explanation of the task should occur primarily in a theory of competence and that the performance theory is likely to be straightforward: that is, deterministic and faithful. But whenever the inherent computational cost differs from measured cognitive cost, complexity theory yields specific insight into the performance theory: what needs to be explained at that level and the form such an explanation might take.

If complexity theory classifies a cognitive problem as intractable, yet humans appear to solve that problem efficiently, this suggests that the performance algorithm restricts its input domain or solves costly instances only approximately (as in *simulated annealing*; see Kirkpatrick, Gelatt, & Vecchi, 1983 for further discussion), or perhaps aided by parallel hardware specially designed for the cognitive problem at hand.

On the other hand, a problem could be easy in principle, yet impossible for people to solve. Then the performance algorithm might be simple-minded, inefficient, or quite restricted (as with the "no reentrant procedures" constraint of an early performance model), or the mental hardware might limit memory use or processing time.

To take a simple example from another cognitive domain, Kirousis and Papadimitriou (1985) consider the complexity of the historically important line-labeling problem in machine vision. They prove that the line-labeling problem and the more general scene recognition problem are NP-complete, and likely to be intractable. Given the apparent speed with which humans recognize scenes, and hence the surprising nature of their result, they suggest that computationally difficult scenes are scarce in practice, or that real-world hints (for example, surface texture and assorted depth clues) might simplify the real-world scene recognition problem.

In either situation, then, the complexity analysis of information-processing tasks can lead to significant conclusions about linguistic performance because complexity theory makes strong empirical predictions. Agreement grammars provide a linguistically and algorithmically neutral model for agreement and ambiguity in natural languages. Agreement grammar recognition is theoretically intractable. We have also observed that the buffalo-type sentences are difficult for humans to process. In order to explain this apparent match between predicted intractability and observed cognitive difficulty, we are led to postulate a deterministic processing model for English, and perhaps for all natural languages.

A. Formal Results

In this appendix we give additional details on the proof technique of *reduction*, followed by a formal proof of the NP-completeness result sketched in the main text.

A.1. *Reduction as a Proof Technique*

Complexity classifications are established with the proof technique of reduction. 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 operates in polynomial time. Therefore, if we had a polynomial time algorithm for solving S, then we could also solve T in polynomial time, simply by converting instances of T into S. (This follows because the composition of two polynomial time functions is also polynomial time.) Formally, if we choose T to be NP-complete, then the polynomial time reduction shows that S is at least as hard as T, or NP-hard. If we were also to prove that S was in $\mathcal{NP}$, then S would be NP-complete.

In this case, the known NP-complete problem T that we will use is 3SAT, and the problem S of unknown complexity is AG recognition. Therefore, the proof will reduce instances of 3SAT (a 3-conjunctive normal form or 3-CNF Boolean formula F) into instances of AG recognition (an AG G and input string x). The *3-satisfiability problem* (3SAT) is to determine, given a Boolean expression in 3-CNF, whether the formula is satisfiable. 3SAT is NP-complete. An example of a satisfiable 3-CNF Boolean formula with five clauses is

$$(a \vee b \vee c) \wedge (\bar{a} \vee d \vee e) \wedge (e \vee \bar{d} \vee \bar{c}) \wedge (b \vee c \vee d) \wedge (\bar{a} \vee \bar{d} \vee \bar{e}).$$

A *Boolean expression* is an expression composed of variables (e.g., x), parentheses, and the logical operators $\vee$ (OR), $\wedge$ (AND), and negation. Negation is represented as a horizontal bar over the negated expression (e.g., $\bar{x}$ is the negation of the variable x). A *literal* is a variable or the negation of a variable. Variables may have the values 0 (false) and 1 (true), as may expressions. An expression is *satisfiable* if there is some assignment of 0's and 1's to the variables that gives the expression the value 1.

A Boolean expression is in *conjunctive normal form* (CNF) if it is of the form $E_1 \wedge E_2 \wedge \cdots \wedge E_k$ and each clause E_i is of the form $\alpha_{i1} \vee \alpha_{i2} \vee \cdots \vee \alpha_{im_i}$, where each α_{ij} is a *literal*—either a variable x or a negated variable $\bar{x}$. An expression is in 3-CNF if each clause in the CNF expression contains exactly three distinct literals.

A.2. AG *Recognition is* NP-*Complete*

LEMMA A.1. *Let* $(\varphi_0, ..., \varphi_k)$ *be a shortest leftmost derivation of* φ_k *from* φ_0 *in an agreement grammar G containing at least one branching production.*[6] *If* $k > |P|$, *where P is the set of productions in G, then* $|\varphi_k| > |\varphi_0|$.

Proof. In the step $\varphi_i \Rightarrow \varphi_{i+1}$, where $\varphi_i = \alpha A'\beta$ *and* $\varphi_{i+1} = \alpha\gamma'\beta$ for $\alpha \in V_T^*$, $\beta \in (V_T \cup K)^*$, *one of the following cases must hold*:

1. The production $A \rightarrow \gamma$ with extension $A' \rightarrow \gamma'$ is nonbranching ($|\gamma| = 1$). In the worst case, we could cycle through every possible nonbranching production (without using a branching production), after which we would begin to reuse them. Any extension of a production that has already been used in this run of nonbranching productions could have been guessed previously, and the length of the shortest nonbranching run must be less than $|P|$.

2. The production $A \rightarrow \gamma$ with extension $A' \rightarrow \gamma'$ is branching ($|\gamma| > 1$). Then $|\varphi_i| > |\varphi_{i+1}|$.

A total of at most $n-1$ branching productions derives an utterance of length n, because there are no null-transitions in an agreement grammar. Each branching production can be separated from the closest other branching production in the derivation by a run of at most $|G|$ nonbranching productions, and the shortest derivation of x will be of length $\theta(|G| \cdot |x|)$. (As is conventional in computer science, the expression $\theta(x)$ stands for "exactly x.")

THEOREM 1. *Agreement grammar recognition is in* $\mathcal{NP}$.

Proof. On input agreement grammar G and input string $x \in V_T^*$, guess a derivation of x in nondeterministic polynomial time as follows.[7]

[6] If the agreement grammar G does not contain a branching production, then $L(G)$ contains only strings of length one and all shortest derivations are shorter than $|P|$: membership for such a grammar is clearly in $\mathcal{NP}$.

[7] Again, we assume G contains at least one branching production. If not, then we should only loop as many times as there are productions, and then halt.

1. Guess an extension S' of S, and let S' be the derivation string.
2. For a derivation string $\alpha A'\beta$, where $\alpha \in V_T^*$, $\beta \in (V_T \cup K)^*$, guess a production $A \to \gamma$ and extension $A' \to \gamma'$ of it. Let $\alpha\gamma'\beta$ be the new derivation string.
3. If $\alpha\gamma'\beta = x$, accept.
4. If $|\alpha\gamma'\beta| > |x|$, reject.
5. Loop to Step 2 (at most $|G| \cdot |x|$ times).

Every loop of the nondeterministic algorithm performs one step in the derivation. By Lemma A.1, the shortest derivation of x is of length at most $\theta(|G| \cdot |x|)$, so we need to loop through the algorithm at most that many times. Guessing an extension of a category may be performed in time $\theta(|F|)$, and an extension of a production may be guessed in time $\theta(|F| \cdot |P|)$. This nondeterministic algorithm runs in polynomial time and accepts exactly $L(G)$; hence AG recognition is in $\mathcal{NP}$. ▮

THEOREM 2. *AG recognition is NP-hard.*

Proof. We reduce 3SAT to AG recognition in polynomial time. Given a 3-CNF formula f of length m using the n variables $q_1 \cdots q_n$, we construct an agreement grammar G_f such that the string w is an element of $L(G_f)$ iff f is satisfiable, where w is the string of formula literals in f. G_f is constructed as follows:

1. G_f includes the set F of feature names $\{\text{STAGE}, \text{LITERAL}, q_1, ..., q_n\}$ with values defined by the function ρ:

$$\rho(\text{STAGE}) = \{1, ..., n+3\}$$
$$\rho(\text{LITERAL}) = \{+, -\}$$
$$\rho(q_i) = \{0, 1\}.$$

The grammar will assign truth-values to the variables and check satisfaction in $n+3$ stages as synchronized by the feature STAGE. The start category is $\{[\text{STAGE } 1]\}$.

2. At each of the first n stages, a value is chosen for one variable; because the q_i are declared as agreement features, the values that are chosen will be maintained throughout the derivation tree. The following $2n$ nonbranching rules are needed, constructed for all i, $1 \leqslant i \leqslant n$:

$$\{[\text{STAGE } i], [q_i\, 0]\} \to \{[\text{STAGE } i+1], [q_i\, 0]\}$$
$$\{[\text{STAGE } i], [q_i\, 1]\} \to \{[\text{STAGE } i+1], [q_i\, 1]\}.$$

Note that square brackets ([,]) delimit features, while curly brackets ({ , }) delimit the sets of features that form nonterminals.

3. At stage $n+1$, the grammar has guessed truth assignments for all variables; all that remains is to use the truth assignments to generate satisfied

three-literal clauses. The following two rules generate enough clauses to match the number of clauses in w:

$$\{[\text{STAGE } n+1]\} \rightarrow \{[\text{STAGE } n+2]\}$$
$$\{[\text{STAGE } n+1]\} \rightarrow \{[\text{STAGE } n+1]\}\{[\text{STAGE } n+2]\}.$$

4. At stage $n+2$, the grammar generates satisfied three-literal clauses—clauses containing at least one true literal. Let C_0 and C_1 be the categories

$$C_0 = \{[\text{STAGE } n+3], [\text{LITERAL } -]\}$$
$$C_1 = \{[\text{STAGE } n+3], [\text{LITERAL } +]\}.$$

Then the following seven ternary-branching rules are needed; any set of three literals makes the clause true, provided at least one literal is true:

$$\{[\text{STAGE } n+2]\} \rightarrow C_0 C_0 C_1$$
$$\{[\text{STAGE } n+2]\} \rightarrow C_0 C_1 C_0$$
$$\{[\text{STAGE } n+2]\} \rightarrow C_1 C_0 C_0$$
$$\{[\text{STAGE } n+2]\} \rightarrow C_0 C_1 C_1$$
$$\{[\text{STAGE } n+2]\} \rightarrow C_1 C_0 C_1$$
$$\{[\text{STAGE } n+2]\} \rightarrow C_1 C_1 C_0$$
$$\{[\text{STAGE } n+2]\} \rightarrow C_1 C_1 C_1.$$

5. Finally, lexical insertion at Stage $n+3$ ties together the truth-values chosen for the variables and the literals. For every q_i, $1 \leqslant i \leqslant n$, we need the following four nonbranching rules, bringing us to a total of $6n+9$ rules:

$$\{[\text{STAGE } n+3], [\text{LITERAL } +], [q_i\, 1]\} \rightarrow q_i$$
$$\{[\text{STAGE } n+3], [\text{LITERAL } -], [q_i\, 0]\} \rightarrow q_i$$
$$\{[\text{STAGE } n+3], [\text{LITERAL } +], [q_i\, 0]\} \rightarrow \bar{q}_i$$
$$\{[\text{STAGE } n+3], [\text{LITERAL } -], [q_i\, 1]\} \rightarrow \bar{q}_i.$$

If some extension of the start category $S = [\text{STAGE } 1]$ can be generated, then the formula f is satisfiable; each extension of the start category that generates a string must encode a satisfying truth assignment. For example, the category

$$\{[\text{STAGE } 1], [q_1\, 1], [q_2\, 0], ..., [q_n\, 1]\}$$

generates 3-CNF formulas f with the satisfying truth assignment $q_1 = 1$, $q_2 = 0$, ..., $q_n = 1$. Note that the agreement grammar constructed in the reduction

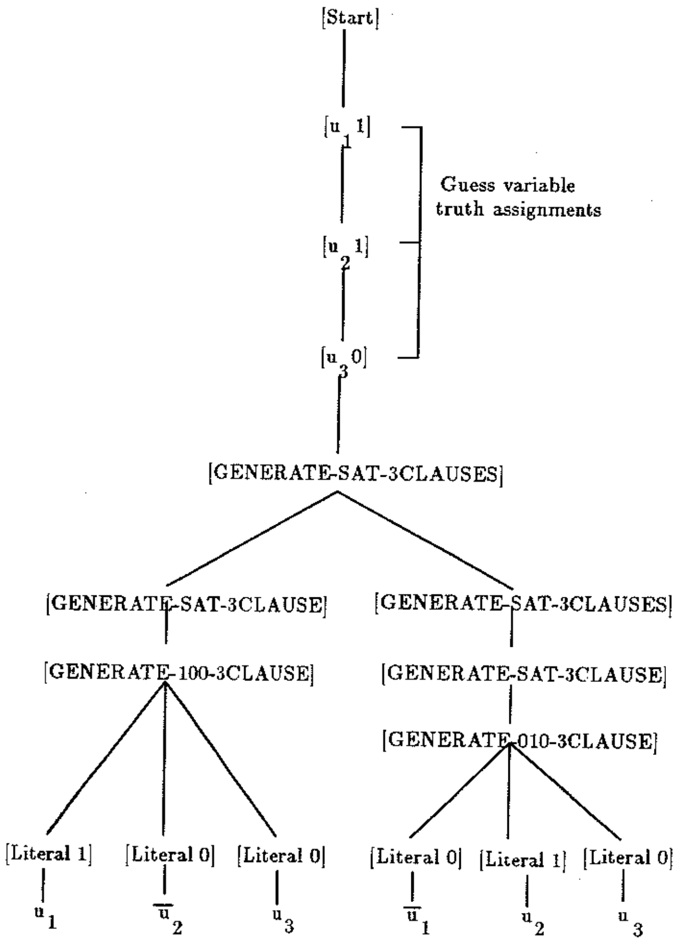

FIG. 1. A sample reduction for the agreement grammar proof, showing a 3SAT instance recoded as an AG parsing problem. The 3SAT problem to solve is $(u_1 \vee \bar{u}_2 \vee u_3) \wedge (\bar{u}_1 \vee u_2 \vee u_3)$, which has the satisfying assignment $u_1 = 1$, $u_2 = 1$, and $u_3 = 0$. The solution expressed at the terminal leaves of the tree is $u_1 =$ (i.e., true); $u_2 = 1$ (true); and $u_3 = 0$ (false).

generates *all* satisfiable 3-CNF Boolean formulas, of any length, using n or fewer variables. ∎

Figure 1 contains a sample reduction, showing the parse tree needed to analyze a 3SAT instance recoded as an AG parsing problem. The 3SAT instance to solve is $(u_1 \vee \bar{u}_2 \vee u_3) \wedge (\bar{u}_1 \vee u_2 \vee u_3)$, which has the satisfying assignment $u_1 = 1$, $u_2 = 1$, and $u_3 = 0$. The corresponding input string to be parsed is a formula of literals, $u_1 \bar{u}_2 u_3 \bar{u}_1 u_3 u_3$. The agreement features are therefore $\{u_1, u_2, u_3\}$.

REFERENCES

BARTON, E., BERWICK, R., & RISTAD, E. (1987). *Computational complexity and natural language.* Cambridge, MA: MIT Press.

BERWICK, R., & WEINBERG, A. (1982). *The grammatical basis of linguistic performance.* Cambridge, MA: MIT Press.

CHOMSKY, N. (1980). *Rules and representations.* New York: Columbia Univ. Press.

CHOMSKY, N., & MILLER, G. (1963). Introduction to the formal analysis of natural languages. In R. D. Luce, R. R. Bush, & E. Galanter (Eds.), *Handbook of mathematical psychology,* Vol. II (pp. 269–322). New York: Wiley.

GAREY, M., & JOHNSON, D. (1979). *Computers and intractability.* San Francisco: W. H. Freeman.

KIROUSIS, L., & PAPADIMITRIOU, C. (1985). The complexity of recognizing polyhedral scenes. In *Proceedings of the 26th annual IEEE symposium on the foundations of computer science* (pp. 175–185). Boston: IEEE Society.

KIRKPATRICK, S., GELATT, C. D., JR., & VECCHI, M. P. (1983). Optimization by simulated annealing. *Science,* **220**, 671–680.

MARR, D. (1980). *Vision.* San Francisco: Freeman.

MILLER, G. A., & CHOMSKY, N. (1963). Finitary models of language users. In R. D. Luce, R.R. Bush & E. Galanter (Eds.), *Handbook of mathematical psychology,* Vol. II (pp. 419–492). New York: Wiley.

RECEIVED: November 9, 1987

Printed by Catherine Press, Ltd., Tempelhof 41, B-8000 Brugge, Belgium