ELSEVIER

Lingua 97 (1995) 37–80

Lingua

The computational complexity of natural language recognition: A tutorial overview*

Hans van de Koot

Department of Phonetics and Linguistics, University College London, Gower Street, London WC1E 6BT, UK

Received November 1994; revised April 1995

* I would like to thank Eric Sven Ristad, an anonymous *Lingua* reviewer, Neil Smith and Villy Rouchota for their comments on a previous version.

SSDI 0024-3841(95)00018-6

1. Introduction

In recent years a substantial amount of research has been invested in applying the computer science tool of computational complexity theory to natural language and linguistic theory. Some of this work has been published (e.g. Ristad, 1986; Barton et al., 1987; Ristad, 1993) while other results have circulated in some other form (e.g. Ristad, 1991; Giorgi et al., 1989; van de Koot, 1992). In this article I give an overview of this exciting area of study and discuss some results and their implications for the field of linguistics.

Every normal human being acquires a natural language and eventually produces and comprehends that language with astonishing ease and speed. These abilities of language users are based on certain computations, which following Ristad (1993) we will refer to as language computations. The ultimate goal to which theoretical linguistics hopes to contribute is to provide a full characterisation of these computations. We will refer to such a complete theory of the computations performed by the language user as the language model.

Unfortunately, we do not have direct access to the computations performed by the language user. One way in which we might find out more about them is to study the knowledge of language that these computations make use of. This is the research program of generative linguistics. A generative grammar is not a computational model: it does not characterize a language computation. Whether such studies give us any direct insight into the language computations performed by language users has long been hotly debated, but it seems uncontroversial to maintain that a generative theory is a necessary first step on the way to a complete theory of human language.

A second way in which we could hope to uncover truths about the language model is to study the complexity of the problems posed by the production, comprehension and acquisition of language. By the complexity of a computation we mean the rate at which the computation consumes abstract resources, such as space and time. This avenue was first explored in the early 1950s following Chomsky's invention of formal languages and his well-known complexity hierarchy. Chomsky demonstrated convincingly that regular grammars and their associated finite-state automata cannot capture essential properties of natural languages, namely those that depend on the hierarchical structure of natural language expressions (e.g. the difference in meaning

between *(black board) eraser* and *black (board eraser)* depends on the way the words in the string are grouped together). This showed that human languages had to be more complex than regular languages.

However, studying the complexity of natural language with the help of the Chomsky hierarchy suffers from two essential weaknesses. First, there is no reason why the human language model should mesh neatly with a formal mathematical model. This has the consequence that although we might be able to find out something about the lower bound of the complexity of the class of natural languages (that it is *at least* so hard), we would only ever find out something about the upper bound (i.e. that it is *at most* so hard) if we had an exact match between language class and automaton, a very unlikely situation. Second, complexity studies based on the Chomsky hierarchy are concerned with weak generative capacity only and therefore do not give us much insight into properties of the grammar. Since our ultimate aim is to learn something about the language model, getting a result about the weak generative capacity that is required to generate natural languages is hardly very satisfying.

The work on complexity theory and natural language discussed below seeks to improve on this state of affairs by using much subtler tools to obtain insight into the complexity of language recognition for particular theories of grammar (Ristad, 1986, 1988, 1991, 1993; Barton et al., 1987; Giorgi et al., 1989; van de Koot, 1992). As we will see, such inquiries not only give us a window on the complexity of natural language comprehension but they can pinpoint unnaturally powerful machinery in a linguistic theory and suggest ways in which the theory might be improved.

Ristad's (1993) pioneering work goes one radical step further. He argues that since the linguistic theory is not a model of the language user *at any level of abstraction* (because it is a model of linguistic knowledge and not of how that knowledge is put to use), looking at the recognition problems associated with linguistic theories is still rather a roundabout way of finding out about the complexity of language computations. A better approach would study the complexity of language comprehension problems directly, without reference to the peculiarities of particular linguistic theories at all. The complexity results so arrived at relate language computations to other computations whose structure we understand and provide a design target for language algorithms (i.e. for algorithmic characterizations of language computations). The latter kind of result makes it possible to choose between competing analyses of a linguistic problem on the grounds that the computational complexity of one but not the other exceeds the design target. Ristad gives a nice example of such a result in the domain of VP-ellipsis.

The picture that emerges from all this work and in particular from Ristad (1993) is that natural language computations are computationally intractable – they cannot be solved in a reasonable amount of time by any known algorithm or machine. But they are intractable in just the right way in that they have the useful property of being on the verge of tractability: their solutions are hard to find but easy to check once found.

I begin this tutorial overview with a brief introduction into some basic concepts and techniques of computational complexity theory and discuss how this theory

could be usefully applied in the domain of linguistic theory (section 2). Here I do not consider any concrete results but focus on the advantages and disadvantages of the method per se. In sections 3 and 4 we will see how complexity theory can be used to obtain an understanding of the complexity of language computations. In section 3 we consider how complexity theory can be used to obtain results about the complexity of so-called Universal Recognition Problems (URPs) for a number of linguistic theories. These results give us insight into how difficult it is to determine that a given sentence is generated by a given grammar. To the extent that this problem is similar to the problem solved by language users, the complexity results thus obtained give us insight into the complexity of language computations. In particular, I review Barton et al.'s (1987) proof that the URP for Lexical Functional Grammar is computationally intractable and my own proof that the same is true of the URP for Word Grammar (van de Koot, 1992). What this means in practice is that there is nothing about these theories of grammar that guarantees efficient parsability. I then take a critical look at Ristad's (1991) proof that the URP for the Lasnik–Saito and Barriers models of grammar is computationally intractable. I argue that there is a mathematical flaw in the proof and in addition that the linguistic model to which it applies is not a correct representation of either Lasnik and Saito (1984) or Chomsky (1986). I conclude that Ristad (1991) has therefore not demonstrated that the URP is computationally intractable for these models. The discussion in section 3 includes substantial excerpts from Ristad (1991). I gratefully acknowledge his permission to reproduce this material here. In section 4 we turn to Ristad's (1993) approach to the study of the computational complexity of language production, comprehension and acquisition. In this work the complexity of language computations is studied *directly*, independently of the peculiarities of particular linguistic theories. I discuss the goals and methods of this work and explain its importance for linguistic research. Section 5 summarizes the main contributions of complexity theory to the field of linguistics.

2. Computational complexity theory

2.1. What is a computation?

A computation may be characterized at various levels of abstraction. Language computations, for instance, may be described as sequences of brain state transitions, or as an algorithm which specifies how each state is mapped into its successor state by some mentally represented computational device, or very abstractly as *a computational problem*. A computational problem only specifies the relation between the inputs and outputs of a computation, abstracting away completely from how a computational device computes that mapping. As a direct consequence of the fact that we can talk about computations at various levels of abstraction, we can also talk about the complexity of a computation at various levels of abstraction. In what follows, we will mostly talk about the complexity of computations at the highest level of abstraction, i.e. about the complexity of computational problems.

2.2. What is complexity?

A computational problem is solved by carrying out a computational procedure (some well-defined sequence of steps), otherwise known as an algorithm. Algorithms are executed by computational devices. Any known computational device (including the reader) needs a certain amount of (computation) time and (memory) space to carry out its computations. The rate at which a computation consumes these resources is a measure of the complexity of that computation. Therefore, the complexity of a problem can be determined by looking at the algorithms that solve it.

We can distinguish between the *time complexity* and the *space complexity* of a computation (or computational problem). It would not be very useful if we had to express the time and space complexity of computations relative to particular kinds of computing devices, so that we would have to revise our complexity estimates every time a new generation of computers appeared on the market. But fortunately there is no need for that. We can express time in terms of the number of 'steps' (state transitions) used and space in terms of the maximum amount of information required at any of these steps. Since the number of steps and the amount of space required is likely to be dependent on the size of the problem, it is necessary to quantify the complexity of a problem in terms of its size (after all you need more time to sort 100 names alphabetically than you need to sort 50). For instance, we might determine that a given computational problem has *polynomial* time complexity. The number of steps required to solve an instance of such a problem would be related to its size by a polynomial function, i.e. a function of the form $c \cdot n^k$, where c and k are constants and n expresses the size of the problem instance. The time complexity of some other problem might be given by a *super-polynomial* function, i.e. by a function with an exponential growth rate (of the form $c \cdot k^n$) or worse.

For the purposes of complexity theory, we are not interested in the value of the constant c in the functions expressing the time and space complexity of a problem. What we really want to know is the *order of growth* of the function. The reason for this is quite simple. The constant in the complexity functions is fairly meaningless: the difference between a problem with a time complexity of say $1000n^3$ and one with a time complexity of just $5n^3$ may be more or less obliterated in practice by the increase in speed of the next generation of processors. But no amount of increase in processor speed will ever bridge the gap between a problem with polynomial time complexity and one with a super-polynomial time complexity. A brief example of a super-polynomial time problem, the Monkey Puzzle Problem (from Harel, 1987) should suffice to make clear why.

One popular incarnation of the Monkey Puzzle Problem involves nine square cards whose sides are imprinted with the upper and lower halves of coloured monkeys. The objective is to find an arrangement of the nine cards into a 3×3 square such that monkey halves match and colours are identical wherever edges meet. What we want to do is look at the complexity of a generalized version of this problem in which we are given $N = M^2$ cards and our task is to determine whether there exists an $M \times M$ arrangement of the N cards such that halves and colours match wherever edges meet. All that is required is a yes/no-answer. We can simply quantify the size

of this general problem in terms of the number of cards we are given. We make the additional assumption that the cards have a fixed top, bottom, left and right side, so that they are not to be rotated.

As Harel points out, a naïve solution to this problem is easily constructed as follows. As each problem instance involves only a finite number of cards, it follows that there are only a finite number of $M \times M$ arrangements. For each such arrangement it is a trivial matter to check whether it is a legal arrangement. So all we need to do is devise an algorithm that will go through all possible arrangements in turn, stopping and saying 'yes' when a legal arrangement is found and stopping and saying 'no' when all possible arrangements have been considered and no legal one was found. What would be the time complexity of this algorithm? Suppose we are given $N = M^2$ cards. The cards will eventually be arranged in a $M \times M$ square. Thus for the first card there will be N possible positions. For the second $N-1$, for the third $N-2$, and so on. The total number of arrangements is therefore given by:

$$N \times (N-1) \times (N-2) \times (N-3) \text{ x...} \times 1$$

In other words by the factorial of N (or $N!$). Thus, for 25 cards there are 25! possible arrangements. In the worst case (that is, if there is no legal arrangement), the computer will have to go through all 25! arrangements. Let us assume that we have a computer which can construct and evaluate a million arrangements per second. This computer will take well over 490 *billion years* to do this. Clearly, an algorithm with this kind of time complexity is not acceptable. Although there exist algorithms for the Monkey Puzzle Problem that will do slightly better than the naïve one outlined above, there does not exist one that makes it worth your while to hang around waiting for solutions, since in the worst case they still take thousands of years just for the small 5×5 version.

Table 1 (from Harel, 1987) gives the time consumption of various hypothetical algorithms for the Monkey Puzzle Problem running on a computer capable of carrying out a million instructions per second. As can be verified without too much effort, waiting times are reasonable as long as the number of instructions to be carried out to obtain a solution stands in a polynomial relation to the size of the problem. By contrast, waiting times for problems which do not have a polynomial-time solution get out of hand rather quickly. And for obvious reasons such problems are often referred to as being 'computationally intractable'.

2.3. Easy, hard and still harder problems: Complexity classes

A lot of everyday problems like alphabetical sorting can be solved in polynomial time on a normal computer (that is in time n^j or less, for some constant j). Such problems are called *tractable*, for fairly obvious reasons. Complexity theory calls this class P: the class of problems solvable in *P*olynomial time on a deterministic[1] Turing Machine (TM).

---

[1] A computer is said to operate deterministically if its actions are fully determined by its input and its internal state. Real life computers must operate deterministically because unfortunately magic oracles of the kind mentioned in the text do not exist.

Table 1
Time consumption of hypothetical solutions to the monkey puzzle problem (assuming one instruction per microsecond).

Function	N				
	10	20	50	100	300
N^2	1/10 000 second	1/2500 second	1/400 second	1/100 second	9/100 second
N^5	1/10 sec.	3.2 seconds	5.2 minutes	2.8 hours	28.1 days
2^N	1/1000 second	1 second	35.7 years	400 trillion centuries	a 75-digit number of centuries
N^N	2.8 hours	3.3 trillion years	a 70-digit number of centuries	a 185-digit number of centuries	a 728-digit number of centuries

For comparison: the 'Big Bang' was approximately 15 billion years ago.

Other problems, like the Monkey Puzzle Problem, take longer, no matter how hard one tries to write an algorithm that will do better. Other famous examples of intractable problems are the Travelling Salesman Problem and Satisfiability (or SAT). A particular variant of SAT called 3SAT goes likes this. Given an arbitrary Boolean formula like

$$(a' \vee b \vee c) \wedge (b' \vee c' \vee a) \wedge (a \vee b \vee c') \wedge (a' \vee b' \vee c') \wedge (a \vee b' \vee c)$$

is there an assignment of true and false to the propositional letters such that the whole expression is true (where a is true if a' is false and vice versa)? 3SAT is difficult because all you can do to solve an instance of it is guess an assignment of truth values to the literals and test for satisfaction. This means that *in the worst case* one may have to try out all possible assignments. With n binary valued variables in an arbitrary formula, one will end up testing 2^n truth-value assignments. Since the number of different variables can rise with the length of the formula, the complexity of 3SAT is proportional to 2^n, or exponential time.

The Monkey Puzzle Problem and 3SAT have one very interesting property in common. Although finding a legal arrangement of the tiles of a Monkey puzzle may take an incredible amount of time in the worst case, checking whether a given arrangement of tiles is a legal one is entirely straightforward. Similarly, given any 3SAT formula, we can verify quite quickly whether a given assignment of truth values to the propositional letters is a good or a bad one. All we have to do is walk through the formula from left to right checking whether the assignment makes each clause true. It should be intuitively clear that this can be done in polynomial time (proportional to the length of the formula times the length of the guess list).

Taking this one step further, suppose we had a computer that contained an oracle which could guess a satisfying assignment of truth values for a 3SAT instance if

there is one. Clearly, such a computer could solve 3SAT in polynomial time (because it could check for satisfaction in polynomial time). Such a guessing computer is called *nondeterministic* and the class of problems solvable by a *N*ondeterministic TM in *P*olynomial time is called *NP*.

Complexity theorists have discovered many problems like 3SAT, which have the property that they only seem to have exponential-time *deterministic* solutions, but which have efficient (i.e. polynomial-time) *nondeterministic* solutions. There is no proof for an exponential time lower bound on the complexity of these problems, but it is strongly suspected that $P \neq NP$. All problems in *P* are of course a subset of those in *NP*. But problems like 3SAT summarize the complexity of a whole class of problems which are in *NP* and not known to be in *P*. This class is held together by a method called *reduction*, which I discuss below. The problems in this class are called NP-hard, because they are as hard as any problem in *NP*. If an NP-hard problem is in addition known to be solvable by the hypothetical guessing computer in polynomial time, as is 3SAT, then it is called NP-complete.

Another complexity class that will be relevant to our discussion is *PSPACE*, the class of problems solvable in polynomial *space* on a deterministic TM. *PSPACE* contains *NP* because polynomial space allows the simulation of an entire *NP*-computation. Intuitively, this is because space, unlike time, can be reused. While *NP*-complete problems can be verified in polynomial time, problems in *PSPACE* probably cannot (although there is no proof for this). A problem which is *PSPACE*-hard and which is furthermore known to be in *PSPACE* is called *PSPACE*-complete. Just as 3SAT is a typical *NP*-complete problem, Quantified Boolean Formulas (QBF) is a typical *PSPACE*-complete problem. As its name already indicates, QBF combines Boolean expressions with the universal ($\forall$) and existential ($\exists$) quantifiers. An instance of the QBF problem is given by a quantified Boolean formula of the form

$$Q_1 y_1 Q_2 y_2 \ldots Q_m y_m F(y_1, y_2, \ldots, y_m)$$

where Q_i is either a universal or an existential quantifier ranging over the set 'true, false' and none of the variables y_i is free. The problem is to determine whether the quantified formula is true. A QBF problem contains an arbitrary number of alternating existential and universal quantifiers. This makes even the verification of a satisfying assignment very hard. Intuitively, this can be understood as follows. A 3SAT problem over *m* variables corresponds to a restricted QBF, which contains only existential quantifiers. Verifying the satisfiability of such a formula is straightforward, because the truth of an existentially quantified formula can be verified by finding *one* true instance. Given a satisfying assignment, all one has to do is make one pass through the QBF checking that every conjunct in the formula comes out true. But a satisfying assignment for the general QBF problem is much more difficult to verify, because the truth of a universally quantified formula can only be verified by checking that *all* possible values of the variable make the formula true. Suppose the length of the QBF is *m*. Then the number of universal quantifiers is proportional to *m*. Therefore, the total number of assignments that have to be checked for verification is proportional to 2^m. Verification of a QBF could therefore take exponential time.

Finally, the classes *EXP* and *EXP-POLY* contain problems that are *provably* exponential time. These classes therefore include all the problems in both *NP* and *PSPACE*, for which such proofs do not exist.

2.4. Problem size

Complexity theory is built on the assumption that problems can grow arbitrarily in size. This is a far from trivial assumption for the natural language recognition problems that we will be looking at. After all, our sentences are bounded in length by such factors as memory limitations and the finite duration of a human life. Since natural language problems are always bounded in size, it would seem that any complexity results are doomed to be irrelevant. As Barton et al. (1987: 20) point out, this argument "explodes on complexity theoretic grounds just as it does in introductory linguistics classes". The point being that once we have established the principles that underlie the structure of sentences of reasonable length, there doesn't seem to be any natural bound on the application of those principles. In other words, the grammar generates sentences of indefinite length. There are practical reasons why sentences exceeding a certain length are never produced but those reasons bear no relation to the underlying structure of the language.

Very much the same reasoning applies in the case of computational complexity. Problems of bounded size always have an efficient solution.[2] This is because we could calculate all the solutions in advance and put them in a table. But this leads to totally counterintuitive results. Barton et al. discuss the example of checkers. They point out that complexity theorists study

> "the playing of checkers on an arbitrary $n \times n$ board, rather than 'real' checkers, because they can use complexity theory to study the structure and difficulty of the problem. The problem with looking at problems of bounded size is that results are distorted by the boring possibility of just writing down all the answers beforehand. If we study checkers as a bounded game, it comes out (counterintuitively!) as having no appreciable complexity – just calculate all the moves in advance – but if we study $n \times n$ boards, we learn that checkers is computationally intractable (as we suspected)." (Barton et al., 1987: 20)

So by restricting a problem to some bounded size we are obscuring the true complexity (and the *sources* of complexity) of that problem. Therefore, the idealization to problems of unbounded size is a necessary and justified one.

2.5. The reduction technique

To show that one problem is as difficult as another, complexity theorists use a technique called reduction. Here is how it works. Take a problem of known complexity, like 3SAT. Then construct an efficient mapping from instances of the problem of known complexity to instances of the new problem, preserving yes/no

---

[2] In fact, complexity theory treats such problems as having zero complexity.

answers.[3] As before, efficient means polynomial time. It follows that the new problem must be intractable as well. It is easy to see why. Suppose the new problem has a polynomial-time solution. Then the problem of known complexity, which we assumed was intractable, is solvable in polynomial time as well (namely by mapping it into the new problem). This is because the composition of two polynomial time functions is itself a polynomial time function:

Known hard →	Polynomial →	New
problem	time	problem
instances	reduction	instances

If in addition an NP-completeness proof is desired, one should demonstrate that the new problem can be solved in polynomial time by the hypothetical guessing computer.

2.6. What's in the next sections?

In sections 3 and 4 we look at how complexity theory can be used to obtain an understanding of the complexity of language computations. In section 3 we use complexity theory to obtain results about the complexity of so-called Universal Recognition Problems (URPs) for Lexical Functional Grammar, Word Grammar, and the Lasnik–Saito and Barriers models of transformational grammar. In section 4 we turn to Ristad's (1993) approach to the study of the computational complexity of language production, comprehension and acquisition.

3. Linguistic theory and computational complexity

3.1. What is a Universal Recognition Problem?

Barton et al. (1987) focus on grammatical recognition problems that have the following form:

> Given a grammar G (in some grammatical framework) and a string α, is α in the language generated by G?

In what follows we consider several recognition problems of this form. This kind of problem allows one to study the complexity of an entire class of grammars, namely those specified by some grammatical theory (like Government Binding theory or Generalized Phrase-Structure Grammar). For this reason, it is also known as the Universal Recognition Problem (URP) for a linguistic model.

---

[3] For the purposes of complexity theory, problems are always (re-)phrased as yes/no-questions. The reduction must be carried out in an 'answer-preserving' way, so that we can solve instances of the old problem by solving the corresponding instances of the new problem.

This should be contrasted with what is the tradition in weak generative capacity analysis, namely, to state such problems as fixed language recognition problems:

Given a string α, is α in some independently specified set of strings?

While these problems look very similar they are not, since in the fixed language recognition problem we are free to choose the grammar any way we like. Therefore, the recognition problem contains just one variable, namely the sentence to be recognized. By contrast, in the URP we have two variables: the string to be recognized plus the grammar. As a consequence the grammar size appears as a parameter in the complexity formulas. As one might expect, the URP is harder because the recognition algorithm must be able to accommodate any grammar. The Fixed Language Recognition problems are easier, because one is permitted to vary the grammar at will to get the most efficient algorithm possible.

The complexity of the URP for a linguistic theory is a direct measure of how difficult it is to *parse* the languages generated by the class of grammars specified by the theory. Parsing is the process of using a grammar to assign a syntactic structure to a string of words. It is widely assumed that parsing is an integral component of the computations carried out by language users[4] and that apart from the constraints imposed by performance limitations, human sentence parsing is extremely efficient. As a result of this generally held view, a sizable section of psycholinguistic research in sentence processing is concerned with the question why parsing sometimes *fails*. One notable exception to the view that parsing is easy appears in Chomsky and Lasnik (1991: 4), who point out that parsing is often "slow and difficult, or even impossible, and it may be in 'error' in the sense that the percept assigned (if any) fails to match the [structural description] associated with the signal ...". They take the view that language is not "readily usable or 'designed for use'. The subparts that are used are usable, trivially; biological considerations lead us to expect no more than that".

Although the assumption that parsing must be very efficient has never been based on anything but the observation that normal humans seem to be rather good at comprehending language, they have the air about them of being common-sense. For this reason, it may come as a bit of a shock that it has been shown for all well-established linguistic theories of human syntactic knowledge that their associated URP is NP-hard. What this means is that not one of these theories guarantees efficient parsing! We will shortly look at some of the proofs for these results. Here we first ask how one goes about constructing such a proof.

We already know that complexity results rely on the reduction technique. We must therefore choose a known NP-hard problem and translate this known problem into the URP for the linguistic theory we are interested in. But how does one know which known problem to pick? The general idea is to spot a similarity between the known and the new problem. All the reductions we will consider in this section will be from 3SAT. Recall that 3SAT is difficult because of the combined effect of what

[4] An assumption which is not shared by Ristad, as we will see in section 4.

might be called lexical ambiguity (each propositional letter can be either true or false) and the need for agreement/consistency (propositional letters must have consistent truth assignments) throughout the formula to be tested for satisfaction. As natural languages exhibit lexical ambiguity (is *police* a noun or a verb?) and agreement phenomena over unbounded distances as well (for instance Case agreement between a *wh*-phrase and its associated gap), we are led to suspect that descriptively adequate linguistic theories will contain mechanisms that are precisely those needed to solve the 3SAT problem.

It is not difficult to find examples of natural language phenomena that can give rise to computationally very difficult problems. For instance, the combination of lexical and structural ambiguity in (1) makes it quite difficult to determine whether it is a grammatical sentence or not:

(1) police police police police police

Interestingly, it is not difficult to verify that (1) *is* grammatical if one is given some solution to the parsing problem it represents, such as the one in (2):

(2) police_N police_V police_N [(who_i) police_N police_V t_i]

Recall that this is the hallmark of *NP*-complete problems: difficult to solve, easy to verify.

The reductions we will look at rely on the mechanisms for agreement and ambiguity in a particular linguistic theory to solve 3SAT. The proofs will be easier to follow if one gives some thought to what is required to solve the 3SAT problem. In general, to solve 3SAT we need three 'components':

(i) A truth-setting component which assigns a truth value to propositional letters;
(ii) A satisfaction component which forces one literal in each clause in the problem instance to be true;
(iii) A consistency component which forces the propositional letters to have consistent truth assignments.

Given these three components it will be possible to decide whether a given problem instance is satisfiable (a basic generate-and-test method will do).

To reduce 3SAT to the URP for some grammatical framework, we must first exhibit a mapping from 3SAT problem instances to sentences for recognition. In the proofs that follow, this is achieved simply by erasing the ' $\wedge$ ' and ' $\vee$ ' symbols and the brackets in the 3SAT problem instance to produce a sentence for recognition. We then demonstrate how to construct a grammar in the particular linguistic theory that will generate such a mapped sentence iff the corresponding 3SAT instance is satisfiable. For the reduction to be any good, we must show that it can be carried out in polynomial time. What building the grammar boils down to is that one demonstrates how the linguistic framework allows one to realize the three components listed above.

3.2. LFG-recognition is NP-hard (Barton et al., 1987)

3.2.1. Outline

What follows is my rendition of Barton et al.'s original proof, which is based on the insight that the essential ingredients of a solution to 3SAT can be very easily realized in LFG. In particular, an LFG can enforce feature agreement across arbitrary distances in a c-structure tree. This is a direct result from (i) the possibility to pass any feature up a c-structure tree (using the f-structure equation ($\uparrow$=$\downarrow$)) and (ii) the presence of the constraint of functional coherence, which guarantees consistent assignment of feature values.

The proof demonstrates how an LFG, together with an appropriate sentence to be tested for LFG membership, can be constructed in time polynomial in the size of an input 3SAT instance. The LFG generates this sentence iff the original 3SAT instance is satisfiable.

3.2.2. The proof[5]

We start out by posing the universal recognition problem:

Given an arbitrary lexical-functional grammar G and a sentence α, is $\alpha \in L(G)$?

Theorem: LFG Recognition (LFGR) is NP-hard.

Proof. The reduction is from 3SAT. Given a 3SAT instance F of length n using the m variables $x_1 ... x_m$, reduce 3SAT, a known NP-complete problem to LFGR in polynomial time.

To construct the LFGR problem, we map F into a sentence α to be tested for LFG membership. This involves erasing the parentheses and the $\vee$ and $\wedge$ symbols in F. We are left with just the string of literals.

Next, we build the LFG that we will use for the membership test. We must write an LFG with the following properties:

(i) It must reproduce the clause structure of 3SAT;
(ii) It must force at least one literal in each clause to be true;
(iii) It must use LFG agreement machinery to enforce coherent truth assignments.

The grammar will contain the following c-structure rules:

1 S → SS
2 S → TTT
3 S → TTF
4 S → TFT
5 S → FTT
6 S → TFF

[5] Although I have not indicated this in the text, I quote liberally from the source throughout the proof.

7 $S \rightarrow FFT$
8 $S \rightarrow FTF$

These rules reproduce the required clause structure and will force one literal in every clause to be true. We can use the same grammar for all problem instances. Finally, we add lexical entries, one pair for each literal x_i and x'_i. The lexicon varies with the particular 3SAT instance:

T:	x_i	F:	x_i
	$(\uparrow x_i = T)$		$(\uparrow x_i = F)$
F:	x'_i	F:	x'_i
	$(\uparrow x_i = F)$		$(\uparrow x_i = T)$

We need four entries per propositional letter to represent the following four cases: (i) x_i=T (pass up the feature x_i=T), (ii) x_i=F (pass up the feature x_i=F), (iii) x'_i=T (pass up the feature x_i=F), (iv) x'_i=F (pass up the feature x_i=T). It follows that we need $4m$ entries in all.[6]

Finally, we require the feature set represented at a given node to be unified with the feature set represented at its mother node. In LFG this operation is represented by the notation $(\uparrow=\downarrow)$ on a node in the c-structure tree. We therefore add the LFG $(\uparrow=\downarrow)$ annotation to all right-hand side constituents in rules 1–8, so that all features associated with the f-structure built up at each node will be unified with the features of its mother. In this way we will be able to enforce agreement between terminals over unbounded distances:

$S \rightarrow$	S	S	
	$(\uparrow=\downarrow)$	$(\uparrow=\downarrow)$	
$S \rightarrow$	T	T	T
	$(\uparrow=\downarrow)$	$(\uparrow=\downarrow)$	$(\uparrow=\downarrow)$

Plainly, the whole construction takes time polynomial in the size of the formula F: The mapping from F to α can be carried out in a linear left-to-right scan of the input formula. G also takes polynomial time to build; the c-structure and f-structure annotations are fixed. The lexicon, being only of size $4m$ can be built in time polynomial in the length F. We can keep track of what variables we have already seen in a list that we rescan at most a polynomial number of times. Thus the total time to construct the LFG is polynomial in the size of F.

Finally, we show that F is satisfiable iff $a \in L(G)$. Suppose that $a \in L(G)$. We must show that F is satisfiable. If $a \in L(G)$, then by the c-structure rules for G, there must be a derivation line of T and F preterminals that directly derives a. But this means that each preterminal triple contains at least one T, and assigning **true** to the literal corresponding to this lexical item yields a satisfying assignment (each clause

[6] Remember that m is the number of different propositional letters in the 3SAT problem instance.

contains at least one true literal). Functional coherence guarantees consistent assignments.

Now suppose that F is satisfiable. Then at least one literal in each clause triple must be true. We must exhibit a valid LFG derivation corresponding to this satisfying assignment. We choose the context-free expansion that expands out to the appropriate pattern of Ts and Fs; this can clearly be done. Then we select lexical entries that correspond to a, from left to right, ensuring that we pick the proper entry according to whether a lexical item is dominated by T or F. This is clearly a valid sentence in G, because it guarantees functional coherence.□

3.2.3. Interpretation of the complexity result for LFG

Barton et al. (p. 112) point out that "the LFG complexity result ... arises because arbitrary lexical ambiguity coupled with arbitrary feature cooccurrence restrictions is a computationally deadly combination". They then discuss Williams (1984), who has suggested that many of the problems associated with LFG's f-structures arise from its failure to observe X′-restrictions, and point out that his suggestion to import X′-theory into the f-structure language would suffice to eliminate the offending source of complexity. Williams has pointed out that the absence of a locality constraint on f-structures allows an LFG to express unnatural forms of feature agreement. He mentions as an example that in LFG a verb could require the object of a more deeply embedded verb to be plural. But if f-structures obeyed X′-theory, then hierarchical unification of features would be blocked by maximal projections. Only the features of the head of a phrase would be accessible for agreement rules.

The authors also point out a second way around the complexity result, which is to rely on performance factors, such as locality principles for parsing, to salvage the complexity of LFG parsing. In particular, they suggest that assuming an LR-type parser[7] would have the effect that in cases in which global lexical ambiguity arises, the resulting sentences will be difficult or impossible to process.

It seems to me that Williams's example above favours the former view. Parsing limitations affect the normal behaviour of our language comprehension system, but that does not mean that in general we cannot give grammaticality judgements for sentences that are difficult to parse. Take the case of sentences with multiple centre-embeddings: these are very difficult to parse, but with pen-and-paper a normal speaker/hearer can determine whether they are grammatical or not. But there is no question whether a language in which a verb requires the object of a more deeply embedded verb to be plural is difficult to parse or not: it appears that languages with such agreement processes do not exist. On the view that such languages do not exist because they are difficult to parse, we would similarly expect centre-embedding to be totally absent in natural languages, which it clearly is not.

A question that might be raised at this point is whether the proof tells us anything at all about the parsability of the whole class of LFGs, given that it relies on the construction of a particular LFG that bears no resemblance to an LFG for any natural

[7] A deterministic parser with lookahead similar to Marcus's (1980) Parsifal.

language. Or to put the same question in a slightly different form: does the particular LFG constructed for the proof go short for the whole class of LFG grammars? There is a clear answer to this question. The proof does not demonstrate that there are no efficiently parsable LFGs. Indeed, it is likely that most LFGs, including those that generate known natural languages, *are* efficiently parsable. The point of the proof is, however, that the LFG theory certainly does not guarantee efficient parsability. If there are no natural languages that require a grammar with properties similar to the ones exploited in the proof, then LFG makes that look like an accident. As discussed above, this accident might then be attributed to performance factors or may alternatively be taken to betray a lack of constraint in the linguistic theory (see Fig. 1).

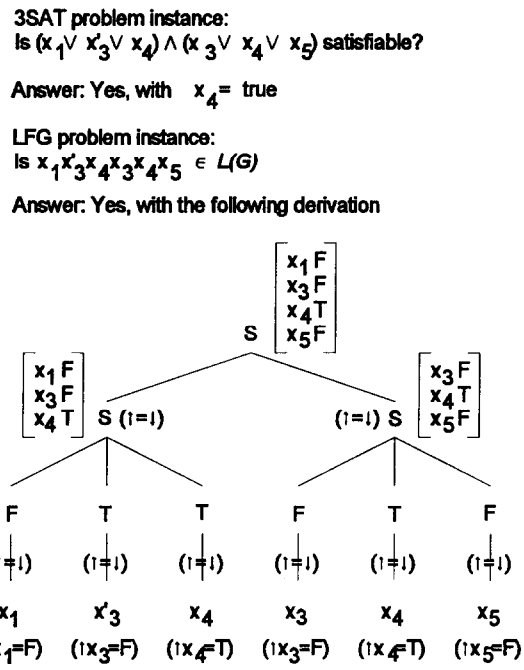

Fig. 1. The reduction described in the text produces this LFG derivation on a sample 3SAT instance (after Fig. 4.1 in Barton et al., 1987).

3.3. WG-recognition is NP-hard (van de Koot, 1992)

3.3.1. Word Grammar

Word Grammar (WG – Hudson, 1984, 1990) is a kind of dependency grammar: the structure of a sentence is given by a set of dependencies, not by a constituent structure. WG is essentially lexicalist in that the internal structure of a word can be generalized to act as the basis for generating syntactic structures. An essential rela-

tion in the theory is that of feature inheritance. This relation holds between an instance and the *model* of which it is an instance. For example, *run* is an instance of the category *verb* and a *verb* is an instance of the category *word*. "Since every word is an instance of some model and every word (or word-type) is, ultimately, an instance of the category *word*, the relation of instantiation provides a hierarchy which brings together every word in the language" (Hudson, 1984: 14). WG imposes on top of this hierarchy a set of statements or generalizations. For instance, one such generalization could apply to the word type *verb* and will be shared by any instance of the model verb (by default inheritance). Furthermore, the grammar may contain statements to the effect that a word in a given dependency configuration inherits certain features. For instance, we might specify that a word inherits certain features of a word that is dependent on it.

For the purpose of the proof that follows this is about all we need to know about Word Grammar. The proof establishes the computational intractability of the recognition problem for WG. It is very similar to the NP-hardness proof for LFG.

3.3.2. The proof

We want to show that WG recognition is NP-hard. Intuitively that this should be the case is easy to see. WG has a feature-inheritance mechanism that is as powerful as the LFG agreement machinery. All we need to do is to find a way to create a WG truth-satisfaction component that mimics the effect of the LFG c-structure rules.

We begin by defining the WG recognition problem with respect to an input sentence α and a WG G as follows:

The WG Recognition Problem is: given a sentence α and a WG G, is $\alpha \in L(G)$?

We then carry out the reduction.

Theorem: WG Recognition (WGR) is NP-hard.

Proof. The reduction is from 3SAT. Given a 3SAT instance F of length n using the m variables $x_1 \ldots x_m$, reduce 3SAT, a known NP-complete problem to WGR in polynomial time.

To construct the WGR problem, we map F into a sentence α to be tested for WG membership. This involves erasing the parentheses and the $\vee$ and $\wedge$ symbols in F. We are left with just the string of literals.

Now we build the WG that we will use for the membership test. The grammar will vary slightly with the problem instance (details follow). Finally, we add lexical entries, one pair for each literal x_i and x'_i. The lexicon varies with the particular 3SAT instance.

The WG must provide a method for reproducing the clause structure of 3SAT. Since WG does not have CF base rules we cannot get away with the simple method for LFG. Instead, we exploit WG's default inheritance scheme and define the WG to have three major category types: cat-1, cat-2 and cat-3, which are all a kind of word.

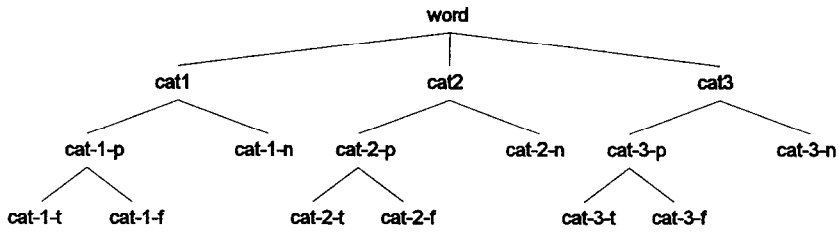

Fig. 2. The category hierarchy used in the proof.

cat-1 always selects cat-2, cat-2 always selects cat-3, and cat-3 always selects cat-1 or no complement at all.[8]

Next we take care of the truth-satisfaction component. In each clause (cat-1,cat-2,cat-3) either cat-1 is true or it makes cat-2 promise to make the clause true. Similarly, if cat-2 has promised cat-1 that it will make the clause true, then either cat-2 itself is true or it makes cat-3 promise to make the clause true. Clearly, if a category is true, it must leave its complement free to be either true or false. To get this idea to work, we define various subcategories for cat-1, cat-2 and cat-3. These subcategories will encode what each category has been made to promise. For instance, a cat-2-p is a category which has promised to make the clause true, while a cat-2-n is a category that has promised nothing.

Inheritance of category membership

We will make use of a set of categories ordered in the following *isa*-hierarchy:

cat-1-t isa cat-1-p.
cat-1-f isa cat-1-p.
cat-1-p isa cat-1.
cat-1-n isa cat-1.

cat-2-t isa cat-2-p.
cat-2-f isa cat-2-p.
cat-2-p isa cat-2.
cat-2-n isa cat-2.

cat-3-t isa cat-3-p.
cat-3-f isa cat-3-p.
cat-3-p isa cat-3.
cat-3-n isa cat-3.

[8] The idea for the strategy I use here was taken from Ristad's (1991) NP-hardness proof for GB-recognition. I discuss this proof in the following section.

cat-1 isa word.
cat-2 isa word.
cat-3 isa word.

Selection of complements

We now specify (in WG notation) that cat-1 and cat-2 (which correspond to the first and the second disjunct in a 3SAT conjunct) take one and only one complement, whereas cat-3 (which corresponds to the third disjunct in a 3SAT conjunct) takes one complement (namely the first disjunct of the next conjunct) or no complement at all (if it corresponds to the third disjunct in the last conjunct of the 3SAT formula).

cat-1 has [1-1] complement.
cat-2 has [1-1] complement.
cat-3 has [0-1] complement.

Next we organize the selection of complements in such a way that of every three consecutive words in the WG sentence corresponding to the 3SAT formula at least one word will come out true. For instance, we specify that the complement of a cat-1-t (a word corresponding to a first disjunct which is true) is a category which promises nothing (a cat-2-n), whereas the complement of a cat-1-f (a word corresponding to a first disjunct which is false) is a cat-2-p (a category which promises either to be true itself or to make its complement promise to be true). To give one further example, the complement of a cat-2-f (a category corresponding to a second disjunct which is itself false and also follows a false first disjunct) must be a cat-3-t (a category corresponding to a true third disjunct). If a cat-3 has a complement, then this complement must of course promise to make the next clause true.

type of object of cat-1-t = cat-2-n.
type of object of cat-1-f = cat-2-p.
type of object of cat-2-t = cat-3-n.
type of object of cat-2-f = cat-3-t.
type of object of cat-2-n = cat-3-n.
type of object of cat-3= cat-1-p.

Features and feature inheritance

WG allows a general statement of feature agreement, which is very similar to the LFG f-structure notation ($\downarrow$=$\uparrow$). What we want is for a head to share all the features of its complements:

feature of word = feature of complement of it.

In WG a word can only inherit a feature for which it is specified. This is to block certain instances of feature inheritance. For the proof to work, we must ensure (i) that each word can inherit any feature and (ii) that each x_i is systematically associated with a feature f_i.

We take care of (i) as follows

word has f_1
...
word has f_m

We take care of (ii) in the lexicon. For every x_i we add the following four entries, where the operator 'slash-colon' denotes disjunction:

Lexicon

x_i isa {/: cat-t, cat-n}.	x'_i isa {/: cat-f, cat-n}.
f_i of x_i = true.	f_i of x'_i = true.
x_i isa {/: cat-f, cat-n}.	x'_i isa {/: cat-t, cat-n}.
f_i of x_i = false.	f_i of x'_i = false.

The first entry for x_i specifies that it is of category *cat-t* or of category *cat-n* and that it carries the feature f_i with the value **true**. The second entry for x_i specifies that it is of category *cat-f* or of category *cat-n* and that it carries the feature f_i with the value **false**. It follows that if x_i is of category *cat-n* it can be either **true** or **false** as required. The other two entries take care of the complementary case x'_i. Thus these four lexical entries capture all the ambiguities associated with an x_i.

The whole construction takes time polynomial in the size of F. The mapping of F to a string α for WG recognition can be done in one pass through F. G also takes polynomial time to build. The specification of category types is fixed across problem instances. The number of features is linear in the number of variable names. The lexicon can be constructed in time polynomial in the length of F, because its size is proportional to $4m$. We can keep track of the variables that we have already seen in a list that we rescan at most a polynomial number of times. Thus the total time to output the corresponding WG is polynomial in the size of F.

We must now show that F is satisfiable iff $\alpha \in L(G)$. Suppose that $\alpha \in L(G)$. We must show that F is satisfiable. If $\alpha \in L(G)$, then by the rules for category membership and selection there must be a derivation of a string of true and false words such that of each triple of words at least one word is true. This means that assigning the value **true** to the literal corresponding to this lexical item yields a satisfying assignment (each clause contains at least one true literal). WG agreement guarantees consistent assignments.

Now suppose that F is satisfiable. Then at least one literal in each clause must be true. We must exhibit a valid WG derivation corresponding to this satisfying assignment. We simply choose the pattern of selectional dependencies that results in the appropriate pattern of true and false words. This clearly can be done. Then, working from left to right we select words corresponding to those in α, the mapping of F, making sure to pick the proper entry depending on whether the item has to be true or false. This is clearly a valid sentence in G, because it satisfies the WG-derivation and

because consistency of variable feature values guarantees satisfaction of WG agreement. This completes the proof.□

3.3.3. Interpretation of the complexity result for WG

The sources of complexity in LFG and WG are basically identical and the comments in section 3.2.3 apply here as well. Both in the case of LFG and in the case of WG we saw that the combination of lexical ambiguity and arbitrary feature cooccurrence restrictions has a severe computational impact. In WG unrestricted hierarchical feature inheritance takes the place of unrestricted hierarchical feature unification. But while in the case of LFG there is a natural candidate for a locality condition on f-structures (namely X′-theory), it is not so easy to see how feature inheritance in WG can be suitably restricted. After all, there are no phrases in WG, just head-head relations. It is therefore impossible to import something like X′-theory into WG. (See Fig. 3.)

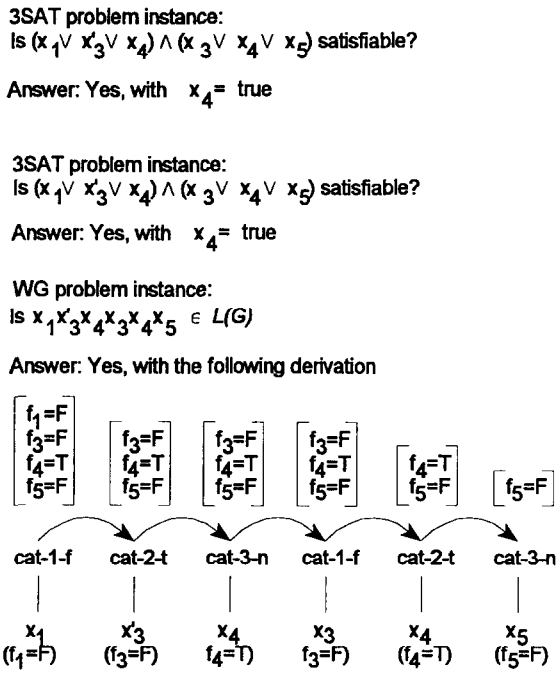

Fig. 3. The reduction produces this derivation on a sample 3SAT instance.

3.4. The complexity of the Barriers and Lasnik–Saito models (Ristad, 1991)

Ristad (1991) presents an NP-hardness proof for the URP for a class of transformational grammars that permit a particular construction, known as a 'stair'. He argues that the version of GB known as the Barriers model of grammar (Chomsky,

1986) permits a stair and that therefore its associated URP is NP-hard. The same is then claimed to be the case for the Lasnik–Saito model of grammar (Lasnik and Saito, 1984). I first recapitulate these proofs, using substantial excerpts from Ristad (1991). I then argue that the proof that the URP is NP-hard for models that permit a stair is flawed. I also show that the linguistic model to which it applies is not a correct representation of either Lasnik and Saito (1984) or Chomsky (1986), since neither of these models in fact permits a stair. I conclude that Ristad (1991) has not demonstrated that the URP is computationally intractable for these models. Finally I discuss the implications of this result.

3.4.1. The stair construction

The proof that we are about to review is based on the idea that the components required to solve 3SAT can be realized in a transformational model in the form of a construction called a stair. Each stair will be used to represent one propositional letter of a 3SAT instance. Three stairs will be needed to represent one clause. A stair is a recursive structure: a stair selects another stair. Within a group of three stairs a chain of selectional dependencies will guarantee that one of the three stairs is assigned the value **true** in virtually the same way in which this was done in the WG proof. So, as was the case there, selection and agreement are correlated. But whereas feature passing methods were used to ensure consistency of truth assignment in the case of LFG and WG, movement and specifier-head agreement perform this function in the transformational theory. Thus, every stair representing some x_i will have available a specifier into which another stair representing an x_i can be moved to check, under specifier-head agreement, that the truth values represented in the two stairs are identical. Because consistency of truth value assignment must be checked for all stairs, every stair must somehow be forced to move to a position of specifier-head agreement. In other words, a stair must have the property that it moves obligatorily. This can be achieved by setting things up in such a way that a stair θ-marks but does not case-mark the stair it selects. Of course it must be guaranteed that the movement of a stair is also permitted and since one stair must be able to move out of another, a stair should be transparent for extraction. A stair is therefore defined as follows:

Definition. A *stair* is an underlying form U_i with the following structure:

1. *Recursive structure.* U_i contains another stair U_{i+1}.
2. *Selection and agreement are correlated.* U_i contains a morpheme μ_i that selects U_{i+1}. Local affixation rules will morphologically merge the head of U_i with the morpheme μ_i, thereby correlating selectional properties of μ_i with the agreement features of U_i in the lexicon.
3. *Undergoes obligatory movement.* U_i selects and assigns a θ-role to U_{i+1}, but does not assign it case. Therefore, U_{i+1} is a properly governed argument that undergoes obligatory movement in order to satisfy the case filter. (The same will be true for U_i.)
4. *Transparent to extraction.* U_i allows nodes that can be moved out of U_{i+1} to also be moved out of U_i.

5. *Contains a landing site.* U_i contains a specifier position that is assigned case. The head of U_i will agree with its specifier; therefore only stairs that agree with the head of U_i can be moved to this specifier position.

The next step is to show that stairs can enforce the 3SAT constraints.

Lemma. Stairs can enforce the 3SAT constraints.

Proof. The idea of this subproof is to represent negation as a morpheme (recall that one of the 3SAT constraints is that x_i and its negation x'_i have opposite truth values) and to encode the truth value of variables in syntactic features. Clausal satisfaction will then be enforced at DS by means of an appropriate chain of selectional dependencies and consistency of truth assignments will be enforced at SS through forced movement to positions of spec-head agreement, as just discussed.

The DS consists of one stair per literal. Thus, each clause

$$C_i = (a_i \vee b_i \vee c_i)$$

will be represented by the three stairs $U_{i,a}$, $U_{i,b}$, $U_{i,c}$ (see Fig. 4). The selectional constraints of the three stairs will ensure that at least one of them is true, although (as before) lexical ambiguity will prevent us from knowing which. This can be done as follows. $U_{i,a}$ must promise to make C_i true, either by being itself true or by forcing $U_{i,b}$ to promise to make C_i true. To fulfill its promise, $U_{i,b}$ must either be true or select a true stair $U_{i,c}$. Affixes in the lexicon will negate or preserve variable truth values, according to whether the corresponding formula literal is negative or positive.

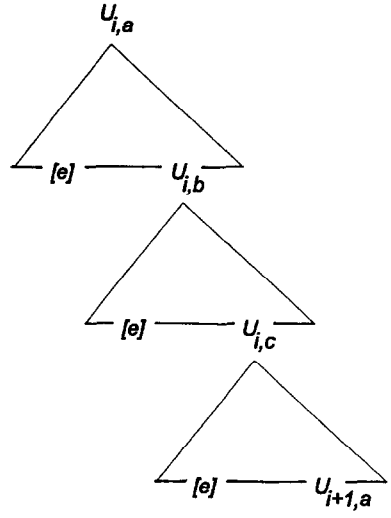

Fig. 4. A 3-stair.

Then, looking at the tree from right to left, each stair is moved to the specifier position of the closest stair of the same variable. Recall that each stair has the property that it moves obligatorily. As a result, at SS, the specifier position of the stair that corresponds to the *i*th occurrence of a given variable contains the stair that corresponds to the $i+1$th occurrence of the same variable. These two stairs agree with each other by specifier-head agreement (recall that each stair has the property that it has a landing site which is a specifier that agrees with its head). In fact, at SS, all the stairs that correspond to literals of a given variable end up being contained in the specifier position of the stair that corresponds to the first occurrence of that variable. Consequently, all the stairs that correspond to literals of a given variable agree with each other by spec-head agreement at this level, as required in order to enforce consistent truth assignments. All clauses contain a true literal by DS-selection. Negation is performed by affixes. The formula is satisfiable iff the corresponding DS and SS are well-formed.□

As in the case of the proofs for LFG and WG that their associated URP is NP-hard, the proof that the URP for models that permit a stair is NP-hard is based on the complexity of a particular subproblem with respect to such a model. This subproblem is the same as it was in the LFG and WG proofs: given a partial syntactic representation R that yields a string of ambiguous or underspecified words, and a lexicon L containing ambiguous words, can the words in R be found in the lexicon L? Ristad refers to this subproblem as the *lexical resolution problem* (LRP) and he proves that the LRP is NP-hard for a model that permits a stair. But of course if a subproblem of a problem is NP-hard, then it follows that the overall problem is NP-hard. Therefore, a proof that the LRP for a model is NP-hard implies that the URP for that model is NP-hard as well.

Now using the stair construction and the fact that words may be ambiguous, Ristad proves the following theorem about the lexical resolution problem:

Theorem. The LRP is NP-hard in models that permit a stair.

Proof. By reduction to 3SAT. The input is a 3SAT formula f; the output is a lexicon L and a structure S containing underspecified words such that the words in S can be found in L if and only if f is satisfiable. The structure S will be built from f according to the stair construction discussed above. Two stairs will agree if and only if they correspond to literals of the same variable and have been assigned the same truth value. The words in the syntactic structure will be ambiguous only in the syntactic distinction that corresponds to truth value. One agreement feature encodes variable truth assignments, and another identifies Boolean variables. One non-agreement feature encodes literal truth values, and a second one keeps track of the promise in the chain of selectional dependencies. The stair construction ensures that the 3SAT constraints are satisfied by all permissible lexical choices for the words. □

The details of this proof, which are quite complex, can be found in Ristad (1988) and are reproduced verbatim in the Appendix. (See Figs. 5, 6, and 7.)

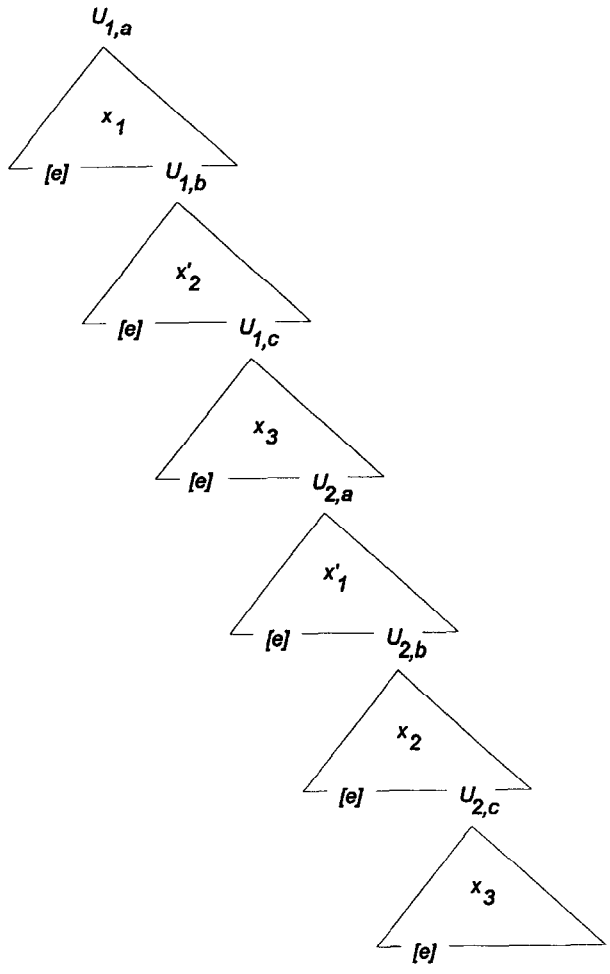

Fig. 5. On the input 3SAT instance $f=(x_1,x'_2,x_3)$, (x'_2,x_2,x_3), the DS in the figure is created to represent f. Each literal in f is represented by a stair construction. For example, the first literal of the first clause, x_1, is represented by the outermost stair construction, $U_{1,a}$. Selectional constraints are enforced at DS. They ensure that every Boolean clause contains a true literal.

Finally, Ristad shows that Barriers allows a stair.

Lemma. Barriers allows a stair.

In Fig. 8:

(a) V moves to I to N to correlate selection of NP_{i+1} and agreement features on I and N; movement is obligatory if I and N are affixes;

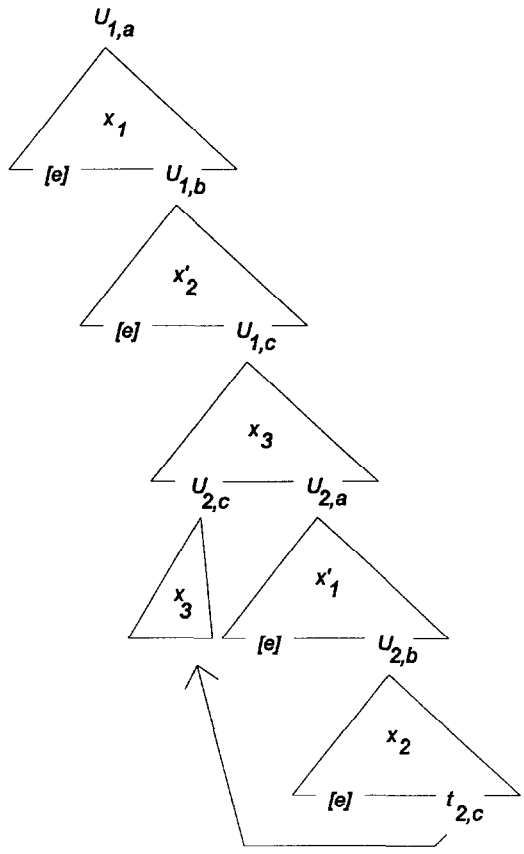

Fig. 6. This figure depicts the first movement transformation that is applied in the mapping of the DS in Fig. 5 to the SS in Fig. 7. The innermost stair ($U_{2,c}$, representing the last literal in f) moves to the specifier position of the third stair ($U_{1,c}$), leaving behind a trace $t_{2,c}$. This movement transformation relates the x_3 literal of the second Boolean clause to the x_3 of the first Boolean clause. Now both stairs agree, by specifier-head agreement; therefore, the corresponding literals of the formula variable x_3 will be assigned the same truth value, even though they appear in different clauses.

(b) I/V assigns case to its spec;
(c) movement of NP_{i+1} is obligatory because V assigns it a θ-role but no case;
(d) NP_i is transparent to extraction; NP_{i+1} can adjoin to VP and void its barrierhood because non-arguments may be freely adjoined to. From there it can raise further. The adjoined trace may be deleted after it has done its γ-marking duty, thus avoiding a Principle C violation.
(e) The internal IP_i contains a spec which acts as a landing site. The NP that moves into this spec will agree with I^0, the head of IP.

Theorem. The LRP is NP-hard in the Barriers model.

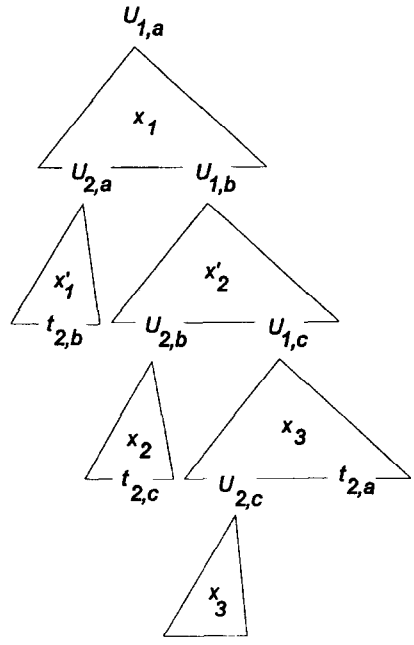

Fig. 7. This figure shows the SS that results from repeatedly applying movement transformations to the DS depicted in Fig. 6. Specifier-head agreement is enforced at SS. It ensures that all instances of a variable are assigned the same truth value.

Proof. The proof follows from the lemma that Barriers allows a stair and the theorem that the LRP is NP-hard in models that permit a stair. The lexicon contains ambiguous inflected nouns '[[V I] N]' that have undergone verbal incorporation.□

Theorem. The LRP is NP-hard in the Lasnik–Saito model.

Proof. The proof proceeds without alteration in the Lasnik–Saito (1984) model because in that model, theta-government suffices for proper government, and traces may be deleted after γ-marking.□

In the following two sections I discuss two problems with the proofs that the LRP is NP-hard in the Barriers and Lasnik–Saito models.

3.4.2. Problem 1: Stairs cannot enforce all 3SAT constraints

Although this is not immediately clear from the presentation in Ristad (1991), there is in fact a Case-theoretic problem with deriving the lemma that stairs can enforce the 3SAT constraints. If the lemma cannot be derived, then the proof that the LRP is NP-hard in models that permit a stair fails and consequently the NP-hardness proofs for the Barriers and Lasnik–Saito models fail as well.

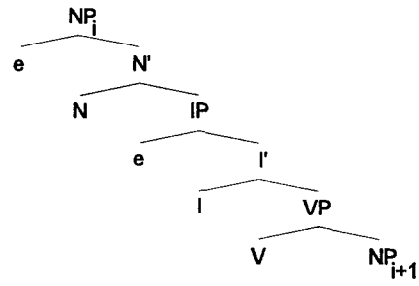

Fig. 8. A Barriers stair.

The lemma fails to be derived for the following reasons. It would seem that a stair-based representation for a 3SAT-instance containing two or more distinct propositional letters must violate the Case Filter. Suppose the original 3SAT problem instance has just 3 different propositional letters. The way any stair in the proof receives Case is by moving into a spec (recall that this is what forces movement in the first place). As movement is obligatory, we must guarantee that there is a case-marked landing-site for every U_i. Suppose our 3 propositional letters are x, y and z respectively. Suppose that some x was the leftmost propositional letter in the formula. Then the stair corresponding to that propositional letter will be the outermost stair and will therefore not require Case.[9] But what about the 'highest' instances of stairs for y and z? These will have *their* specs filled with stairs that moved into them to get case. But where do these 'highest' stairs which are not outermost move to themselves? The answer is that they have nowhere to go. Therefore these 'highest' non-outermost stairs must receive Case in-situ.

To illustrate this point, consider again Fig. 7. In the SS depicted in this figure neither $U_{1,b}$ nor $U_{1,c}$ receives Case, because neither has moved to a position in which Case is assigned. The DS positions these stairs occupy are Caseless (by the definition of a stair). Therefore, the structure should be ruled out by the Case Filter.

In the more detailed version of the proof in Ristad (1988) the problem noted here is avoided by assuming that

"[t]he leftmost literal of each formula variable is assigned case in situ by the formula literal that selects it; all other literals of the variable receive a theta role but no case, which is what forces them to move." (Ristad, 1988: 25)

Ristad does not present the details of how this would work. I think it can be argued that in fact it cannot work. It is clear what Ristad's proposal requires: we must add to the lexicon L for each lexical item x_i that does not assign Case another lexical item that is exactly like x_i except that it does assign Case. This raises the question how we avoid that the Case-assigning variant is inserted in a position where it selects a literal

[9] On the not unreasonable assumption that the root node does not have to satisfy the Case Filter.

that is not leftmost. If this question cannot be answered, and I do not see how it could be, then considerations of Case will fail to force movement in a stair-based representation. But if movement cannot be forced, then the stair construction fails to enforce one of the 3SAT constraints, namely coherent truth assignments, and the proof that the LRP is NP-hard in models that permit a stair does not go through.

We now expect that there will be 3SAT instances which are not satisfiable but which do have a well-formed DS and SS in the Barriers and Lasnik–Saito models. As an example, consider (3):

(3) $(a \vee a \vee a) \wedge (a' \vee a' \vee a')$

This 3SAT instance is not satisfiable: if a is **true** then the right conjunct is false, if a is **false** then the left conjunct is false. But in the corresponding stair-based DS and SS in the Barriers and Lasnik–Saito models, the stairs corresponding to a and a' could all be **true**. This is because we could select the lexical items in the DS in such a way that all stairs receive Case in-situ and that consequently specifier-head agreement is not violated at SS.

3.4.3. Problem 2: Neither the Barriers nor the Lasnik–Saito model allows a stair

The proofs that the LRP is NP-hard in the Barriers and Lasnik–Saito models is based on a second lemma, namely that these models allow a stair. It can easily be shown that this lemma fails to be derived, because it is based on the incorrect assumption that long distance A-movement is allowed in these models. If long A-movement can be shown to be impossible, then there will be stairs which are unable to move to a Case-marked position without violating a locality constraint. This is easy to see: two occurrences of the same variable in a 3SAT instance may be separated by an arbitrary number of other propositional variables. In the DS corresponding to the 3SAT instance, the two stairs corresponding to the two occurrences of the same variable may therefore be separated by an arbitrary number of intervening stairs.

Ristad's assumption that long A-movement is possible is based on the following reasoning:

"In the Barriers stair construction, an argument undergoes long movement by adjoining the argument NP to VP, γ-marking its trace, and then deleting the intermediate A′-trace at LF.

This is the exact derivational sequence (adjoin, γ-mark, delete adjoined trace) used in Barriers (pp. 21–22) to move a wh-subject from a theta-marked CP complement to a specifier of CP, provided the wh-phrase is licensed at LF. Barriers attempts to exclude similar long movement of an NP from a similar (but caseless) subject position by appeal to Binding Condition C at S-structure: the NP-trace in subject position would be an A′-bound R-expression A-bound in the domain of its head (p. 93, fn. 20) ... .

Crucially, condition C will exclude long movement of NPs only if trace deletion is restricted to LF. Otherwise adjoined traces could be deleted before causing an S-structure binding violation. But trace deletion cannot be restricted solely to LF. If it were, then any ECP violation created by LF-movement may be avoided, simply by deleting offending intermediate traces after they have done their γ-marking duty. This can be done because adjoined A′-traces are not required by the extended projection principle at LF. This is why neither Barriers nor Lasnik–Saito in fact restrict trace-deletion to LF. Therefore, long movement of an NP is not excluded in these models." (Ristad, 1991: 60)

While this is of no importance for my objections, it is a mystery to me how restricting trace deletion to LF could have the consequence that "any ECP violation created by LF-movement may be avoided, simply by deleting offending intermediate traces after they have done their γ-marking duty". Despite this we can grant the conclusion that trace deletion is not restricted to LF in the Barriers and Lasnik–Saito models, as Lasnik and Saito (1984) make this fully explicit. This is because they assume that affect-α operates in both the syntactic component and the LF component. Cf. (4) below (Lasnik and Saito's (180)):

(4) *Syntactic component*
D-structure
Affect α: (a) optionally move anything anywhere, subject to Subjacency, creating a trace or not; (b) optionally delete or insert anything lacking semantic content (e.g. *of, that, it, t*).
Index Comp.
Assign [+γ] to any properly governed argument.
Obligatorily assign [−γ] to any argument that is not properly governed.
S-structure: Filters

LF component
Affect α (identical to syntactic operation, except not subject to Subjacency).
Index Comp.
Assign [+γ] to any properly governed category.
Obligatorily assign [−γ] to any category that is not properly governed.
LF: Filters, including * [... *t* ...]
[−γ]

What Ristad fails to take into account, however, is that both in the syntactic component and in the LF component Affect α and γ-marking are assumed to be ordered with respect to each other: "γ-assignment follows Affect α in each component" (Lasnik and Saito, op. cit.: 285). It is also clear from Lasnik and Saito's presentation that these operation apply *in the mapping to* S-structure and *in the mapping to* LF, so that the filters at the level of S-structure follow Affect α and γ-assignment in the S-structure component and the filters at the level of LF follow Affect α and γ-assignment in the LF component. It follows that illicit movement can be ruled out in such a model by appeal to Principle C at S-structure. Consider (5), in which irrelevant structure has been omitted:

(5) John seems that it is $[_{VP}$ t′ $[_{VP}$ considered [t to be intelligent]

There are two possibilities to consider.

Suppose Affect α does not delete *t′* in the S-structure component. Then *t* will be antecedent governed and marked [+γ] in the S-structure component. Since *t′* can be

deleted in the LF component, (5) will not violate the ECP at LF. But because t' cannot be deleted before LF, (5) will violate Condition C at S-structure.[10]

Suppose Affect α deletes t' in the S-structure component (or, alternatively, suppose Affect α does not leave the trace t' in the S-structure component). Then t will not be antecedent governed and will therefore be marked [−γ] in the S-structure component. If t is deleted in the mapping to LF, then the Projection Principle is violated. If t is not deleted in the mapping to LF, (5) will violate the ECP at LF.[11]

Long A-movement is blocked for exactly the same reasons in the Barriers theory, since this model incorporates the Lasnik and Saito γ-marking and γ-checking mechanisms:

"For concreteness, let us assume further the mechanism proposed by Lasnik and Saito (1984) for determining satisfaction of the ECP. We say that α assigns the feature [+γ] to β if it properly governs β and that β receives the feature [−γ] if it is not properly governed, where γ-marking, once assigned, is permanent. γ-marking takes place at S-structure for elements of chains terminating in A-positions and at LF for elements of chains terminating in A′-positions, perhaps as a consequence of the Projection Principle in some manner. To satisfy the ECP, a nonpronominal empty category must have the marking [+γ] at LF." (Chomsky, 1986: 17–18)

Finally, consider Ristad's claim that

"even if trace deletion were disallowed entirely, long movement would still be possible from theta-marked noun complements, and the proof of lemma 3.3.1 [Barriers allows a stair – jvdk] would proceed, because θ-government cannot be eliminated without negative consequences in the rest of the theory." (Ristad, 1991: 61)

I will not go into the arguments that Ristad advances for the view that θ-government cannot be eliminated without negative consequences. Instead, I note that NP-movement will still be subject to a somewhat less stringent locality condition, namely Subjacency. It is therefore not the case that Ristad's modified theory (in which θ-government suffices for the ECP) allows NP-movement over arbitrary distances, as would be required to save the lemma.

3.4.4. Conclusions: Agreement and locality

I have shown that the proof that the LRP is NP-hard for models that allow a stair fails because of a problem with the lemma that stairs can enforce the 3SAT constraints. But even if this lemma could be derived, the proofs that the LRP is NP-hard in the Barriers and Lasnik-Saito models would still fail. This is because the locality conditions in these models conspire with Principle C to disallow a stair.

---

[10] This is because the foot of the chain [*John*, t', t] is an A′-bound R-expression that is A-bound in the domain of the head of its chain.

[11] Note that a derivation in which *John* is raised without leaving t' cannot be salvaged in the LF component by lowering *John* to the VP-adjoined position and then raising it back again, this time leaving t'. This derivation, although possible, cannot undo the [−γ] feature assigned to t in the S-structure component.

Remember that in the LFG and the WG proofs we made crucial use of fairly unrestricted methods for enforcing non-local agreement. As we discussed in connection with LFG, such agreement machinery appears to be too powerful in that it allows non-natural languages. In the Barriers and Lasnik–Saito models the various constraints on movement conspire in just the right way to rule out unrestricted non-local agreement.

This is of course not to say that no NP-hardness proof could be found for the Barriers or Lasnik–Saito model, or even that one should be concerned if one was found. As we will see in section 4, there is every reason to expect that certain components of an empirically adequate linguistic theory *will* give rise to computational intractability. But the worst-case languages ruled out by the Barriers and Lasnik–Saito models are non-natural languages and it is therefore a welcome result that the attempted NP-hardness proof considered here fails.

4. The language complexity game (Ristad, 1993)

4.1. Introduction

The main result of Ristad's monograph is a complexity thesis which states that language computations are NP-complete. This thesis therefore sets tight upper and lower bounds on the complexity of human language. Recall that a problem is NP-complete if it is NP-hard and it can furthermore be shown that the problem can be solved by a nondeterministic TM in polynomial time. Such problems have the important property that they are difficult to solve but that their solutions are easy to verify.

Ristad's monograph is important for a number of reasons, not all of which will be given equal attention here. While the complexity thesis is a major result in itself, the true value of the research lies in the novel characterization of human language on which it is based and in the linguistic insights that can be obtained from the methodology that guides it. In the following sections I briefly sketch Ristad's model of the language user, the methodology on which the defence of his complexity thesis is based, the defence of the thesis itself and finally what contributions to linguistics his work has made.

4.2. The object of study: Direct complexity analysis

Ristad (1991, 1993) argues that since the linguistic theory is not a model of the language user *at any level of abstraction* (because it is a model of linguistic knowledge and not of how that knowledge is put to use), looking at the recognition problems associated with linguistic theories is still rather a roundabout way of finding out about the complexity of language computations. It would be better to study the complexity of language computations directly, without reference to the peculiarities of particular linguistic theories at all. Following Ristad we will refer to this kind of study as direct complexity analysis.

An immediate problem that arises for this approach is that we know very little, if anything, about the computations performed by language users. How can we study the complexity of language computations if we do not know what they look like? As Ristad (1993) points out, computations may be characterized at various levels of abstraction. Language computations, for instance, may be described as sequences of brain state transitions, or as an algorithm which specifies how each state is mapped into its successor state by a given computational device, or very abstractly as *a computational problem.* A computational problem only specifies the relation between the inputs and outputs of a computation, abstracting away completely from how a computational device computes that mapping. At which level we can pitch our description of a given computation depends on the empirical evidence available. So little is known about language computations that we have no choice but to characterize them as computational problems. But how do we characterize language computations as computational problems?

Ristad proposes an interpretation of language according to which language acquisition, comprehension and production are all instances of a unitary computation. This computation maps some extralinguistic information i and a set of possible linguistic representations M into a linguistic representation r and a revised set of possible linguistic structures M'.[12] The extralinguistic information includes not just 'sensations' but also "intentions ..., beliefs, models of agents (other agents as well as oneself), mental lexicon, conceptual system, and so forth" (p. 9). The sets M and M' (also known as 'models') are a subset of $\mathfrak{M}$, the set containing all humanly possible sets of linguistic representations. On this view there is no principled distinction between comprehension and production: both are a computation from extralinguistic information to a linguistic representation of that information.

In order to make further investigation possible, it is necessary to decompose each model M into a set of disjoint 'submodels'. A submodel represents a natural class of linguistic knowledge, such as knowledge about referential dependencies. Crucially, a representation in a submodel does not just represent knowledge about some extralinguistic information. This is not difficult to see: in order to represent knowledge about e.g. referential dependencies, we also need some linguistic information, such as what phrase structure is assigned to the extralinguistic input by other submodels. The results of the investigation will therefore be theory-neutral to the extent that the linguistic input to the computational problem under study is theory-neutral.

Ristad's characterization of language differs radically from Chomsky's I-language. Consider language user U with language computation f_U, whose knowledge of language is characterized by the I-language I. I is a generative procedure: it enumerates the set of complete structural descriptions. As such it provides only a partial characterization of the set M, the set of all possible representations computed by f_U, which includes incomplete representations for incomplete or insufficient extralinguistic evidence. But the input to the generative procedure I and the computation f_U

[12] Thus, the unitary language computation is a function $f(i,M) = \langle r, M' \rangle$, where $M = M'$ if no acquisition takes place.

performed by language user U are also different. The input to I is an underlying form, for instance an array of items taken from the lexicon, whereas the input to the computation f_U is the extralinguistic evidence supplied by other cognitive systems.

The interpretation of language according to which language comprehension and production are instances of the same unitary computation from extralinguistic information to a linguistic representation of that extralinguistic information has a very important consequence:

"... if the producer and comprehender have roughly the same extralinguistic information as input, they will output the same linguistic representation using the same computational resources of time and space. This suggests that language computations cannot be too complex. However, it is not always the case that the producer and comprehender have roughly the same extralinguistic information as input. In some cases, such as when their mental lexicons or conceptual systems differ significantly, their inputs will be so different that it is impossible in principle for them to output the same linguistic representation. In other cases, such as when the comprehender's sensory information is impoverished, their inputs will be sufficiently similar that it is possible in principle for them to output the same linguistic representation, although comprehension will require significantly more computational resources than production did. In short, our interpretation suggests that language can be complex, but not too complex." (Ristad, 1993: 14)

But of course this is a near perfect characterization of an NP-complete problem. Such a problem is difficult, but not too difficult: a candidate solution is easily verified to be correct. The point of Ristad's interpretation of language is that if the comprehender's extralinguistic information is (nearly) identical to that of the producer, then since the producer was able to efficiently compute the intended linguistic representation, the comprehender should be able to do so too. Ristad points out that this view of language therefore places a heavy burden on a cooperative interaction between speakers and hearers. Without this cooperation the intractability inherent in the NP-complete natural language computation would make communication very difficult quite quickly.[13]

4.3. The method of study: The language complexity game

The methodology on which Ristad's results are based is designed to guarantee the accuracy of the complexity results. It is at once simple, effective and entertaining. Two players, known as the maximizer and the minimizer, play a number of rounds of an adversarial game, called the language complexity game. The maximizer will try to demonstrate that natural language computations are more difficult than was previously thought, whereas the minimizer will try to convince us that the opposite is the case and that natural language computations are less complex than we previously believed. The maximizer's attitude to the game is likely to lead to the discovery of complex properties of language which had previously gone unnoticed, thereby

[13] It seems to me that this view of communication meshes nicely with a theory of utterance interpretation of the type proposed by Grice (1975). For a reinterpretation of Grice's ideas in a cognitive setting, see Sperber and Wilson (1986).

making sure that our language models are not too simple. By contrast, the minimizer's attitude to the game will be to identify unnecessary complexity in our language models, thereby making sure that they are not more complex than can be empirically motivated. A game is initiated by one of the players proposing a plausible submodel of linguistic knowledge. The game ends if neither player is able to make a further contribution to our understanding of natural language that affects our views concerning its computational complexity, with the last player to make a successful move the winner of the game. Each player will be highly motivated to put forward an optimally accurate submodel, in the hope that his current move will end the game.

The maximizer and the minimizer will typically make rather different contributions to our knowledge of language. It is the maximizer's business to prove a lower bound on the complexity of language computations (i.e. that some language problem *L* is at least as difficult as problem X, where problem X is of known complexity). As we have seen in section 3, this involves carrying out a reduction from the problem of known complexity X to the new problem *L*. Since a reduction is based on some structural similarity between the old and the new problem, the reduction is likely to advance our understanding of the language problem *L*. By contrast, the minimizer wants to prove an upper bound on the complexity of the language problem *L* (i.e. that *L* is at most so difficult). Such a proof requires the minimizer to literally exhibit an algorithm that solves *L*. If the minimizer's move is in response to a move by the maximizer, then a natural first step for the minimizer is to carefully consider the sources of complexity the maximizer has uncovered in an attempt to exhibit additional empirical evidence that will falsify each source of complexity. This may then lead to the construction of a less complex submodel.

4.4. The complexity of language computations

Ristad's monograph brings together five rounds of the complexity game, with the author taking turns as the maximizer and the minimizer. All five rounds investigate the complexity of a submodel for anaphoric agreement. The linguistic problem of choice is the Anaphora Problem. The formulation of this problem is based on the observation that anaphoric elements (e.g. pronouns, reflexives and reciprocals) are dependent on other linguistic elements for their reference. Anaphoric elements must have antecedents. Therefore, these elements pose a computational problem, namely to determine a unique antecedent for each anaphoric element in a given linguistic representation. The output of the Anaphora Problem has the useful property that it is independent of other cognitive systems. In particular, the Anaphora problem is logically independent of the problem of assigning a referent to referring expressions. What is the input to the Anaphora problem? First of all, the input to the problem must include a set of available antecedents. This set will consist of the available antecedents in the current utterance and in the preceding utterances. But of course there is other linguistic information that is relevant to the Anaphora Problem, such as the linguistic features carried by elements involved in anaphoric relations and the structural configuration in which such elements appear. Ristad states what he calls the 'broad' Anaphora Problem as follows:

(6) *Anaphora problem*
Given a linguistic representation R lacking only relations of referential dependency, and a set A^C of available antecedents, output an antecedent in A^C for every anaphoric element in R.

This statement of the problem abstracts away from the fact that more than one set of linkings may be consistent with the set A^C in R, although only one of these will be salient to the language user. Ristad shows, however, that the simple Anaphora problem reduces to a more refined version which takes account of salience. It follows that complexity bounds on the simple problem carry over to the more refined problem, so that the complexity game may safely be based on the simple version of the Anaphora Problem in (6).

The first move in the game is played by the maximizer, who puts forward a precise model of a widely accepted constraint on anaphoric agreement, the standard agreement condition (SAC). The SAC captures the fact that anaphoric elements must agree in certain inflectional features with their antecedents. According to this condition, an anaphoric element agrees with its antecedent only if they do not disagree on the value of any inflectional feature. The maximizer has a field day and proves without any difficulty that the Anaphora Problem is NP-hard under this model of anaphoric agreement.

It then is the minimizer's turn to look for weaknesses in the submodel proposed by the maximizer. It does not take this player much trouble to find some clear counterexamples to the SAC and he proposes an improved model of anaphoric agreement, based on an alternative agreement condition called the Anaphoric Uniqueness Condition (AUC), according to which the Anaphora Problem is contained in P. In other words, the Anaphora Problem has an efficient solution under this model.

At this point the maximizer decides to extend the scope of the anaphora complexity game and to consider also the interpretation of anaphora in elliptical constructions. In particular, he draws attention to the fact that the pronoun in the ellipsed part of sentences such as *Max hates his neighbours and so does Sam* has a strict and a sloppy reading. In order to be able to consider the complexity of the Anaphora Problem when such facts are taken into account, the maximizer must propose a model of ellipsis. Taking the copy-theory of ellipsis as his starting point, the maximizer is able to prove that the Anaphora Problem is *PSPACE*-hard under this model.

Undeterred by this onslaught, the minimizer puts his finger on two weaknesses in the copy-theory. He points out that this theory predicts that the original VP and its copy in a VP-ellipsis construction should be subject to the same post-copying constraints, although operations applied after copying should be able to apply to the original and the copy independently. The minimizer shows that neither of these predictions holds and proposes an alternative model of ellipsis based on function-sharing. This model is based on the observation that the overt and null VPs of a VP-ellipsis construction are, in some sense, the same predicate. The minimizer then shows that the Anaphora Problem is contained in *NP* according to the function-sharing model by exhibiting an *NP* algorithm. The maximizer gives up. The outcome of this

game therefore strengthens the thesis that the unitary natural language computation is NP-complete.

4.5. Contributions of Ristad's work to the field of linguistics

The characterization of human language put forward by Ristad is a necessary first step in the investigation of the computational properties of human language and therefore an important contribution to the field as a whole.

The proposed research methodology is arguably a key component of the next generation of linguistic theories, which will probe directly into the nature of the unitary language computation performed by language users. In Ristad's view this new generation of theories will supersede current generative theories, which, he argues, are not a model of the language user at any level of abstraction.

The complexity game itself makes a number of substantial contributions to our understanding of human language. First, the game led to an investigation of the intrinsic properties two elements in an anaphoric relationship must have. In this part of the game the minimizer falsified a widely-accepted agreement constraint (the SAC) and proposed an alternative agreement condition (the AUC). Second, the game has considered aspects of the interpretation of anaphora which are not captured by work that only considers the syntactically determined distribution of coreference relations, the typical approach in the generative tradition. In particular, the three principles of the standard Binding Theory do not specify a complete mapping from phrase structure to referential dependencies. Condition A does not link an anaphor to a unique antecedent because there may be more than one potential antecedent in its local domain. Condition B and Condition C only specify relations of obviation and therefore do not link an anaphor to an antecedent at all. The investigation of the Anaphora Problem is a first step towards a complete specification of the computations language users must perform in order to interpret utterances containing anaphora. Finally, the third result of the complexity game is that it forces a clear choice between two competing analyses of VP-ellipsis, which it is difficult to choose between using just the methods of generative linguistics.

5. Conclusions

We have seen that URP for LFG and WG is NP-hard. In particular, it was shown that a sub-problem, known as the Lexical Resolution Problem (LRP), is NP-hard with respect to these theories. Ristad's proof that the LRP with respect to the Barriers and Lasnik–Saito models of grammar is also NP-hard was shown to fail for essentially non-technical reasons. Apart from a Case-theoretic problem with the proof, it relied on the assumption that the Barriers and Lasnik–Saito models allow long distance A-movement, whereas in fact they do not.

Other work by Barton et al. (1987), not discussed here, has demonstrated that the URP for GPSG (Gazdar et al., 1985) is EXP-POLY time-hard, whereas the URP for a suitably revised version of GPSG has been shown to be NP-complete. There are

also complexity results for context-free language recognition, context-sensitive language recognition and certain transformational language recognition problems (Aspects style transformational grammars). Table 2 gives an overview.

Table 2
The complexity of some linguistic recognition problems.

Class	Example problem	Resource bound
P	Context-free language recognition	Polynomial time on a deterministic TM
NP	LFG recognition; WG recognition; Revised GPSG (Barton et al., 1987); Lasnik–Saito/Barriers recognition?	Polynomial time on a nondeterministic TM
PSPACE	Context-sensitive language recognition	Polynomial space on a deterministic TM
EXP	Certain transformational language recognition problems	Exponential time on a deterministic TM
EXP-POLY	GPSG recognition (version Gazdar et al., 1985)	Det time $O(c^{f(n)})$ for some constant c and polynomial f(n)

It seems that most modern linguistic theories are – computationally speaking – in the same boat. Their associated recognition problem is NP-hard. For Revised GPSG there is even a proof that its recognition problem is NP-complete (i.e. in NP). If it could be shown for all these theories that the associated recognition problem is in fact NP-complete, then that would be a very interesting result. Recall that NP-complete problems have the curious property that they are hard to solve but their solutions are easy to verify. In this sense, NP-complete problems are on the verge of tractability. Problems that are PSPACE-hard, EXP-hard or even EXP-POLY-hard cannot be verified efficiently.

I mentioned earlier that it follows from a proof that the URP for a linguistic model is NP-hard that there is nothing in that model that guarantees efficient parsing. I was careful at that point not to add: "and therefore that linguistic model is inadequate", because that is not at all a necessary consequence. How, then, should we interpret a proof that the URP for some linguistic model is NP-hard?

This question is best answered by asking another one. Do we have any a priori reasons to assume that there do not exist any subproblems of the general problem of natural language comprehension that are computationally intractable? The answer to this question, as we have seen in section 4, is negative. Ristad's direct complexity analysis shows that language users must perform computations which are NP-hard.

Direct complexity analysis provides insight into which language computations are computationally intensive and which are not. Thus, it becomes possible to predict in which areas of an empirically adequate model of the language user computational

complexity will show up. Since the linguistic theory specifies the linguistic knowledge of the language user, this may allow one to identify parts of the linguistic theory as computationally inadequate or unnaturally powerful, although some caution is required in the case of the latter. This is because an overly powerful linguistic theory might be restricted either by constraining the linguistic theory itself or by the imposition of an appropriate performance constraint (e.g. a locality principle for parsing; cf. the discussion of the LFG complexity results in section 3.2.3). So if one shows that certain parts of a linguistic model make the URP for that model computationally intractable, then one must ask whether those parts are unnaturally powerful (because they are compatible with phenomena which do not occur in natural languages) or whether their complexity is inherited from the complexity of the underlying language comprehension problem (in which case the parts of the model giving rise to computational intractability are a necessary component of an empirically adequate linguistic theory). As we have seen, there are reasons to believe that the NP-hardness proofs for LFG and WG have identified unnaturally powerful machinery.

Finally, direct complexity analysis opens up a new avenue to the study of language, complete with a research methodology with a rather different orientation than that of the generative tradition. It is based on the interpretation of human language as a computational system and studies the complexity of natural language computations with the aim to improve our understanding of its form.

Appendix

The following proof is taken verbatim from Ristad (1988).

Theorem. The LRP is NP-hard in the Barriers model.

Proof. On input 3-CNF formula F, the reduction will create a lexicon L and a D-structure that represents F, then apply move-α, and finally insert underspecified lexical forms to derive an incomplete S-structure that can be completed according to lexicon L if and only if F is satisfiable. The crux of the reduction is to represent each literal with the noun complement structure *[N [I VP]]* in (1) (for example, *desire [to visit places]*):

(1) $[_{\mathrm{NP}_j} \cdots [_{\mathrm{NP}_i} N\ [_{IP_i}\ [e]\ I\ [_{VP_i}\ V\ NP_{i+1}\ ]]\ ]]$

The Barriers model endows this construction with the required characteristics:

1. *Transparent to extraction.* NP_{i+1} can be moved out of NP_i in the structure (1). VP is a BC and barrier for NP_{i+1} because it is not L-marked, but NP_{i+1} can adjoin to the non-argument VP and void its barrierhood because nonarguments may be freely adjoined to. Both NP_i and IP_i are L-marked, and therefore are neither BCs nor barriers for further NP_{i+1} raising. Thus, NP_{i+1} can be A-moved to any c-commanding specifier-of-IP position *[e]* without violating the ECP because all traces

are properly governed (both θ-governed by the verb V that selects NP_{i+1}, *and* γ-marked (antecedent-governed) by the deleted trace adjoined to VP).

Reinhart (personal communication) suggests a similar, albeit marginal, natural example where an NP containing an argument trace is topicalized to CP specifier from an L-marked position:*

(7)? [What burning $t_{i+1}]_i$ did John say [of what book]$_{i+1}$ [t_i would be magnificent]

2. *Contains a landing site that agrees with the constituent head.* The internal IP_i contains a specifier position (landing site) that will agree with I by specifier-head agreement in non-lexical categories; the specifier position will also agree with N (the constituent head), by predication. Alternatively, head movement from V to I to N can create an inflected noun *[$_N$ [$_I$ V I] N]* in the X^0 position of NP_i that will agree with the landing site. Although I cannot find a natural example of such an inflected noun, no arguments or analyses exclude it in principle. A close natural example is noun incorporation in Mohawk verbs (Baker, 1985: 139).

3. *Undergoes obligatory movement.* V assigns a θ-role but no case to the NP_{i+1} position, requiring NP_{i+1} to move. This is possible if V has lost its ability to assign case (passive morphology) or if NP_{i+1} is the underlying subject of VP_i, as in many current accounts (Sportiche, 1986; Fukui, 1986; Larson, 1988, etc.).

4. *Selectional properties are correlated with φ-features.* V undergoes obligatory head movement to the affix I, creating an inflected verb in the head of IP. As noted above, the φ-features will appear on the inflected verb by specifier-head agreement, where they may be systematically correlated with the verb's selectional properties in the lexicon.

Details. The input to the reduction is a 3-CNF formula

$$f = (u_{11}u_{12}u_{13}),(u_{21},u_{22}u_{23}),...,(u_{n1}u_{n2}u_{n3})$$

with n clauses, $3n$ literals, and variables $q_1 ... q_m$. As before, the reduction will create a D-structure that represents F, and then apply move-α and lexical insertion to derive the S-structure S. The reduction output will consist of an S-structure S containing underspecified words and a lexicon L, such that the words in S can be resolved according to L if and only if F is satisfiable.

---

* Chomsky (pc) suggests that the correct analysis of (7) is

(6) [what burning]$_i$ did John say [[t_i of what book] would be magnificent]

and that a better topicalization example might be *What burning did John say (that) of that book, Mary thought would be magnificent.*]

The reduction algorithm consists of five steps:

1. *Create the two φ-features* VBL *and* TRUE
   The *m*-ary VBL feature identifies the formula variables: the formula variable q_k is associated with the feature [VBL *k*]. The final binary φ-feature TRUE denotes the truth assignment to the given variable.

2. *Create the D-structure representation of f.*
   For each literal u_{ij} in the i^{th} clause, create the D-structure noun complement construction:

   (8) $[_{NP_{ij}}\ N_{ij}\ [_{IP_{ij}}\ [e]\ I_{ij}\ [_{VP_{ij}}\ V_{ij}\ NP_{ij+1}\ ]]\ ]$

   where NP_{ij} represents the subformula u_{ij}... u_{n3}; NP_{ij+1} represents the subformula u_{ij+1} ... u_{n3}; and N^0_{ij} bears the φ-features of the variable q_k that corresponds to the literal u_{ij}.

3. *Apply head-movement within noun complements.*
   Move V_{ij} to I_{ij}, and then move the inflected verb $[V_{ij}\ I_{ij}]$ to N_{ij}:

   (9) $[_{NP_{ij}}\ N_{ij}\ [_{IP_{ij}}\ [e]\ I_{ij}\ [_{VP_{ij}}\ V_{ij}\ NP_{ij+1}\ ]]\ ]$

   (10) $[_{NP_{ij}}\ [[V\ I]\ N_{ij}]\ [_{IP_{ij}}\ [e_{ij}]\ t\ [_{VP_{ij}}\ t\ NP_{ij+1}\ ]]\ ]$

   As a result, all X^0 positions other than N^0 positions have traces in them; N^0 positions are filled with inflected nouns of the form [[V I] N]. (The goal of this operation is to force agreement between the verb and the noun; this could also be done by predication.)

4. *Apply long distance NP-movement across noun complements.*
   Starting with the rightmost, innermost NP (NP_{n3}) and scanning left and up, move NP_{ij} to the specifier position $[e_{lm}]$ of the closest inflection I_{lm} that agrees with it in φ-features:

   (11) ... $[_{NP_{lm}}\ [[V\ I]\ N_{lm}]\ [_{IP_{lm}}\ [e_{lm}]\ t\ [_{VP_{lm}}\ t\ NP_{lm+1}$...

   $[_{NP_{ij}}\ [[V\ I]\ N_{ij}]\ [_{IP_{ij}}\ [e_{ij}\ t\ [_{VP_{ij}}\ t\ NP_{ij+1}\ ]]\ ]]\ ]]\ ]$

   (12) ... $[_{NP_{lm}}\ [[V\ I]\ N_{lm}]\ [_{IP_{lm}}\ [NP_{ij}]\ t\ [_{VP_{lm}}\ t\ NP_{lm+1}$... $t_{ij}\ ]\ ]]\ ]$

The leftmost literal of each formula variable is assigned case in situ by the formula literal that selects it; all other literals of the variable receive a θ-role but no case, which is what forces them to move. All literals of a variable but the rightmost one assign case to their specifier position, so that a literal will be able to land there. Our third ally in making the movement obligatory is the extended projection principle, which requires specifier of IP to be filled.

Movement is by adjunction to intermediate VPs, deleting intermediate traces. This movement satisfies the ECP because all traces are properly governed and therefore γ-marked at S-structure (both θ-governed by the verb V_{ij} that selects them, *and* antecedent-governed by the deleted trace adjoined to VP). The movement satisfies minimality trivially because the deleted trace is not excluded by VP. Because NPs are only moved to specifier positions that c-command their bound traces, the resulting structure also satisfies the chain condition and binding theory when the chain is created. Note that the movement is permissible independent of whether we express the antecedence relations induced by move-α using indices or links (cf. Higginbotham, 1983).

Now NP_{ij} agrees with its c-commanding NP_{lm} on the TRUE feature by specifier-head agreement in IP and head-head agreement between I_{lm} and the head of NP_{lm}.

5. *Perform (underspecified) lexical insertion.*
Insert the morphological form $\alpha_j\beta_j\gamma\varphi_k$ for the inflected noun [[V I] N_{ij}] when $u_{ij} = q_k$ (that is, u_{ij} is not a negated variable); when $u_{ij} = q'_k$, insert the morphological form $\alpha_j\beta_j\gamma'\varphi_k$ instead.

(13) ... $[_{NP_{lm}}$ *[[V I] N_{lm}] $[_{IP_{lm}}$ [NP_{ij}] t $[_{VP_{lm}}$ t NP_{lm+1}]]]*

(14) ... $[_{NP_{lm}}$ *[$\alpha_i\beta_i\gamma\varphi_k$] $[_{IP_{lm}}$ [$\alpha_i\beta_i\gamma'\varphi_t$] t $[_{VP_{lm}}$ t NP_{lm+1}]]]*

Note that each '$\alpha_j\beta_j$' is a listed word; that γ and γ' are suffixes; and that φ_k is the morphological realization of the φ-feature [VBL k].

Further note that the surface structure is a string of inflected words. For example, on input $f = q_1q'_2q_3q_2q'_1q_3$ the reduction algorithm would yield the surface string or linguistic expression:

$\alpha_1\beta_1\gamma\varphi_1\ \alpha_2\beta_2\gamma'\varphi_1\ \alpha_2\beta_2\gamma'\varphi_2\ \alpha_1\beta_1\gamma\varphi_2\ \alpha_3\beta_3\gamma\varphi\ \alpha_3\beta_3\gamma\varphi_3$

We now turn to morphological and lexical details of the reduction, namely how we ensure that each clause has one true literal, that literals and variables may be true or false, and that negation changes truth values.

The *lexicon* contains ambiguous inflected nouns *[[V I] N]* that have undergone verbal incorporation. we define the morphological primitives and lexicon to ensure that the N-complement structure NP_{i1} for the first literal in the ith clause u_{i1} either (a) corresponds to a true literal, or (b) selects a true literal (as represented by) NP_{i2}, or (c) selects a false literal NP_{i2} that will select a true literal NP_{i3}. Similarly, the N-complement structure NP_{i2} for the second literal in the ith clause u_{i2} must either (a) correspond to a true literal, (b) correspond to a false literal, or (c) correspond to a false literal and selects a true literal NP_{i3}. Finally, the structure NP_{i3} for u_{i3}, the third and final literal in the ith clause, may correspond to a true or false literal.

The lexical system requires one φ-feature TRUE on nouns to encode variable truth assignments; one non-φ-feature LITERAL to encode literal truth values on both nouns

and verbs; and one non-φ-feature T on nouns that verbs can select for. Recall that we are assuming that the m-ary φ-feature VBL will be morphologically realized on the nouns in the lexicon, and will distinguish the formula variables: N_{ij} and N_{lm} bear the same VBL value if and only if u_{ij} and u_{lm} are literals of the same variable.

The *morphological system* is constructed assuming the relativized head definition of DiScullio and Williams (1987). There are six ambiguous morphological primitives: the prefixes α_1, α_2, α_3 and roots β_1, β_2, β_3. Together they define three ambiguous words $\alpha_1\beta_1$, $\alpha_2\beta_2$, and $\alpha_3\beta_3$, which may each combine with either of the two nominal suffixes γ, γ'.

Listed word	Prefix (α_i) features	Root (β_i) features
$\alpha_1\beta_1$	N [LITERAL 1]	V [LITERAL 1]
	V [LITERAL 0, Select T]	N [LITERAL 0]
$\alpha_2\beta_2$	N [LITERAL 1, T]	V [LITERAL 1]
	V [LITERAL 0]	N [LITERAL 0]
	V [LITERAL 0, Select T]	N [LITERAL 0, T]
$\alpha_3\beta_3$	N [LITERAL 1, T]	V [LITERAL 1]
	V [LITERAL 0]	N [LITERAL 0]

The suffixes relate variable truth assignments to literal truth values. For example γ' will only be lexically inserted to represent a negated variable, and therefore inversely relates the truth value of the variable and literal.

Suffix	Features	Selects
γ	N [TRUE 1]	[LITERAL 1]
	N [TRUE 0]	[LITERAL 0]
γ'	N [TRUE 1]	[LITERAL 0]
	N [TRUE 0]	[LITERAL 1]

Finally, there are m listed inflectional affixes corresponding to the m possible values of the VBL φ-feature. This completes the presentation of the reduction algorithm.

References

Baker, M., 1985. Incorporation: A theory of grammatical function changing. Ph.D. disseration, MIT Department of Linguistics and Philosophy. (Published by University of Chicago Press, 1987.)

Barton, G., R. Berwick and E. Ristad, 1987. Computational complexity and natural language. Cambridge, MA: MIT Press.

Bresnan, J., 1978. A realistic transformational grammar. In: M. Halle, J. Bresnan, G. Miller (eds.), Linguistic theory and psychological reality. Cambridge, MA: MIT Press.

Bresnan, J., 1982. The mental representation of grammatical relations. Cambridge, MA: MIT Press.

Brody, M., to appear. Lexico-logical form: A radically minimalist theory. Cambridge, MA: MIT Press.

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

Chomsky, N., 1986. Barriers. Cambridge, MA: MIT Press.
Chomsky, N. and H. Lasnik, 1991. Principles and parameters theory. Ms., MIT. To appear in J. Jacobs, A. von Stechow, W. Sterneveld, T. Vennemann (eds.), Syntax: An international handbook of contemporary research. Berlin: Walter de Gruyter.
Chomsky, N. and G. Miller, 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, 269–322. New York: Wiley.
Fukui, N., 1986. A theory of category projection and its applications. Unpublished Ph.D. dissertation, Department of Linguistics and Philosophy, MIT
Gazdar, G., E. Klein, G. Pullum and I. Sag, 1985. Generalized phrase structure grammar. Oxford: Blackwell.
Giorgi, A., F. Pianesi and G. Satta, 1989. The computational complexity of binding theory's satisfiability and verification problems. Unpublished manuscript, IRST, Trento, Italy.
Grice, P., 1957. Logic and conversation. In: Cole and J.L. Morgan (eds), Syntax and semantics, Vol. 3: Speech acts, 41–58. New York: Academic Press.
Harel, D., 1987. The spirit of computing. Wokingham: Addison-Wesley.
Higginbotham, J., 1983. Logical form, binding and nominals. Linguistic Inquiry 14, 395–419.
Hudson, R., 1984. Word grammar. Oxford: Blackwell.
Hudson, R., 1990. English word grammar. Oxford: Blackwell.
Koot, J. van de, 1992. Word grammar recognition is NP-hard. In: J. van de Koot (ed.), UCL Working Papers in Linguistics 4.
Larson, R., 1988. On the double object construction. Linguistic Inquiry 19(3), 335–391.
Lasnik, H. and M. Saito, 1984. On the nature of proper government. Linguistic Inquiry 15, 235–289.
Marcus, M., 1980. A theory of syntactic recognition for natural language. Cambridge, MA: MIT Press.
Ristad, E.S., 1986. Sources of complexity in GPSG Theory. Theoretical Linguistics 13(1/2), 105–124.
Ristad, E.S., 1988. The complexity of human language comprehension. Technical Report 964, Artificial Intelligence Laboratory, MIT.
Ristad, E.S., 1991. A constructive complexity thesis for human language. Ms., Princeton University.
Ristad, E.S., 1993. The language complexity game. Cambridge, MA: MIT Press.
Sperber, D. and D. Wilson, 1986. Relevance: Communication and cognition. Oxford: Blackwell.
Sportiche, D., 1986. A theory of floating quantifiers and consequences. Proceedings of NELS 17.
Williams, E., 1984. Grammatical relations. Linguistic Inquiry 15, 639–673.