Context-Free Parsing of a Tree Adjoining Grammar Using Finite-State Machines

Alexis Nasr*, Owen Rambow†, John Chen‡, and Srinivas Bangalore‡
* *Université Paris 7,* † *University of Pennsylvania,* ‡ *AT&T Labs – Research*
nasr@linguist.jussieu.fr, rambow@unagi.cis.upenn.edu, jchen@research.att.com, srini@research.att.com

1. Introduction: A Dependency-Only Parser

In this paper, we describe work in progress (in fact, in rather preliminary progress) that originated with the goal of creating a dependency parser which does not use phrase structure, but does use an explicit generative (and of course lexicalized) encoding of the grammar.[1] The motivation for this goal is twofold.

- First, we believe that the phrase structure used by many (if not most) linguistic theories is useful only in deriving the syntactic behavior of lexical heads (or classes of lexical heads) from more genral principles(which is of course the goal of syntactic theory); once derived, the syntactic behavior can be expressed in simpler terms that can be easier to manipulate in a computational setting.[2] Furthermore, applications need as output from a parser something close to dependency (typically, lexical predicate-argument structure), but not phrase-structure.[3]

- Second, we prefer an explicit encoding of a grammar in an inspectable declarative syntactic formalism over an implicit encoding in a computational framework. While we believe that explicit grammars have various advantages, such as the ability to test different syntactic hypotheses, or the possibility of hand-crafting grammars for limited domains (as is commonly done in commerical speech-based systems), we do not attempt to justify this preference further.

- Third, we believe that generative formalisms have certain advantages over constraint-based formalisms, in particular computational advantages. A large body of research on parsing in generative formalisms can be reused for different formalisms if they share certain basic peoperties, and we exploit this fact.

The formalism we used is based on that presented in (Kahane et al., 1998), but in the first phase we have omitted the non-projectivity, leaving us with a simple generative algorithm which allows us to lexicalize a context-free grammar by allowing regular expressions in the right-hand side.[4] We call this formalism GDG,or Generative Depenendcy Grammar. Its parsing algorithm, naturally, expresses the regular expressions as finite-state machines (FSMs), and the chart parser records FSMs and the state they are in during the parse.

It is well known that the derivation tree of a Tree Adjoining Grammar (TAG) is a dependency tree if the grammar is lexicalized, and as a result, there are many parallels between a dependency analysis and a TAG analysis if we abstract from the phrase structure of the latter (Rambow and Joshi, 1997). In fact, a dependency parse can be seen as a direct parse of a TAG derivation tree. And furthermore, we can derive a grammar in our lexicalized GDG formalism from a TAG grammar in a relatively straightforward manner. This has advantages in grammar reuse. Given the close relation between dependency and TAG, our approach can be seen as an exercise in applying the FSM-based parsing techniques of Evans and Weir (1997) and (1998) to the Tree Insertion Grammars of Schabes and Waters (1995), though the different origin of the approach (dependency parsing vs. TAG parsing) translates to differences in the way the FSMs are used to parse.

In this overview, we discuss related work in formalizing dependency in Section 2. We introduce GDG and its parsing algorithm in Section 3, and describe how we compile a TAG to a GDG in Section 4.

1. We would like to thank David Chiang, Anoop Sarkar, and three insightfully critical anonymous reviewers for their comments, explanations, and suggestions.
2. We leave open whether this point also has cognitive relevance.
3. The best-known (and perhaps only) application that requires phrase structure as the output of a parser is the parser evaluation methodology and software PARSEVAL (E. Black et al., 1991).
4. In this we follow a suggestion made by Abney (1996), essentially extending the formalism of (Gaifman, 1965).

© 2002 Alexis Nasr, Owen Rambow, John Chen, Srinivas Bangalore. *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*, pp. 100–108. Universitá di Venezia.

2. Related Work in Formalizing Dependency

There have been (at least) four types of formalizations of dependency grammars in the past. The first approach (for example, (Lombardo and Lesmo, 2000)) follows Gaifman (1965) in proposing traditional string rewriting rules, which however do not allow for an unbounded number of adjuncts. In the second approach, the dependency structure is constructed in reference to a parallel ("deeper") structure (Sgall et al., 1986; Mel'čuk, 1988). Because the rules make reference to other structures, these approaches cannot be formalized in a straightforward manner as context-free rewriting formalisms. In the third approach, which includes formalizations of dependency structure such as Link Grammar (Sleator and Temperley, 1993) or the tree-composition approach of Nasr (1996), rules construct the dependency tree incrementally; in these approaches, the grammar licenses dependency relations which, in a derivation, are added to the tree one by one, or in groups. In contrast, we are interested in a string-rewriting system; in such a system, we cannot add dependency relations incrementally: all daughters of a node must be added at once to represent a single rewrite step. In the fourth approach, the dependency grammar is converted into a headed context-free grammar (Abney, 1996; Holan et al., 1998). This approach allows for the recovery of the dependency structure both from the derivation tree and from a parse forest represented in polynomial space. (In fact, our parsing algorithm draws on this work.) However, the approach of course requires the introduction of additional nonterminal nodes. Finally, we observe that Recursive Transition Networks (Woods, 1970) can be used to encode a grammar whose derivation trees are dependency trees. However, they are more a general framework for encoding grammars than a specific type of grammar (for example, we can also use them to encode CFGs). In a somewhat related manner, Alshawi et al. (2000) use cascaded head automata to derive dependnecy trees, but leave the nature of the cascading under-formalized. Eisner (2000) provides a formalization of a system that uses two different automata to generate left and right children of a head. His formalism is very close to the one we present, but it is not a string-rewriting formalism (and not really generative at all). We are looking for a precise formulation of a generative dependency grammar, and the question has remained open whether there is an alternate formalism which allows for an unbounded number of adjuncts, introduces all daughter nodes at once in a string-rewriting step, and avoids the introduction of additional nonterminal nodes.

3. GDG and Parsing GDG

An Extended Context-Free Grammar (or ECFG for short) is like a CFG, except that the right-hand side is a regular expression over the terminal and nonterminal symbols of the grammar.[5] At each step in a derivation, we first choose a rewrite rule (as we do in CFG), and then we choose a string which is in the language denoted by the regular expression associated with the rule. This string is then treated like the right-hand side of a CFG rewrite rule. A Generative Dependency Grammar (GDG) is a lexicalized ECFG. For our formalism, this means that the regular expression in a production is such that each string in its denoted language contains at least one terminal symbol. When we use a GDG for linguistic description, its left-hand side nonterminal will be interpreted as the lexical category of the lexical item and will represent its maximal projection.

The parsing algorithm is a simple extension of the CKY algorithm for CFG. The difference is in the use of finite state machines in the items in the chart to represent the right-hand sides of the rules of the ECFG. A rule with category C as its left-hand side will give rise to a finite state machine which we call a **C-FSM**; its final states mark the completed recognition of a constituent of label C. Let T be the parse table for input sentence W and GDG G such that $T_{i,j}$ contains (M, q) iff M is a C-FSM, q is one of the final states of M, and we have a derivation C of substring $w_i \cdots w_j$ from C in G.

Initialization: We start by adding, for each i, $1 \leq i \leq n$, w_i to $T_{i,i}$.

Completion: If $T_{i,j}$ contains either the input symbol w or an item (M, q) such that q is a final state of M, and M is a C-FSM, then add to $T_{i,j}$ all (M', q') such that M' is a rule-FSM which transitions from a start state to state q' on input w or C.

Scanning: If (M_1, q_1) is in $T_{i,k}$, and $T_{k+1,j}$ contains either the input symbol w or the item (M_2, q_2) where q_2 is a final state and M_2 is a C-FSM, then add (M_1, q) to $T_{i,j}$ (if not already present) if M_1 transitions from q_1 to q on either w or C.

5. ECFG has been around informally since the sixties (e.g., the Backus-Naur form); for formalizations see (Madsen and Kristensen, 1976) or Albert et al. (1999).

At the end of the parsing process, a packed parse forest has been built. The nonterminal nodes are labeled with pairs (M, q) where M is an rule FSM and q a state of this FSM.

Obtaining the dependency trees from the packed parse forest is performed in two stages. In a first stage, a forest of binary syntagmatic trees is obtained from the packed forest and in a second stage, each syntagmatic tree is transformed into a dependency tree. We shall not give the details of these processes.

4. Compilation of a TAG into GDG

In fact, we do not derive a formal GDG from a TAG but instead directly compile a set of FSMs which are used by the parser (though we consider the distinction irrelevant, as FSMs and regular expressions are easily inter-convertible). To derive a set of FSMs from a TAG, we do a depth-first traversal of each elementary tree in the grammar (but excluding the root and foot nodes of adjunct auxiliary trees) to obtain a sequence of nonterminal nodes. Each node becomes two states of the FSM, one state representing the node on the downward traversal on the left side (the **left node state**), the other representing the state on the upward traversal, on the right side (the **right node state**). For leaf nodes, the two states are juxtaposed. The states are linearly connected with ϵ-transitions, with the left node state of the root node the start state, and its right node state the final state (except for predicative auxiliary trees – see below). To each non-leaf state, we add one self loop transition for each tree in the grammar that can adjoin at that state from the specified direction (i.e., for a state representing a node on the downward traversal, the auxiliary tree must adjoin from the left), labeled with the tree name. For each pair of adjacent states representing a substitution node, we add transitions between them labeled with the names of the trees that can substitute there. For the lexical head, we add a transition on that head. For footnodes of predicative auxiliary trees, we take the left node state as the final state. There are no other types of leaf nodes since we do not traverse the passive valency structure of adjunct auxiliary tees. Note that the treatment of footnodes makes it impossible to deal with trees that have terminal, substitution or active adjunction nodes on both sides of a footnode. It is this situation (iterated, of course) that makes TAG formally more powerful than CFG; in linguistic uses, it is very rare.[6] The result of this phase of the conversion is a set of FSMs, one per elementary tree of the grammar, whose transitions refer to other FSMs. We give a sample grammar and the result of converting it to FSMs in Figure 1.

As mentioned above, this approach is very similar to that of Evans and Weir (1997). One important difference is that they use FSMs to encode the seuqence of actions that need to be taken in a standard TAG parse, while we encode the active valency of the lexical item. As a result, our FSMs can readily represent the fact that we have seen a noun phrase and an adverb, and hypothesize that they are dependents of a particular verb found in the sentence, without having seen that verb yet (other than in the preprocessing stage that sleected the lexicalized subgrammar). In some sense, in our approach an Early-style prediction is built-in.

****discussion of TIG**** Chiang (2000) also parses with an automatically extracted TIG, but unlike our approach, he uses standard TAG/TIG parsing techniques.

To investigate the adequacy of a context-free parser for English, as well as the speed of the parser, we use an automatically extracted grammar called "Bob" (Chen, 2001). Bob has been extracted from Sections 02-21 of the Penn Treebank. Parameters of extraction have been set so that the tree frames of the resulting grammar have a "moderate" domain of locality, and preserve many but not all of the empty elements that are found in the Penn Treebank. Bob consists of 4909 tree frames. We tested our parser with Bob on 1,814 sentences of Section 00 of the PTB with an average sentence length of 21.2 words (excluding pure punctuation, i.e., punctuation which does not play a syntactic role such as conjunction or aposition). We evaluate performance using **accuracy**, the ratio of the number of dependency arcs which are correctly found (same head and daughter nodes) in the best parse for each sentence to the number of arcs in the entire test corpus. We also report the percentage of sentences for which we find the correct analysis (along with many others, of course).

To show that GDG is adequate for parsing English, we use the correct supertag associated with each input word and evaluate the performance of the parser. We expect only those sentences which do not have a context-free analysis not to have any analysis at all. This is indeed the case: there is no case of non-projectivity in the test corpus. (Note that since we analyze matrix verbs as depending on their embedded verb, followingthe TAG analysis, long-distance *wh*-movement is not in fact non-projective or us. However, the period associated with the

6. Our construction cannot handle Dutch cross-serial dependencies (not surprisingly), but it can convert the TAG analysis of *wh*-movement in English and similar languages.

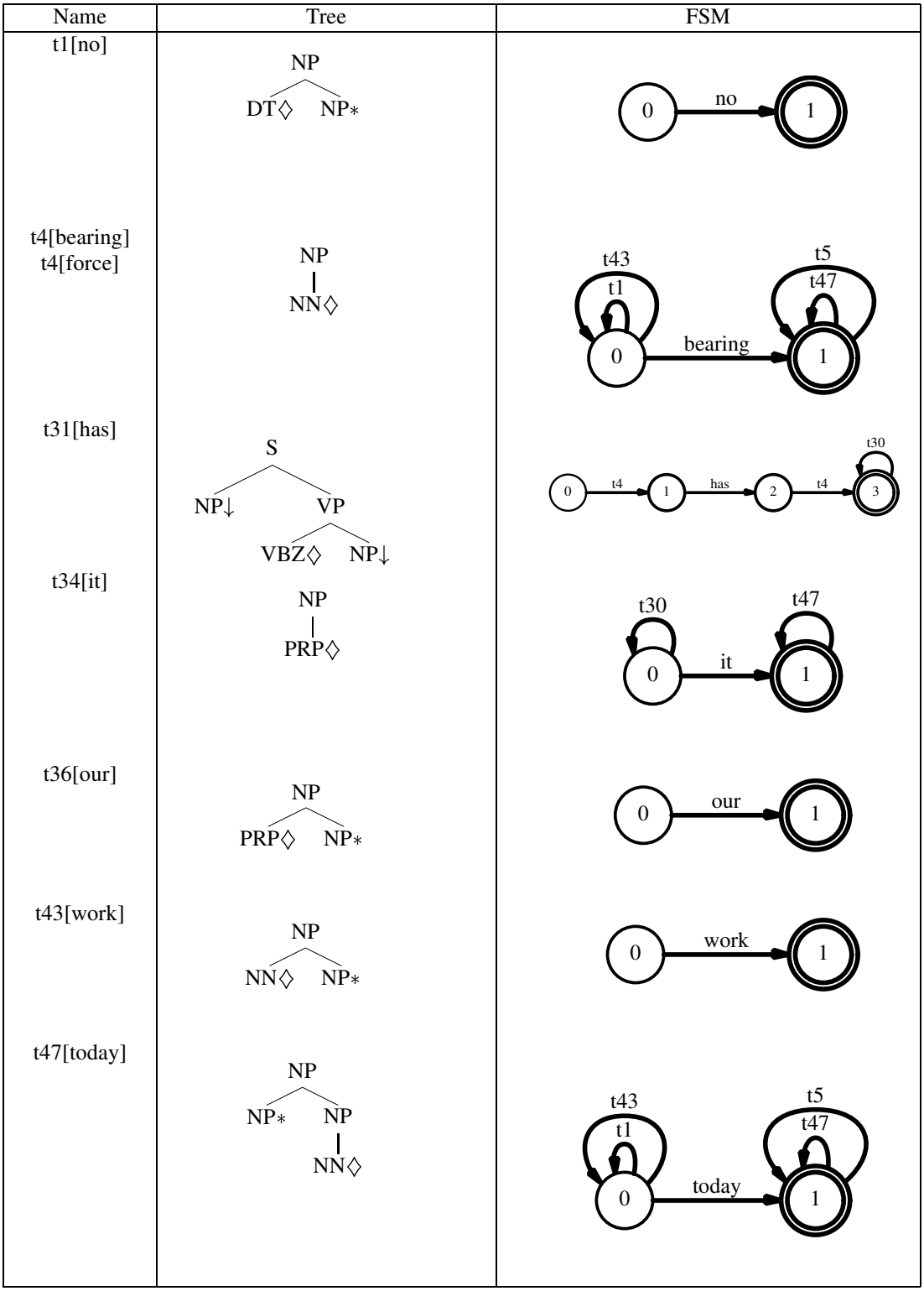

Figure 1: Bob subgrammar for sentence *It has no bearing on our workforce today* and set of FSMs that corresponds to that subgrammar

matrix verb does cause non-projectivity, but since we are disregarding true punctuation, this does not affect our result.

The average run-time for each sentence is 56ms, which to our knowledge is significantly quicker than existing full-fledged TAG parsers.[7] We show the dependence of parse time on sentence length in Figure 2. As is known both theoretically and empirically (Sarkar, 1998), lexical ambiguity has a drastic effect on parse time. We have not yet run experiments to show the relation (and, as mentioned above, this is where we hope to profit from using FSMs).

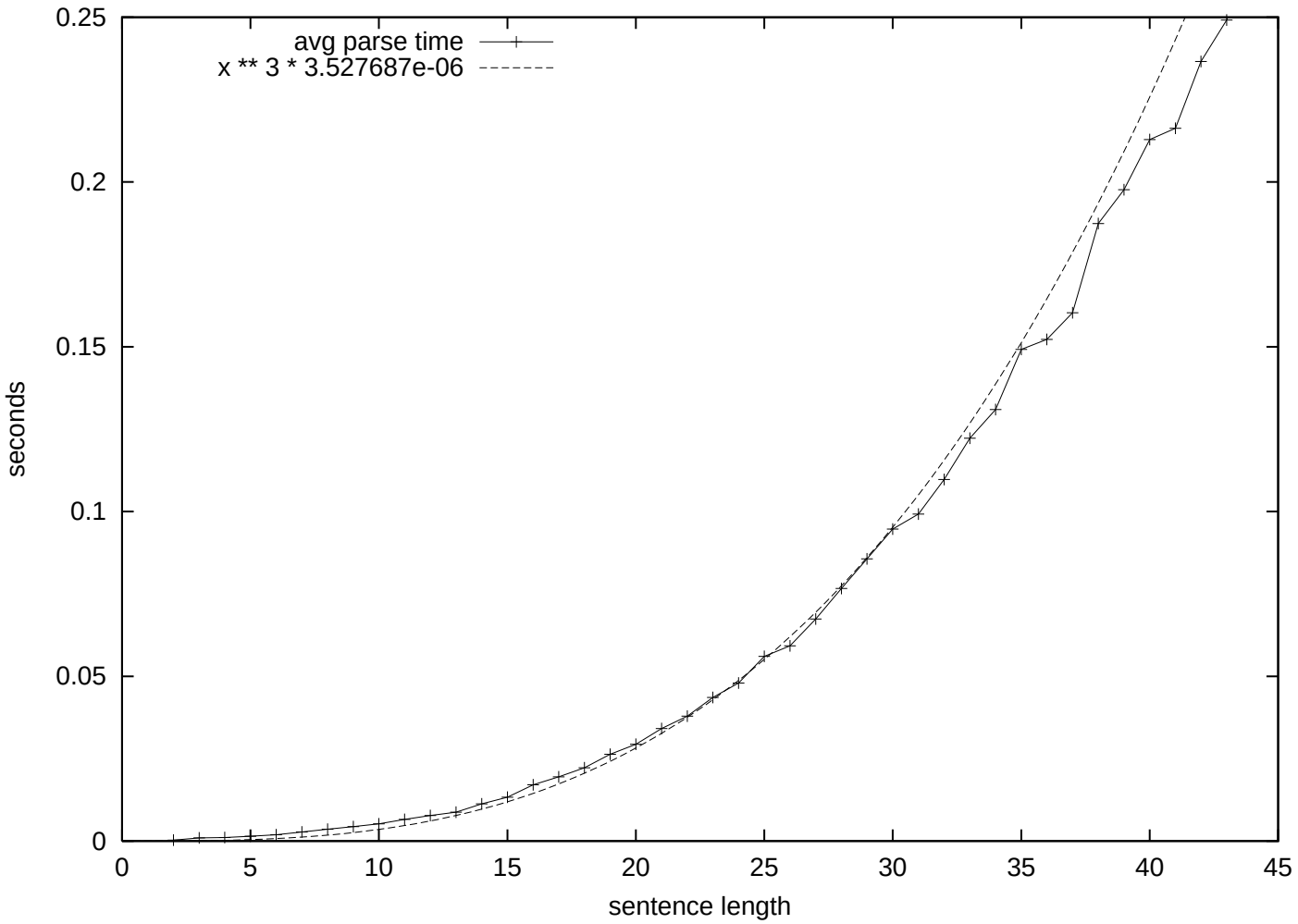

Figure 2: Average analysis time (in ms) against sentence length

5. Whither TAG Parsing?

The recent development of parsers using bilexical stochastic models which are derived from phrase-structrure corpora with the aid of "head percolation tables" (following (Magerman, 1995; Collins, 1997)).[8] The use of the same tool (head percolation tables), often the same tables, in automatically extracting TAG grammars raises the

7. Note that the extacted grammar does not have any features, so no feature unification is performed during parsing. Agreement phenomena can be enforced by using extended label sets, at the cost of increasing the size of the grammar. Of course, features in TAG are always bounded in size anyway, and hence always equivalent to an extended label set.
8. Of course, this work can also be interpreted as a rediscovery of the core insight of TAG.

question whether a stochastic TAG parser or TIG parser (such as (Chiang, 2000)) using bilexical stochastic models can ever outperform the non-TAG based approaches using head percolation (David Chiang, personal communication). We suggest that the interest in pursuing parsing using lexicalized grammar formalisms suchas TAG, TIG, or GDG lies not in a simple replication of the results obtained previouslyusing head percolation. Rather, the lexicalized formalisms allow for a different approach, which is not possible with the phrase structure- and head percolation-based approaches: namely gathering as much information as possible about the syntactic properties of the words of the input sentence using very fast algorithms. The best-known such an approach is supertagging (Bangalore and Joshi, 1999). While supertagging has been explored in the past (Bangalore and Joshi, 1999; Chen, 2001), we are not aware of a systematic investigation into the relation between the quality of supertagging, the use of n-best supertagging, pasre quality, and parse time. We intend to perform such investigations usingthe framework we have developed, and hope that the use of an explicit lexicalized grammar formalism can be shown to be useful precisely because it allows us to use other algorithms which the non-lexicalzied formalisms do not support.

References

Abney, Steven (1996). A grammar of projections. Unpublished manuscript, Universität Tübingen.

Albert, Jürgen; Guammarresi, Dora; and Wood, Derick (1999). Extended context-free grammars and normal form algorithms. In Champarnaud, Jean-Marc; Maurel, Denis; and Ziadi, Djelloul, editors, *Automata Implementations: Third International Workshop on Implementing Automata (WIA'98)*, volume 1660 of *LNCS*, pages 1–12. Springer Verlag.

Alshawi, Hiyan; Bangalore, Srinivas; and Douglas, Shona (2000). Learning dependency translation models as collections of finite-state head transducers. *cl*, 26(1):45–60.

Bangalore, Srinivas and Joshi, Aravind (1999). Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–266.

Chen, John (2001). *Towards Efficient Statistical Parsing Using Lexicalized Grammatical Information*. PhD thesis, University of Delaware.

Chiang, David (2000). Statistical parsing with an automatically-extracted tree adjoining grammar. In *38th Meeting of the Association for Computational Linguistics (ACL'00)*, pages 456–463, Hong Kong, China.

Collins, Michael (1997). Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, Madrid, Spain.

E. Black et al. (1991). A procedure for quantitatively comparing the syntactic coverage of english grammars. In *Proceedings of the February 1991 DARPA Speech and Natural Language Workshop*. DARPA.

Eisner, Jason (2000). Bilexical grammars and their cubic-time parsing algorithms. In Bunt, Harry C. and Nijholt, Anton, editors, *New Developments in Natural Language Parsing*. Kluwer Academic Publishers.

Evans, Roger and Weir, David (1997). ***check title***a. In *5th International Workshop on Parsing Technologies*.

Evans, Roger and Weir, David (1998). **check title**. In *36th Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING-ACL'98)*, Montréal, Canada.

Gaifman, Haim (1965). Dependency systems and phrase-structure systems. *Information and Control*, 8:304–337.

Holan, Tomáš; Kuboň, Vladislav; Oliva, Karel; and Plátek, Martin (1998). Two useful measures of word order complexity. In Kahane, Sylvain and Polguère, Alain, editors, *Processing of Dependency Grammars: Proceeding of the Workshop*, pages 21–28, Montréal, Canada. ACL/COLING.

Kahane, Sylvain; Nasr, Alexis; and Rambow, Owen (1998). Pseudo-projectivity: A polynomially parsable non-projective dependency grammar. In *36th Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING-ACL'98)*, pages 646–652, Montréal, Canada.

Lombardo, Vincenzo and Lesmo, Leonardo (2000). A formal theory of dependency syntax with empty units. *Traitement automatque des langues*, 41(1):179–209.

Madsen, O.L. and Kristensen, B.B. (1976). *LR*-parsing of extended context-free grammars. *Acta Informatica*, 7:61–73.

Magerman, David (1995). Statistical decision-tree models for parsing. In *33rd Meeting of the Association for Computational Linguistics (ACL'95)*.

Mel'čuk, Igor A. (1988). *Dependency Syntax: Theory and Practice*. State University of New York Press, New York.

Nasr, Alexis (1996). *Un système de reformulation automatique de phrases fondé sur la Théorie Sens-Texte : application aux langues contrôlées*. PhD thesis, Université Paris 7.

Rambow, Owen and Joshi, Aravind (1997). A formal look at dependency grammars and phrase-structure grammars, with special consideration of word-order phenomena. In Wanner, Leo, editor, *Recent Trends in Meaning-Text Theory*, pages 167–190. John Benjamins, Amsterdam and Philadelphia.

Sarkar, Anoop (1998). Conditions on Consistency of Probabilistic Tree Adjoining Grammars. In *36th Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING-ACL'98)*, pages 1164–1170, Montreal.

Schabes, Yves and Waters, Richard C. (1995). Tree Insertion Grammar: A cubic-time, parsable formalism that lexicalizes Context-Free Grammar without changing the trees produced. *Computational Linguistics*, 21(4):479–514.

Sgall, P.; Hajicova, E.; and Panevova, J. (1986). *The meaning of the sentence and its semantic and pragmatic aspects*. Reidel, Dordrecht.

Sleator, Daniel and Temperley, Davy (1993). Parsing english with a link grammar,. In *Proceedings of the Third International Workshop on Parsing Technologies IIWPT'93)*.

Woods, William A. (1970). Transition network grammars for natural language analysis. *Commun. ACM*, 3(10):591–606.