Word Grammar Recognition is NP-hard

HANS VAN DE KOOT

1 Introduction

The aim of this paper is to demonstrate that Word Grammar[1] Recognition (WGR) is computationally intractable, in particular that it is NP-hard[2]. I will show that the source of WG's computational complexity lies in its unnaturally powerful machinery for feature inheritance[3]. WG shares this weakness with Lexical Functional Grammar, which contains very similar machinery allowing unrestricted hierarchical feature unification. The proof I present is similar to the NP-hardness proof for LFG[4] Recognition in Barton, Berwick and Ristad (1987).

The paper is organized as follows. I first discuss some key concepts and techniques from the theory of computational complexity (section 2). In section 3, I apply the notion of complexity to linguistic theories and discuss how we may distinguish between natural and unnatural computational complexity. In section 4 I prove that Word Grammar Recognition is NP-hard.

2 Computational complexity theory

2.1 Easy and hard problem: the classes *P* and *NP*

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 integer j). Such problem are called *tractable*. Table (1) (after Garey and

[1]Hudson (1984) and (1990).

[2]See section 2 below for a definition.

[3]Default inheritance plays a vital role throughout WG. My proof will rely on the fact that in WG a head may inherit all the features of its complement for which it is itself specified.

[4]Bresnan (1978) and (1982).

Johnson 1979) shows why. Complexity theory dubs this class *P*: the class of problem solvable in *P*olynomial time on a deterministic TM.

(1)

	Problem size, *n*		
Time compl.	10	50	100
n^3	.001 sec	.125 sec	1.0 sec
2^n	.001 sec	35.7 years	10^{15} cent

Other problems take longer, no matter how hard one tries to write an algorithm that will do better. Famous examples are the travelling sales person problem and Satisfiability (or SAT). SAT goes likes this. Given an arbitrary Boolean formula like (2)

(2) $(x \vee y' \vee z') \wedge (y \vee z \vee u) \wedge (x \vee z \vee u') \wedge (x' \vee y \vee u)$

is there an assignment of true and false to the propositional letters such that the whole expression is true (a is true if a' is false and vice versa). In fact, (2) is a restricted version of SAT called 3SAT in which every clause contains exactly 3 literals. 3SAT is not easier than SAT. 3SAT is difficult because all one 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 obviously rise with the length of the formula, the complexity of SAT is proportional to 2^n, or exponential time. Such problem are called intractable and again table (1) shows why. The entries in this table assume that each algorithmic instruction takes 1 microsecond, but the shape of the curve relating problem size to processing time is what is really important.

Given any 3SAT formula, we can verify quite quickly whether a given guess is a good one or a bad one. All we have to do is walk through the formula from left to right checking whether our guess makes each clause true. It should be intuitively clear that this can be done in polynomial time (proportional to length of formula times length of guess list).

Taking this one step further, suppose we had a computer that could try out all guesses for the 3SAT problem in parallel. Clearly, such a computer

could solve 3SAT in polynomial time. Such a guessing computer is called *nondeterministic* and the class of problems solvable by a *N*ondeterministic computer in *P*olynomial time is called *NP*.

Complexity theorists have discovered many problems like SAT, which have the property that they only seem to have exponential-time *deterministic* solutions, but which have efficient *nondeterministic* solutions. We do not have a 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 also in *NP*. But problems like SAT 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, as is 3SAT, then it is called NP-complete.

2.2 The reduction technique

Showing that one problem is as difficult as another relies on a technique called reduction. Here is how it works. One starts off with a problem of known complexity, like 3SAT. One then constructs an efficient mapping from the problem of known complexity to the new problem, preserving Yes/No solutions. In this context efficient means polynomial time. If this can be done, then the new problem must be NP-hard. This is easy to see. Suppose that the new problem could be solved in polynomial time. Then the old problem, which we assumed was intractable, is solvable in polynomial time as well. This is because the composition of two polynomial time functions is itself a polynomial time function.

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.

3 Natural vs unnatural computational complexity

Over the past 5 years evidence has been accumulating that the general natural language comprehension problem (LC problem) is NP-hard and possibly NP-complete (cf. Barton, Berwick and Ristad 1987 and especially Ristad 1991). Ristad (op. cit.) applies the tool of complexity analysis to obtain NP-hardness

results for various aspects of the LC problem which hold irrespective of the particulars of any linguistic theory.

These results are likely to have a dramatic effect on our appraisal of the complexity of the universal recognition problem for grammatical theories[5], as they lead us to expect that certain parts of linguistic theories will inherit the complexity of the LC problem. An example should make clear why.

Ristad (op. cit.) defines the Anaphora Problem as follows:

(3) *Anaphora Problem*
Given a structural description *S* lacking only relations of referential dependency, and a set *A* of available antecedents, decide if all anaphoric elements in *S* can find their antecedent in *A*.

He demonstrates that the Anaphora Problem is NP-hard. His proof abstracts away completely from the particulars of linguistic theories of anaphora. But given the proof, it immediately follows that the Recognition Problem for any syntactic theory which poses the Anaphora Problem will be NP-hard. Anaphora resolution in GB theory is not NP-hard, but this is only because GB does not pose the Anaphora Problem (as defined by Ristad).

Put from a different angle, direct complexity analysis helps us to interpret the results of the complexity analyses of particular grammatical theories, because it tells us in which parts of linguistic theory we should expect complexity to show up. Complexity is "unnatural" if it shows up where it is not needed.

Thus, suppose that a linguistic theory poses the Anaphora Problem, as defined above. Then that part of the theory that deals with anaphora will be able to solve NP-hard problems (assuming that the theory is descriptively adequate with respect to anaphora). Such complexity is natural. By contrast, suppose we have reason to believe that a certain linguistic phenomenon, say subcategorization, is computationally simple. Then we would not be happy to discover that our theory can model subcategorization phenomena that do not in fact occur in natural languages.

Barton, Berwick and Ristad (1987) study the computational complexity inherent in particular linguistic theories. Although this method is less direct than Ristad's direct complexity analysis, it is useful in that it may uncover

---

[5]The *Universal Recognition Problem* for a linguistic theory *T* has the following form: given a sentence *x* and a grammar *G* specified by *T*, is $x \in L(G)$?

aspects of linguistic theories which give rise to unnatural computational complexity.

In the following section I will consider the computational complexity inherent in the grammatical theory known as Word Grammar. I will show that the unrestricted hierarchical feature inheritance that Word Grammar allows makes its recognition problem NP-hard. The kind of complexity the proof uncovers is unnatural, because the WG agreement machinery can model agreement phenomena that are not found in natural languages.

Williams (1984) points out that many of the problems of the LFG f-structure language stem from its failure to observe X'-restrictions[6]. Very much the same is true of the way feature inheritance works in WG. There are no "maximal projections" in WG to block further hierarchical feature inheritance. As Williams observes with respect to LFG, this leads to a framework in which one could force, say, the object of an embedded verb to be plural. Although it is fairly obvious how X'-theory could be imported into the functional language of LFG, I have no suggestions as to how one might go about imposing similar restrictions on WG representations.

4 Word Grammar Recognition is NP-hard

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

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

We then proceed with the reduction.

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

Proof. The reduction is from 3SAT. Given a 3SAT instance F[7] of length n using the variables $x_1 \ldots x_m$, reduce 3SAT, a known NP-complete problem to WGR in polynomial time.

---

[6]LFG has X'-theory in its constituent structure but not in its f-structure.

[7]For instance, a sentence such as (2) above.

To construct the WGR problem, we map F into a sentence x 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. We must write a WG 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 WG agreement machinery to enforce coherent truth assignments.

This grammar will vary slightly with the problem instance (details follow). Finally, we add lexical entries, one for each literal x_i and x'_i. The lexicon varies with the particular 3SAT instance.

Let us begin with (i). We define the WG to have three types of words: cat-1, cat-2 and cat-3. cat-1 always selects cat-2, cat-2 always selects cat-3, and cat-3 always selects cat-1 or no complement at all.

Next we take care of (ii), 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.[8] To get this idea to work, we need to specify subcategories of these word types. This is because the rules for complement selection must be able to tell each complement what it is supposed to do. For instance, a cat-2-tf is a category which has promised to make to make the clause true, while a cat-2-n is a category that has promised nothing. There are other clauses that define inheritance of category membership.

/* inheritance of category membership */

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

[8]This technique is essentially the one Ristad (1991) uses for an attempted NP-hardness proof for the Barriers model of grammar.

```
cat-2-t isa cat-2-tf.
cat-2-f isa cat-2-tf.
cat-2-tf isa cat-2.
cat-2-n isa cat-2.

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

cat-1-t isa cat-t.
cat-2-t isa cat-t.
cat-3-t isa cat-t.
cat-1-f isa cat-f.
cat-2-f isa cat-f.
cat-3-f isa cat-f.
cat-1-n isa cat-n.
cat-2-n isa cat-n.
cat-3-n isa cat-n.

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

/* selection of complements */

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

type of object of cat-1-t = cat-2-n.
type of object of cat-1-f = cat-2-tf.
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-tf.
```

/* features and feature inheritance */

WG allows a very general statement of feature agreement. What we want is for a head to share all the features of its complements:[9]

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[10]. 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:

/* lexicon */[11]

x_i isa {/: cat-t, cat-n}.
f_i of x_i = true.

x'_i isa {/: cat-f, cat-n}.
f_i of x'_i = true.

x_i isa {/: cat-f, cat-n}.
f_i of x_i = false.

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

---

[9]This is basically identical to the LFG f-structure notation (↓=↑).

[10]For instance, in many languages one wants to prevent a preposition from inheriting all the features of its complement.

[11]The symbol "/:" is the WG-notation for disjunction.

This construction takes time polynomial in the size of F. The mapping of F to a string x 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 $x \in L(G)$. Suppose that $x \in L(G)$. We must show that F is satisfiable. If $x \in L(G)$, 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 x, 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. QED.

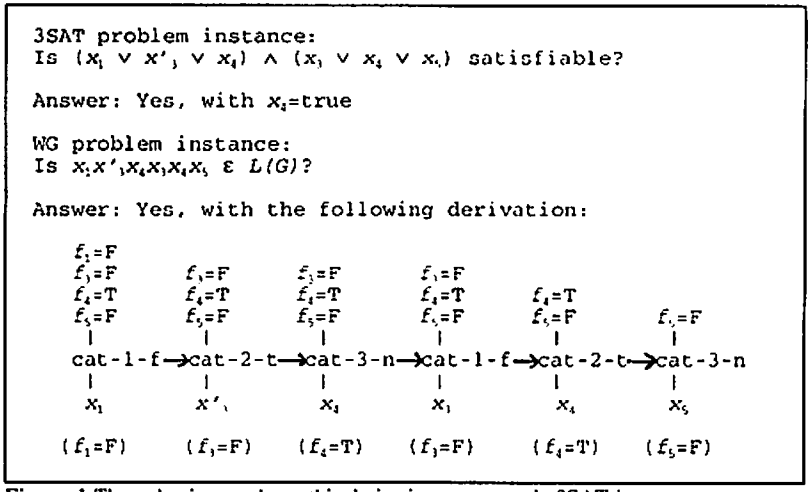

Figure 1 The reduction produces this derivation on a sample 3SAT instance.

References

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 and G. Miller, eds., *Linguistic Theory and Psychological Reality*. Cambridge, MA: MIT Press.

Bresnan, J. (1982). *The Mental Representation of Grammatical Relations*. Cambridge:MIT Press.

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

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

Hudson, R. 1984. *Word Grammar*. Oxford: Blackwell.

Hudson, R. 1990. *English Word Grammar*. Oxford: Blackwell.

Ristad, E. (1991). *A Constructive Complexity Thesis for Human Language*. [Revised version of 1990 PhD Thesis] Ms. Princeton University.

Williams, E. (1984). Grammatical Relations. *Linguistic Inquiry* 15:639-673.