Computers and Artificial Intelligence, Vol. 8, 1989, No. 5, 423—441

NATURAL LANGUAGE, COMPUTATIONAL COMPLEXITY, AND GENERATIVE CAPACITY

Robert C. BERWICK

NE43-838 MIT Artificial Intelligence Laboratory Cambridge, MA 02139, USA

Abstract. We show that the logical expressions of natural languages are strictly context-sensitive. However, even though context-sensitive languages are intractable in general, this weak generative capacity result does not necessarily imply any computational difficulty. Given a finite number of names, such expressions are still efficiently decidable (in polynomial time). We show this by providing an explicit algorithm for computing logical form binding in these cases. More broadly, such a result shows that the computational complexity of natural languages should be studied directly, rather than indirectly through the mechanism of weak generative capacity.

Естественный язык, вычислительная сложность и генеративная способность
Роберт С. Бервик

Резюме. Показано, что логические выражения естественных языков строго контекстно-зависимы. Однако, несмотря на то, что контекстно-зависимые языки практически неразрешимы, этот результат слабой генеративной способности не влечет за собой обязательно вычислительные трудности. Если задано конечное количество названий, такие выражения все еще эффективно разрешимы (в полиномиальном времени). Это показано путем предоставления явного алгоритма для вычисления связи логических форм в этих случаях. Вообще говоря, такой результат показывает, что вычислительную сложность естественных языков следует изучать лучше прямо, чем косвенно через посредство механизма слабой генеративной способности.

Keywords: Natural language processing, computational complexity, formal language theory, logical form.

CR classification descriptors: I.2.7 [Natural Language Processing], Language parsing and understanding.

1. INTRODUCTION

What constitutes knowledge of language? At heart, it is an ability to pair sound and meaning, ultimately an information-processing task. Modern computational complexity theory can provide powerful insights into the structure of this problem by providing an

algorithm-neutral analysis of information-processing structure. This paper investigates the computational complexity of syntax and semantics by examining particular computational problems posed by natural languages rather than their generative capacity.

This approach has several advantages. Many linguistic theories distinguish different representational levels for describing the syntax and semantics of natural languages, for example, the levels of surface structure (SS) and logical form (LF) in current theories.[1] Questions naturally arise as to the generative power of such multi-level systems. While there has been some preliminary and suggestive research in this area (see section 4) the results have engendered controversy and confusion: How can one compare theories that set different boundaries between syntax and semantics? What is the expressive power of logical form? If logical form is non-context-free, then do these results imply anything about the nature of linguistic theories or sentence processing?

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

In contrast, a generative capacity analysis can only tell us indirectly about computational complexity. To underscore this point, in this paper we show that the well-formed logical expressions of English, and presumably most, perhaps all natural languages, are non-context-free.[2] More interestingly, this weak generative capacity result apparently poses *no* computational hurdle for processing. Section 3 sketches efficient (polynomial time) algorithms for disjoint reference and nonvacuous quantification (Details are given in appendix B. An algorithm for the no-free-variables-constraint is analogous and is omitted).

These results hold under certain routine assumptions regarding logical form: that well-formed LF sentences are roughly of the sort described by CHOMSKY [6], HIGGINBOTHAM [10] and others, crucially obeying a disjoint reference (DR) constraint following LASNIK [13], and that coreference is indicated by coindexing.[3] The structure of the

[1] Note that even in theories that do not posit an explicit level of logical form generally contain interpretation rules that pair structural descriptions and interpretations (in some language), and this pairing may itself be regarded as a language.

[2] More precisely, we exhibit a regular set F, a homomorphism h, and a general sequential mapping g such that ($g(F \cap h$ (well-formed English LF expressions)) is a demonstrably non-context-free language. Since context-free languages are closed under homomorphisms, general sequential mappings, and intersection with regular sets, it may be concluded that English LF expressions themselves are non-context-free.

[3] The disjoint reference constraint bars coindexing between names or epithet NPs in a c-commanding domain, e.g.,

(1) * Reagan$_i$ thinks that [the aging actor]$_i$ is losing touch.

For the argument to go through we could restrict ourselves to names if necessary, avoiding epithets entirely.

argument would seem to hold for any theory that faithfully encodes the disjoint reference constraint.[4]

These findings do *not* necessarily imply that the full mapping from surface structure to logical form is computationally tractable, only that three specific LF constraints are. Even so, the approach shows how computational complexity analysis may be used to pose problems about the surface structure—LF mapping generally.

That LF constraints are efficiently computable should come as no surprise, because as emphasized in [4] many strictly non-context-free languages (such as $a^n b^n c^n$) may be efficiently analyzed. Thus, while the *class* of strictly non-context-free languages has no general, efficient recognition algorithm, this fact does not imply that particular strictly non-context-free languages of that class will be likewise intractable. The disjoint reference example explicitly provides such a case. We therefore conclude once again that there is no *necessary* link between strict context-sensitivity and computational difficulty.

Beyond this narrow point, the result shows that an exclusive focus on the context--free/context-sensitive language hierarchy is misleading. Presumably, the aim of a weak generative capacity analysis is to forge some connection (however tenuous) between a linguistic formalism and computational processing demands. If so, then examples such as the one provided here show how poor a proxy weak generative capacity can be. If one aims to study the computational demands imposed by a linguistic formalism, it would seem best to study those computational demands *directly*, since, as the three example LF constraints demonstrate, it may well turn out these are far less than what weak generative capacity indicates. In section 4 we suggest more generally that the proper analysis of multi-level linguistic formalisms, LF, or other constraints should be cast as *complexity problems* rather than as the weak generative capacity analysis of *languages*. For example, the disjoint reference *problem* for English may be stated as follows: Given an arbitrary sentence of English, does there exist an assignment of indices to the names or definite NPs in the sentence that meet the disjoint reference constraint (as usually stated)? As is generally the case with problem statements, this one abstracts away from a consideration of *how* exactly disjoint reference is to be determined—whether by "syntactic" or "semantic" means, whether by a one-level or two-level theory, what algorithm is to be used, and so forth. It poses an abstract computational problem, in the sense intended by MARR, and thus effectively sidesteps the difficulty of how a formalism sets the boundaries of "syntax" and "semantics", or whether there is any distinction between the two at all. It also explicitly incorporates the grammar as an object of investigation (in contrast to a weak generative capacity analysis, which explicitly disavows the grammar's role). (See [2] and [3] for a more extensive discussion and justification of this point.) In contrast, the laborious process of first determining the class of languages fixed by a disjoint reference-obeying LF would seem to be a more indirect way of grappling with the disjoint reference constraint.

---

[4] While it is well known that a first order predicate calculus language enforcing such contraints as nonvacuous quantification or no-free-variables is non-context-free (see, e.g., [11], p. 231, or [15]), such a result has never been directly obtained for natural languages. Further, this example refutes a conjecture raised in [15] that "additional constraints" could "in principle cut" [English LF—rcb] "down to a CF subset" (p. 188).

2. ENGLISH LF IS NOT A CONTEXT-FREE LANGUAGE

We now demonstrate informally that English logical form is a non-context-free language. The result follows from the disjoint reference constraint for names or epithet NPs generally: if such an element X c-commands another name or epithet NP Y, then X and Y may not be coindexed, e.g., in the sentence below, the occurrences of *Reagan, the aging actor, and the current President* and must be distinct:

(2) Reagan said that the aging actor thought that the current President likes ice-cream.

This is conventionally indicated by subscripting the occurrences with distinct indices:

(3) Reagan$_j$ said that the aging actor$_k$ thought that the current President$_l$ likes ice-cream, $j \neq k, k \neq l, j \neq l$

Intuitively, we can show that any LF language that includes such a constraint must be non-context-free for the following reason. Any context-free language permits one to arbitrarily duplicate certain embedded segments if they are long enough, and still obtain a string in the original language. If the set of LFs obeying disjoint reference formed a context-free language, then one ought to be able to carry out this conventional "pumping". However, considering a variant of (2) above, we note that if we duplicated a middle portion *that the aging actor$_j$ thought*, we would obtain an LF with a nondisjoint index. This would violate DR and hence must be a string not in the original LF language:

(4) Reagan$_i$ said that the aging actor$_j$ thought that the aging actor$_j$ thought that the current President$_l$ likes ice-cream

Since this string is not a valid English LF, our original assumption that the English LFs obeying disjoint reference formed a context-free language must have been incorrect. Indeed, since the language $A = \{a^j b^k c^l | j \neq k, k \neq l, j \neq l, \forall \text{ integers } j, k, l > 0\}$ is well known to be non-context-free (see [12], p. 130), it should not be surprising that the DR constraint would lead to non-context-freeness.
Appendix A gives the details.

3. LF CONSTRAINTS CAN BE EFFICIENTLY PROCESSED

In this section we show that while English LF, and presumably the LF for other natural languages, is strictly non-context-free, this fact does not necessarily imply any processing difficulty, because constraints like disjoint reference and nonvacuous quantification are all quickly computable. This result impugns weak generative capacity as a good indicator of computational tractability.

We first show that the DR constraint is efficiently computable and then repeat the demonstration for the standard formulation of the nonvacuous quantification constraint. The result for the no free variables constraint is similar and is omitted. While we consider only these two aspects of LF here, it is straightforward to sequentially check LF for other conditions, e.g., that labeled brackets are properly paired. Appendix B gives the details along with a worked-out example. We assume a finite number of names.

3.1. Assumptions

Logical form representation. We take LF to be roughly bracketed S-structure, with quantifiers in Complementizer positions binding quantified NP positions. We assume that Names occur singly under NP nodes, and that there is no vacuous LF branching (e.g., a degenerate tree of the form NP–NP–NP–...–Name); note that this latter assumption follows directly from a formalism such as that adopted in [14]. We ignore *wh*-bound variable coindexing and all other dependencies such as predication and the like that might be indicated by subscripting, or superscripting. These may be independently (sequentially) checked by a similar algorithm. Finally, as noted above, we will assume that LF strings contain properly paired labeled brackets; if not, this condition is easy to check in advance; a procedure to do so will be omitted here.

Disjoint reference. We must state precisely what we take the DR computational problem to be. The input to the algorithm will be a well-formed LF, without indices for names. Therefore, we may assume that each such (bracketed) LF is unambiguous. The output from the algorithm will be the same LF, but with a valid pairing of indices and names or epithets. Thus, we define the DR problem to be one of disjoint reference *construction* (finding a valid disjoint indexing), rather than disjoint reference *verification* (verifying that a given indexing assignment is valid). At first glance, this would seem to be the wrong problem formulation because it seems to make the DR problem rather trivial: after all, all one has to do to get *some* valid index assignment is simply give all names or epithets distinct indices. One could do this simply by maintaining a counter and just incrementally assigning unary numbers to names, left to right, in the LF string. Of course, this would omit all possible cases of coreference. Therefore, it would seem more valid to solve the DR *verification* problem. However, as we shall observe, the algorithm given in Appendix B is not this trivial one and may be readily used for disjoint reference verification as well. We present the constructive version because it is somewhat simpler to understand.

Measurement of algorithm complexity. Algorithmic complexity is standardly evaluated with respect to a Turing Machine (TM) computation model: as a measure of time, we count the number of primitive instruction steps it takes for an algorithm to solve the DR problem on a 1-tape TM, as a function of the length of input formulas, n. As is also conventional, we will consider a problem to be *tractable* if it has an algorithm solving it that takes no more than n^j time steps, for some fixed integer j. (This includes all polynomial functions such as n^2, $n^3 + n$, and so forth, and is therefore called *polynomial time*.)

3.2. An algorithm for disjoint reference

We now proceed to describe informally an algorithm that will assign a valid set of disjoint name indices to an input LF without such indices, and follow with a more precise description. Though the algorithm considers only names, an extension to epithets is straightforward.

To make the demonstration easier, we shall use a turing machine that has 3 work tapes

and one read-only input tape. The input tape will initially hold the LF formula to process. Of the three work tapes, one will hold a list of bracket symbols and nonaccessible Name and index pairs, with such Name-index pairs currently unavailable for coindexing with other Names; one tape will hold a list of accessible Names and indices, possibly available for coindexing; and the third tape will hold a counter indicating the numerically largest index assigned so far. We will call the first tape the ACCESS tape; the second the DISJOINT tape, and the third the COUNTER. Indices will be encoded in unary. As is well known, such a 3-tape machine algorithm could be converted into a 1-tape machine algorithm while at worst squaring the time; therefore, if there is a polynomial-time algorithm on a 3-tape TM, there is a polynomial-time, hence tractable algorithm on a standard TM.

The intuition behind the algorithm is as follows. A given formula will be processed in one pass, left to right. The DISJOINT tape will be used to store left and right labeled brackets in order to compute c-command domains, as well as store names and indices that are not allowed to be co-indexed. As labeled left brackets indicating the start of a (branching) c-command domain are encountered, they will be placed on the DISJOINT tape in pushdown stack order, as will names. Before a name is placed on the DISJOINT tape, the algorithm will check the ACCESS and COUNTER tapes to see (1) what the next available index should be; (2) whether an already-assigned index can be used. As appropriately labeled closing right brackets are encountered, these will complete c-command domains. responding to this, the algorithm will remove name-index pairs from the DISJOINT tape in pushdown stack order, from right to left, up to the corresponding left bracket, and make those name-index pairs available for co-indexing by placing them on the ACCESS tape to the right of any existing name-index pair. Irrelevant brackets and tokens will be skipped over in the input and ignored.

By the way the algorithm is designed, at any one time all names within the same c-command domain are on the DISJOINT list. This makes it easy to enforce the DR constraint, by checking whether the indices assigned to these names are distinct.

3.3. An algorithm for nonvacuous quantification

Simply stated, the nonvacuous quantification constraint (NVQ) says that every quantifier $\forall x_i$, $\exists x_j$ (i, j again in unary) must bind some corresponding variable. This amounts to two constraints: (1) there must exist such a corresponding variable somewhere to the right of the quantifier; (2) there must be no other intervening, c-commanding quantifier that binds the same variable. Note that x here is a fixed terminal in the LF vocabulary.

An algorithm for *verifying* this constraint can use the same approach as the one for DR. Since the NVQ problem is one of verification, we assume an input LF with indices already assigned. To further simplify matters, we assume that all lexical quantifiers such as *all*, *very*, etc., are translated to occurrences of $\forall$, $\exists$, while *wh* quantifiers are translated as the token *wh*; the extension to a finite number of other quantifiers is straightforward. It is also irrelevant whether we assume that an entire quantified NP is moved to a Complementizer position or just, say, only a Determiner is moved.

Using a tape called QUANT, and another worktape, the algorithm works essentially by stacking quantifiers within c-command domains until potentially matching candidates are found. We omit the details here. The algorithm is designed to place c-commanding quantifier domains in pushdown stack order on QUANT. Any time an empty category—index pair is encountered, the first possible quantifier list is checked for a matching index, and this quantifier-index pair is eliminated. If any c-commanding domain contains a quantifier that has no matching variable, then such a domain contains a vacuous quantifier, and the input LF is rejected. Relevant c-command domains are marked off by $\bar{S}$ pairs. Note that the algorithm handles the case where the Complementizer position could be on the right rather than on the left, since in this case the variable index is saved on the worktape to wait for a possible quantifier in Complementizer position. (This is irrelevant for the DR' constraint, which depends strictly on c-command and not on the Complementizer position.)

The no-free-variable constraint (every variable must be bound by some quantifier) may be handled in a similar manner.

4. DISCUSSION AND CONCLUSIONS

The main results of this paper can be put quite briefly: (1) Because of disjoint reference, English LF is not a context-free language; (2) this weak generative capacity finding does not apparently imply anything about sentence processing, since DR (and two other LF constraints) is efficiently computable with a finite number of names.

Beyond these narrow findings, the results have broader implications for the study of the relationship between representational levels in linguistic theories. Other researchers have considered the computational complexity of multiple-level natural language theories, and it is worthwhile to compare their results to those described here.

First, it seems likely that other LF components not considered here, such as pronominal indexing, make LF processing still more complex. CORREA proposes a nondeterministic algorithm to compute all LF indexing [8], and a preliminary, informal analysis indicates that his approach would make a single left-to-right pass through an LF string in time proportional to the length of the string. Straightforward conversion to a deterministic algorithm would demand time proportional to 2^{cn}, for some constant c. However, since this is just one possible algorithm it remains to establish the complexity of the LF *problem* in full. There may be a better deterministic algorithm.

MARSH and PARTEE investigate the non-context-freeness of the NVQ constraint and no-free-variable constraint as applied to fragments of the first order predicate calculus [15]. Both of these languages are well known to be non-context-free, as pointed out for example in [10]. MARSH and PARTEE show that the no-free-variables (NFV) constraint is b a strictly context-sensitive language, in fact, can be generated by an indexed grammar as defined by AHO in [1].[5] Intuitively, this means that one can check the NFV constraint

[5] The languages generated by indexed grammars form a strict subset of all the context-sensitive languages.

by encoding at each variable name a string of indices that c-command it. Then, one can use this encoded list as a means to control the expansion of the variable name into at least one of those indices, thus ensuring that there will be no free variables in the resulting logical form sentence. In contrast, MARSH and PARTEE suggest that NVQ is not an indexed language, and they show why the index checking power of an indexed grammar seems to fall short of what is needed for NVQ.

It is interesting to compare MARSH and PARTEE's approach with that presented here. MARSH and PARTEE do not investigate the NVQ and NFV *problems*; rather, they aim to establish the weak generative capacity of various artificial *languages* that embody the NVQ and NFV constraints. This weak generative capacity approach reveals its own limits: (1) no results about the actual computational complexity of the constraints are forthcoming; (2) since the "programming" ability of an indexed grammar is so limited, one quickly runs into difficulty establishing just what sort of languages NVQ describes; (3) questions arise as to the division between "syntax" and "semantics". Note that even if one established that NVQ was a non-indexed language, that would leave open a question of its recognition complexity, as is the case with NFV when encoded as an artificial language. In contrast, the results given here show directly that NVQ and NFV are polynomial-time computable, without resorting to intermediate weak generative capacity arguments. Further, as noted below, by posing linguistic questions as problems one can bypass essentially irrelevant disputes about syntax and semantics.

Turning to other approaches, BORGIDA models stratificational grammar as a restricted mapping from one (generally context-free) language to a second language, where the first language may be taken as a syntactic description, the second an LF-type description [5]. Though drawing on the characteristic vocabulary of stratificational theory, the model seems flexible enough to apply to any theory that decomposes an overall grammatical description into distinct representational levels.

BORGIDA obtains several weak generative capacity and recognition complexity results relevant in the current context. For example, suppose that both representational levels are described by context-free grammars, and that the second level ("LF") contains no empty elements, as is the case since coindexed empty categories are actually terminal elements in LF. Suppose in addition that the mapping between S-structure and LF is linear — given some S-structure string, the corresponding LF string is at most k times larger, for some k fixed in advance. This constraint seems plausible, and is given some justification in [4]; we will put aside the question of indexing for the moment, which complicates matters.

If these assumptions hold, then BORGIDA shows that the set of all languages generable by all such context-free two-level grammars is exactly the set of *quasi-realtime languages*. These are the languages recognized by a non-deterministic Turing machine in time exactly n, where n is input sentence length; they include some strictly context-sensitive languages[6]. Because the quasi-realtime languages are probably not recognizable in polynomial time, this result would seem to conflict with the polynomial time DR and NVQ algorithms given earlier. However, it is not yet clear how BORGIDA's results mesh

[6] Recall that a Turing machine is *deterministic* if for every combination of state, input symbol, and tape symbols there is exactly one move it can make, and is *nondeterministic* otherwise.

with the LF models proposed in government-binding theory and in this paper. First, we have discussed just three LF constraints; others may indeed be more complex. Second, if we insist that the indices themselves be encoded as part of LF, then a logical form is more than linear in the size of S-structure.[7]. Since constraining LF makes it easier to process, perhaps a less constrained, non-linear LF wil be *harder* to process. Still, it remains to fix precisely the computational complexity of LF problems generally.

PLÁTEK and SGALL, using an algebraic approach, obtain results similar in kind to BORGIDA's. They show in [16] how to model multiple-level grammatical theories as follows: Start with the set of all context-free languages, T_1. Then, translate T_1 to a new set of languages T_2, by applying all possible deterministic, linear, pushdown automaton transductions, denoted M. That is, for some integer k fixed in advance, the mapping may not reduce or expand strings in T_1 by more than a factor k. One may imagine a pushdown automaton *transducing* an input by processing some input string, say, an S-structure, and outputting a new, translated string, say, a logical form. Then $T_2 = M(T_1) \not\subseteq T_1$, and in fact T_2 includes some strictly context-sensitive languages (such as $\{ca^n ca^n ca^n\}$, for all integers n). Suitably tailored to conform to BORGIDA's restrictions, it may be conjectured that T_2 is in fact precisely the class of quasi-realtime languages.

By posing DR or any grammatical constraint as a *computational problem* we can dispose of computationally irrelevant distinctions between syntax and semantics, or the boundaries between them. If a problem is computationally intractable, it may be presumed intractable with respect to any proposed algorithm that makes the usual assumptions of complexity theory. But this means that any reasonable way of dividing the problem into distinct "syntactic" and "semantic" languages cannot be any easier, for this would in turn imply a tractable solution to the intractable problem in the first place.[8] On the other hand, if we can show that a grammatical problem has an efficient solution, then this may involve a sophisticated algorithm that does not abide by conventional notions of "syntax" and "semantics". In general, we can model a grammatical problem as the mapping between two languages, and pose computational problems with respect to one or another of these, without labeling one language "syntax" and the other "semantics". This would seem to be the most neutral possible stance; on either possibility, by formulating a grammatical constraint as a computational problem, we can not only draw on the usual tools of modern computer science, but we can also attack the processing demands of grammar directly, without the possibly misleading proxy of weak generative capacity.

Acknowledgements. The idea that the DR problem could lead to a non-context-free language first arose in a discussion with G. Edward BARTON, Jr. Noam CHOMSKY provided additional comments.

[7] If a sentence n S-structure tokens long contains proportional to n indexable elements, then the corresponding LF will be of length $n + \sum_{i=1}^{i=n} \log i > kn$, for fixed k. An alternative is to adopt the approach in [4], and assume that indexable items are drawn from an indefinitely large terminal vocabulary, e.g., that each occurrence of *Susan* is distinct to begin with. The implications of this latter approach will not be pursued here.

[8] By "reasonable" we mean roughly any way that does not hide an exponential amount of computation.

This research has been supported in part by grants from the National Science Foundation grants DCR-8552543, under a Presidential Young Investigator Award, and by support for the Artificial Intelligence laboratory provided in part by the Advanced Research Projects Agency under Office of Naval Research contract N00014-85-K-1024.

A. DETAILS OF THE CONSTRUCTION THAT ENGLISH LF IS NON-CONTEXT-FREE

A.1 Assumptions

In order to proceed, we must make certain assumptions regarding LF, indexing, c-command, and the disjoint reference constraint for names.

Logical form representation (LF). LF is assumed to consist of essentially bracketed S-structure, plus the application of Move-alpha to quantified NPs that moves such constituents into Complementizer positions, leaving behind variables. Such elements, along with *wh*-phrases, are coindexed along the lines suggested by CHOMSKY in [6] and HIGGINBOTHAM in [10], among others. Following [6], coindexing is assumed free unless otherwise constrained. Each sentence of logical form is unambiguous, though there may of course be more than one LF associated with a given surface sentence. For example, the following LF would correspond to the surface sentence, *What did John eat* (ignoring irrelevant details and the matter of indices, to which we return immediately below):

(5) $[_S [_{Comp}\ What_i]_{Comp} [_S [_{NP}\ John_j]_{NP} [_{VP}\ eat\ x_i]_{VP}]_S]_S$

More completely, we assume that the language of English LFs is described over an alphabet Σ that includes the symbols [and], appropriately labeled by S, NP, VP, etc.; 1 (the latter to encode subscripts in unary, see below); a special symbol # to replace blanks; plus the usual orthographic symbols required to write down English words. As is standard, we may replace all occurrences of the blank space following each word in English logical forms with the symbol # by inserting a # to the right of each word. Thus the string

(6) $[_S$ John$_1$ said $[_S$ that John$_{11}$ said $[_S$ that Sally$_{101}$ likes ice-cream]]]

becomes

(7) $[_S$ # John$_1$ # said # $[_S$ # that # John$_{11}$ # said $[_S$ # that # Sally$_{101}$ # likes # ice-cream]]]

Indices. It is conventional in the linguistics literature to indicate coindexing via integer subscripts, as above, e.g., x_i, x_j, and so forth. We could encode these integers as binary numbers. Alternatively, one could encode integers in *unary*, with a 1 denoting 1; 11 denoting 2; 111 denoting 3, and so forth. This one could write as x', x'', etc., or, more transparently, x_i, x_{ii}, etc. Each of the i tokens counts as a distinct terminal word in the LF string, just like *John* or left and right labeled brackets. To be absolutely clear about this, for example, (5) will be written with indices as follows, where spaces have been left between tokens for readability. (note that any number of i's besides two would count as a valid index for the *wh* word and the coindexed variable).

(8) $[_S [_{Comp}$ *What iii* $]_{Comp} [_S [_{NP}$ John *i* $]_{NP} [_{VP}$ eat x *iii* $]_{VP}]_S]_S$

We adopt the unary encoding here, because it simplifies the proof; on the other hand, as described in section 3, it does make some of the algorithm complexity calculations slightly more complicated.[9] As part of the LF language itself, the indices should be thought of as generated by some underlying grammar.

C-command. A node α *c-commands* a node β if the first branching node that dominates α dominates β. Any of the common variants of c-command could be assumed, without altering the basic result.

Disjoint reference. Given two referential NPs X, Y, denoting names or epithets, if X c-commands Y then the indices for X and Y must be distinct.

A.2 A proof that English LF is not context-free

As is standard, we may show that well-formed English LFs do not form a context-free language using a proof by contradiction. We first assume that such sentences do form a context-free language. We then apply certain operations on this language known to preserve the property of being context-free. Finally, we show that the altered language is not context-free. We therefore conclude that our original assumption that the well-formed English LFs form a context-free language must be false.

The demonstration has four steps. We begin with the set of all English well-formed LFs. First, we will erase all brackets. This prepares us for the next step, which is to intersect all such debracketed English LFs with a regular set that filters out all such forms save for those that have a particularly simple structure, with three sets of indices. The reason for excising the brackets in the first place is to ensure that this filtering set can be regular set (one generated by a finite-state grammar), so that the intersection will preserve context-freeness. Third, we apply a general finite-state mapping that will produce our test language A that we can show to be non-context-free. Let us examine in more detail how this works, following along with an example. We start with the LF:

(9) $[_S [_S [_{NP}$ John *i* $]_{NP} [_{VP}$ said $[_S$ that $[_S [_{NP}$ John *iii* $]_{NP} [_{VP}$ said $[_S$ that $[_S [_{NP}$ John *iiiii* $]_{NP}$ $[_{VP}$ likes $[_{NP}$ ice-cream$]_{NP}]_{VP}]_S]_S]_{VP}]_S]_S$

Step 1. Apply a homomorphism h that erases all brackets. This step preserves context-freeness.[10] Applied to our example sentence, the output of h would be:

(10) John *i* said that John *iii* said that John *iiiii* likes ice-cream.

Construction of h: the debracketing homomorphism h may be described as follows.

$h(a) = \emptyset$, $\forall a \in X$, where X is a finite set of brackets $\{[_S,]_S, [_{NP}, \ldots\}$
$h(a) = a$, otherwise

Plainly, h "erases" brackets. A proof is omitted.

[9] Section 3 briefly describes what it would mean to shift to a binary or other radix for encoding indices. This would not alter the basic result.

[10] It is well known that h preserves context-freeness; see [12].

Step 2. Intersect the debracketed language with a regular set F that filters out everything *except* those debracketed LFs in the following form, where superscripts indicate repetition:[11]

(11) John i^j said that John i^k said that John i^l likes ice-cream
$\forall$ integers $j, k, l > 0$ and $j \neq k$, $k \neq l$, $j \neq l$.

Thus, our example (10) would be admitted by this filter, while the following string would be blocked, because it contains nondisjoint subscripts:

(12) John i said that John i said that John ii likes ice-cream

The filtering set F itself is simply the set of all sentences in the form in (11) but also including *all possible* subscripts, both disjoint and nondisjoint. As is conventional, the plus sign + is Kleene plus, denoting 1 or more repetitions of an element.

(13) John i^+ said that John i^+ said that John i^+ likes ice-cream.

Plainly, F is a regular language because it is a regular expression. F fails to obey disjoint reference and in addition includes only a small subset of possible debracketed English LFs. When we intersect this regular filter set with the set of all well-formed English debracketed LF expressions that must obey the noncoreference constraint, only those with (properly) noncoindexed subscripts can be left behind.

Step 3. Apply a *general sequential machine mapping* g—*a* generalized homomorphism based on a finite-state machine—that (1) erases all tokens in the given string up to the first string of is, and replaces the i's with a's; (2) erases the next string of tokens up to the next string of i's, and replaces them with b's; and finally, (3) erases the remaining tokens up to the final string of i's, and replaces them with c's. This machine mapping preserves context-freeness (see 9, pp. 200—205).

In our running example, g would output the new string.

(14) *abbbccccc*, e.g., ab^3c^5

In general, step 3 must yield all strings of the form,

$$a^j b^k c^l,\ j \neq k,\ k \neq l,\ j \neq l,\ \forall \text{ integers } j, k, l > 0$$

This general finite-state machine mapping g is called a *sequential transducer* in [9], p. 199. This is simply a finite-state machine that maps input strings to output strings, by associating some string of output symbols that are paired with each transition in the finite-state machine. Theorem 6.4.3 of [9], p. 200 shows that if L is a context-free language and S is a sequential transducer, then $S(L)$ is a context-free language.

The g we need to convert the examples above into language A may be described as in Table 1. By convention, machine transitions between states s_1, s_2 are described by a pair $x:y$, where x as usual gives the symbol that allows a transition between s_1 and s_2, while

[11] Again, the intersection of a regular set and the set of strings determined by a context-free language is a context-free language; see [12].

Table 1: The sequential transducer (general sequential machine, or gsm) that maps strings of the form in (13) to a string of a's and b's. Possible transitions not listed lead to rejection.

State	x:y pair	New state
s_0	John: nil #: nil *i*: *a*	s_0 s_0 s_1
s_1	*i*: *a* #: nil	s_1 s_3
s_2	#: nil *said*: nil *i*: *b*	s_2 s_2 s_3
s_3	*i*: *b* #: nil	s_3 s_4
s_4	*i*: *c* *said*: nil *that*: nil *John*: nil *likes*: nil *ice-cream*: nil	s_4 s_4 s_4 s_4 s_4 s_4
s_4	(no input)	Accept

y is the symbol output as a result of making that particular transition. The initial state is s_0, and the final state is s_5. The machine accepts if it is in a halting state with no more input. The alphabet of the machine includes a finite input and output vocabulary. It should be apparent that the input vocabulary must include the symbols in the relevant example string, i.e., *John, said, that likes, ice-cream, i, #*. The output vocabulary consists of a blank whitespace symbol, denoted *nil*; *a*; and *b*. It is easy to check that g meets the requirements for a sequential transducer. Clearly this general sequential machine maps the strings described by [11] into the language A.

Step 4. Show that

$$A = \{a^j b^k c^l | j \neq k, k \neq l, j \neq l, \forall \text{ integers } j, k, l > 0\}$$

is not a context-free language, thereby obtaining a contradiction and concluding the proof. This fact about A is proved in [12], p. 130. The proof can be carried out quite easily by applying a strong "pumping" lemma for context-free languages. According to this lemma, given any context-free language, and any sentence of that language, if we "pump" (duplicate) a particular, long enough portion of that sentence, we must obtain a sentence that is still in the language we started with. But this fails to hold for A. Intuitively this is so because if we duplicate that portion of a string in A containing just one index (say, only the a's), we must obtain duplicated subscripts as in sentence (12), thereby obtaining a sentence not in A. □

B. POLYNOMIAL-TIME ALGORITHM FOR DR

For disjoint reference we give the algorithm itself and follow with an example. We then show that the algorithm correctly obeys the c-command condition, and that its running time is no more than polynomial in the length of input LFs. In the algorithm that follows, we omit obvious details of the Turing machine construction. The control flow passes directly from top to bottom, aside from conditional statements and go-tos back to Step (1). Important: only a *finite* number of names LS assumed.

Algorithm for DR:

Initialization: all three TM work tapes are blank and the TM is scanning three (arbitrary) initial positions on them. The input tape holds the LF formula to process, and the TM is scanning the first blank tape cell to the left of the input formula.

(1) Read the next input token I of the LF *Comment* at the start this will be the first, lefmost token of the input
and **do**
If $I = \emptyset$ *Comment* there are no more input tokens
then if DISJOINT $= \emptyset$, halt and *accept*;
else *reject*
Else go to Step (2)

(2) **If** $I \in BLN$, BLN = a finite set of *left bracket branching nodes* $= \{[_S, [_S, [_{VP}, [_{NP}\}$ **then do**
Copy I onto the DISJOINT tape to the right of all current symbols on that tape
Go to Step (1)

(3) **If** $I = [_{Name_i}$ **then do**
(a) Read the next two input tokens $I + 1$, $I + 2$ and **do**
If *category* $I + 1 \neq$ Name $\vee$ $I + 2 \neq [_{Name_i}$ **then** *reject*
(b) Scan DISJOINT from right to left and **do**
If the rightmost token on DISJOINT $= [_{NP}$ **then** delete from DISJOINT
else *reject*
(c) Scan ACCESS from right to left and **do** *Comment* note that ACCESS tape contains Name-index pairs in adjacent tape cells;[12]
(d) **If** $\exists$ token on ACCESS $= I$ **then do**
Copy after I the same index as the one paired with the token on ACCESS just matched
Put the I-index pair on the DISJOINT tape
else do
Consult the COUNTER tape
Increment the value on COUNTER by adding a 1 to the first blank tape square on COUNTER to the right of the scanning head
Copy I onto the DISJOINT tape followed by the value on the COUNTER tape to the immediate right of the existing contents on the DISJOINT tape
(e) Copy $I + 1$ as the lefmost symbol on the DISJOINT tape
(f) Go to Step (1);

[12] We omit obvious details of how this matching could be done rapidly with 2 tapes; see [12], p. 287 for a similar procedure for a different language.

(4) **If** $I \in BRN$, where BRN = a finite set of *branching right bracket nodes* = $\{]_S,]_S,]_{VP}\}$ $\vee$
If $I =]_{NP} \wedge$ the rightmost symbol in DISJOINT $\neq]_{Name}$ **then do**
Copy all Name, index pairs on DISJOINT leftwards back to the first occurrence of a labeled left bracket of the same type onto the ACCESS tape, starting at the rightmost nonblank tape square **unless** any such Name-index pair already exists on ACCESS
Go to Step (1)

(5) Skip the input symbol and go to Step (1).

Note that this algorithm will not construct *all possible* valid index assignments. This is because it will stop at the first matching name in the ACCESS list to find a candidate coreferent Name. It is straightforward to modify the algorithm so that it looks through the entire ACCESS list and picks a matching Name at random.

An example. Consider the operation of the algorithm on the following conjoined sentence: [*John said that John likes Sue*] *and* [*John said that Bill likes Sue too*]. Its nonindexed LF is:

(15) $[_S [_S [_{Comp}\ ]_{Comp} [_S [_{NP} [_{Name}$ John $]_{Name}]_{NP} [_{VP}$ said $[_S [_{Comp}$ that $]_{Comp} [_S [_{NP} [_{Name}$ John$]_{Name}]_{NP}$
$[_{VP}$ likes $[_{NP} [_{Name}$ Sue$]_{Name}]_{NP}]_{VP}]_S]_S]_{VP}]_S]_S$
and
$[_S [_{Comp}\ ]_{Comp} [_S [_{NP} [_{Name}$ John $]_{Name}]_{NP} [_{VP}$ said $[_S [_{Comp}$ that $]_{Comp} [_S [_{NP} [_{Name}$ Bill $]_{Name}]_{NP}$
$[_{VP}$ likes $[_{NP} [_{Name}$ Sue $]_{Name}]_{NP}]_{VP} [_{Adv}$ too $]_{Adv}]_S]_S]_S$

The first three brackets are left branching, and hence Step (2) applies. The fourth Comp bracket is skipped by Step (5), while the fifth, $[_{NP}$, is deleted by the next iteration at step (3a) because it is followed by a Name bracket. The actual name *John* is also read under Step (3a). The ACCESS tape is blank, so Step (3d) fails to get a match and the *else* clause is executed: the COUNTER tape is incremented by adding a 1 and *John* along with its index, a single 1, is placed onto the DISJOINT tape. Step (3e) places the right Name bracket on DISJOINT. At this point, the contents of the DISJOINT, ACCESS, and COUNTER tapes are as follows:

(1) DISJOINT: $[_S [_S [_S$ John 1 $]_{Name}$
ACCESS: blank
COUNTER: 1

Next, the algorithm processes the sequence $[_{VP}$ said $[_S [_{Comp}$ that $]_{Comp} [_S [_{NP} [_{Name}$ John$]_{Name}]_{NP}$. As before, Step (2) will apply to the branching node brackets, placing them to the right of the current material on DISJOINT, while intervening words and the Comp brackets will be skipped. When the NP bracket is reached, Step (2) copies it onto DISJOINT, but it is erased on the next round by Step (3b). The remainder of Step (3) will operate as before. Since the ACCESS tape is blank the COUNTER will be incremented by 1, and made the index placed alongside the next occurrence of *John* on the DISJOINT tape. Finally, the closing Name bracket will be added. Now the tape contents will be as follows:

(2) DISJOINT: $[_S [_S [_S$ John 1 $]_{Name}$ $[_{VP} [_S [_S$ John 11 $]_{Name}$
ACCESS: blank
COUNTER: 11

The algorithm then processes the next portion of the LF input, $[_{VP}$ likes $[_{NP} [_{Name}$ Sue $]_{Name}$ $]_{NP}$ $]_{VP}$ $]_S$ $]_S$. The VP branching bracket will be placed on DISJOINT, *likes* is skipped, and by step 3 the NP-Name combination will place Sue followed by the incremented index 111 on the DISJOINT tape.

Next the algorithm first processes closing c-command domains. First, the bracket $]_{VP}$ is matched back to its left-hand mate. By Step (4), we copy the pair (Sue 111) onto the ACCESS tape, thereby making it available for coindexing, and removing all other material in between; this is correct, since *Sue* now can no longer c-command any remaining material in the sentence. Next, the bracket $]_S$ is processed; this prompts us again via Step (4) to copy (John 11) onto the ACCESS tape to the right of Sue, 111. Finally, the bracket $]_S$ is read and according to Step (4) simply causes the machine to go back and erase the first corresponding left bracket. After all of this, the contents of the three work tapes look like this:

(3) DISJOINT: $[_S [_S [_S$ John 1 $]_{Name} [_{VP}$
ACCESS: Sue 111 John 11
COUNTER: 111

The next part of the input LF string is $]_{VP}$ $]_S$ $]_S$. These closing brackets will prompt Step (4) to pull off the pair John 1 and move it to ACCESS, while removing all that remains on DISJOINT except for a single $[_S$, as is correct:

(4) DISJOINT: $[_S$
ACCESS: Sue 111 John 11 John 1
COUNTER: 111

The TM will now process the second conjunct. The new Names *John* and *Sue* may be coindexed with the previous occurrences of *John* or *Sue*, and we note that the ACCESS list holds, appropriately, all these previous occurrences.

First, as before the algorithm will place the opening brackets up to the name *John* onto the DISJOINT tape. Next, Steps 3(a) through (3d) execute. The TM obtains a match with the rightmost *John* in ACCESS. Therefore, by Step (3d), we do not increment the counter but assign the index 11 to this new occurrence of *John*.

We omit the operation of the system with the Name *Bill*; it is made disjoint from *John* and *Sue*. Similarly, the rest of the input LF will be processed and the next occurrence of *Sue* will get the same possible index 111 as its previous occurrence. At this point the work tapes will look like this:

(5) DISJOINT: $[_S [_S [_S$ John 1 $]_{Name} [_{VP} [_S [_S$ Sue 111 $]_{Name}$
ACCESS: Sue 111 John 11 John 1
COUNTER: 111

Finally, the closing bracket sequence will erase the DISJOINT tape while *not* copying any duplicate Name-index pairs onto the ACCESS tape. Since there is no more input and DISJOINT is empty, the TM halts and accepts, by Step (1).
We now verify informally that the algorithm works correctly and executes in polynomial time.

Claim 1. The algorithm correctly enforces DR.

Proof. By Step (4), a Name can appear on the ACCESS tape if and only if a branching node bracket domain enclosing the node has been processed. By definition, these are Names that can no longer c-command any other material in the sentence and hence can be coindexed with any later Names that appear, as computed by Step (3d). By Step (3), a Name is on the DISJOINT tape if and only if an opening branching bracket has appeared that encloses that Name but no closing branching bracket. By definition, since they are in the same branching domain, Names to the left in this tape must c-command Names to the right, and by Step (3d) they are given distinct indices. □

Claim 2. The algorithm will take at most kn^3 primitive steps on a 3-tape TM, where k is some constant fixed in advance and n is the length of the input LF.

Proof. To see this, we consider each of the algorithm's steps, and determine which takes the most time for processing any *single* input token. We then multiply this worst-case possibility by n possible tokens in the input to get the execution time as a function of the input length.

Step (1), reading the input and checking for the end of the input, will take at most some constant number of primitive steps.

Step (2), checking for an opening branching node bracket, will take at most a constant number of primitive steps, since the number of such brackets is fixed in advance and may be stored in the finite control of the TM. Depending on the location of the read/write head for DISJOINT, copying the bracket symbol could take at most time proportional to n^2, or $c \cdot n^2$ (see analysis for Step(3c)). If executed, the go-to takes at most a constant number of primitive steps.

The equality check at the beginning of Step (3), the two read steps of (3a) and the **if-then** clause will take at most a constant number of primitive steps.

Step (3b), scanning for $[_{NP}$, can take time proportional to the length of the disjoint tape. But this tape itself can be at most n^2 long (see analysis for Step (3c)), so the total time here is at worst $c' \cdot n^2$.

Step (3c), matching against the ACCESS tape, will take at most time proportional to the length of that tape, since the TM has 2 scanning heads. Since ACCESS can contain at most n Names in an input n tokens long, this can take at most proportional to n^2 primitive instructions (because the tape contains Name-index pairs, and each index itself can be at most n symbols long, we could need up to proportional to $1 + 2 + \ldots + n$, or $c'' \cdot n^2$ primitive steps to scan the indices and n primitive steps to scan the Names themselves).

Step (3d), copying a matched Name-index pair onto the DISJOINT tape, could take

at worst time proportional to n, for copying and moving the writing head on the DISJOINT tape. Likewise, incrementing COUNTER and copying the new Name-index pair to DISJOINT, can take no more than time $c''' \cdot n$, since this is as large as an index or the counter may get.

Step (4), reading and copying names from the DISJOINT tape to the ACCESS tape, then going to Step (1), can take no more than time $c \cdot n''' n^2$, since again there can be no more than n Names and brackets in the DISJOINT list and since the indices can total no more than $1 + 2 + \ldots + n$ long in all. The go-to adds at most a constant number of primitive steps.

Step (5), skipping individual symbols and returning to Step (1), takes no more than a constant number of primitive steps.

Adding things up, for an input n tokens long the algorithm takes no more than time proportional to $n \times n^2 = n^3$ primitive steps. □

As noted earlier, this algorithm will only assign one possible coreferent index to a non-c-commanding Name-index pair. However, it is easy to see that one could modify the algorithm to pick a coreferent at random. It is also easy to see how to *verify* disjoint reference: given an LF *with* indices assigned, the search in Step (3) must check that a new Name-index pair does not match any Name-index currently in the DISJOINT list and may match a Name-index pair in the ACCESS list.[13]

REFERENCES

[1] Aho, A.: Indexed grammars—an extension of context-free grammars. Journal of the Association for Computing Machinery, 1968, 15, pp. 647—671.

[2] Barton, E.: The computational structure of natural language. Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, 1987.

[3] Barton, E.—Berwick, R.—Ristad, E.: Computational Complexity and Natural Language. Cambridge, MA: MIT Press, 1987.

[4] Berwick, R.—Weinberg, A.: The Grammatical Basis of Linguistic Performance. Cambridge, MA: MIT Press., 1984.

[5] Borgida, A.: Some formal results about stratificational grammar and their relevance to linguistic theory. Mathematical Systems Theory, 1983, 16, pp. 29—56.

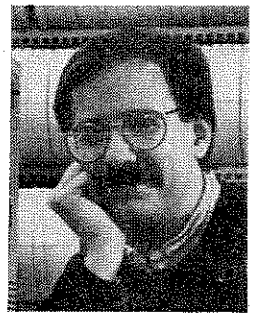
[6] Chomsky, N.: On binding Linguistic Inquiry, 11, 1980, pp. 1—46.

[7] Chomsky, N.: Lectures on Government and Binding. Dordrecht: Foris Publications, 1981.

[8] Correa, N.: An attribute-grammar implementation of GB theory. Ph.D. dissertation, Department of Electrical and Computer Engineering, Syracuse University, 1987.

[13] There is one subtlety. If we are given an indexed LF to verify, then that LF may include arbitrarily large indices. Since index encoding is unary, this can in effect artificially enlarge the input string: for instance, if the index on an NP is arbitrarily one million, then the unary encoding of this number will take a million cells, when it ought to take just log 10^6 cells (if in base 10). This in turn affects the complexity calculation, since that is based on input formula length. If that length is artificially large, then the calculated time might seem small (as a function of the input length) when it is not. But this problem can be sidestepped in the present case, since plainly even if we excluded such pathologic examples, the time to verify a formula would be polynomial in input sentence length. Further, one can use a binary encoding for variable indices, though this significantly complicates the demonstration that A is non-context-free.

[9] HARRISON, M.: Introduction to Formal Language Theory. Reading, MA: Addison-Wesley, 1978.
[10] HIGGINBOTHAM, J.: Logical form, binding, and nominals. Linguistic Inquiry, 1983, 14, 395—420.
[11] HIGGINBOTHAM, J.: English is not a context-free language. Linguistic Inquiry, 1984, 15:2, pp. 225—234.
[12] HOPCROFT, J.—ULLMAN, J.: Introduction to automata Theory, Languages, and Computation. Reading, MA: Addison-Wesley, 1979.
[13] LASNIK, H.: Remarks on coreference. Linguistic Analysis, 1976, 2, pp. 1—22.
[14] LASNIK, H.—KUPIN, J.: A restrictive transformational theory. Theoretical Linguistics, 1976.
[15] MARSH, W.—PARTEE, B. H.: How non-context-free is variable binding? In Proceedings of the Third West Coast Conference on Formal Linguistics, ed. Mark Cobler, Susannah MacKaye, and Michael Wescoat, 1984, Vol. 3, Stanford, CA: Stanford Linguistics Association, pp. 179—190.
[16] PLÁTEK, M.—SGALL, P.: A scale of context-sensitive languages: applications to natural languages. Information and Control, 1978, 38:1, pp. 1—20.



Robert C. BERWICK is an Associate Professor of Computer Science and Engineering at the Artificial Intelligence Laboratory, in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. Since 1982 he has headed the Laboratory's natural language processing research into the interaction between computational complexity and language, language acquisition, and machine translation. Professor Berwick is the author of five books and many articles on natural language processing, language acquisition, and computation, including, most recently a textbook Computational Linguistics (MIT Press, 1990).

AII '89

INTERNATIONAL WORKSHOP ON ANALOGICAL AND INDUCTIVE INFERENCE

Reinhardsbrunn Castle, GDR, October 1—6, 1989

organized by Leipzig University of Technology, Dept. of Mathematics & Informatics
supported by Gesellschaft f. Informatik d. DDR

AII '89 is the second workshop in the AII series started with AII '86 (see Lecture Notes in Computer Science 265). These workshops are focussed to all formal approaches to algorithmic learning particularly emphasizing analogical reasoning and inductive inference.

Topics:
• recursion-theoretic inductive inference • approaches to machine learning • explanation based learning • similarity based learning • formalizing analogy concepts • analogical reasoning • inductive program synthesis • approaches relating different learning concepts to each other.
The workshop is intended to offer opportunities for intensive and detailed discussions. It should especially stimulate new scientific contacts between scientists from Artificial Intelligence and from Theoretical Computer Science. Therefore, the number of participants will be restricted to about 60. There will be no parallel sessions. Additionally, the program committee is preparing some panel discussions.

Program committee:
Klaus P. JANTKE, Chairman (Leipzig, GDR), Setsuo ARIKAWA (Fukuoka, Japan), Jan M. BARZDIN (Riga, USSR), Robert P. DALEY (Pittsburgh, USA), Katharina MORIK (Berlin/West/), Luc DE RAEDT (Leuven, Belgium), Carl H. SMITH (College Park, USA), Leslie G. VALIANT (Cambridge, USA), Fritz WYSOTZKI (Berlin, GDR).

The Conference site
is a pretty old German castle in the hilly area called Thuringian Forest. Lodging will be provided in the castle.

For more information write to:

Prof. Dr. Klaus P. JANTKE
Leipzig University of Technology
Dept. Mathematics & Informatics
P. O. Box 66
7030 Leipzig
German Democratic Republic