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 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 threefold.

- 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 general 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.
- 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 commercial 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 properties, 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.[3] We call this formalism GDG,or Generative Dependency 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 (though not always the linguistically most plausible one), 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 (TIG) 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 introduce GDG and its parsing algorithm in Section 2, and describe how we compile a TAG to a GDG in Section 3. We discuss the empirical adequacy and speed of the parser in Section 4, and conclude with a discussion of how our projected work relates to other current work in parsing. We postpone a discussion of related work in formalizing dependency to another publication.

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. In this we follow a suggestion made by Abney (1996), essentially extending the formalism of (Gaifman, 1965).

 Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6), pp. 100–105. Universitá di Venezia.

2. 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.[4] At each step in a derivation, we first choose a rewrite rule (as we do in CFG), and then we choose a string of terminal and nonterminal symbols 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 right-hand side symbols represent the active valency of the lexical head of the rule, i.e., the categories that must or may (depending on whether the symbol is optional) be dependents on this lexical head for it to be the root of a well-formed subtree. Passive valency (a representation of where and to what other heads a lexical head may attach itself) is not explicitly encoded.

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.

Note that because we are using a dependency grammar (or, in TAG terms, parsing the derivation tree directly), each scanning step corresponds to one attachment of a lexical head to another (or, in TAG terms, to an adjunction or a substitution). 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.

3. 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 interconvertible). 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 immediately follow one another. 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

4. 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).

auxiliary trees which are left auxiliary trees (in the sense of Schabes and Waters (1995), i.e., all nonempty frontier nodes are to the left of the footnode), 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.[5] 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.

The construction treats a TAG as if were a TIG (or, put differently, it coerces a TAG to be a TIG): during the traversal, both terminal nodes and nonterminal (i.e., substitution) nodes between the footnode and the root node are ignored (because the traversal stops at the footnode), thus imposing the constraint that the trees may not be wrapping trees and that no further adjunction may occur to the right of the spine in a left auxiliary tree. Furthermore, by modeling adjunction as a loop transition, we adopt the definition of adjunction of Schabes and Shieber (1994), as does TIG. Chiang (2000) also parses with an automatically extracted TIG, but unlike our approach, he uses standard TAG/TIG parsing techniques. Rogers (1994) proposes a different context-free variant, "regular-form TAG". The set of regular-form TAGs is a superset of the set of TIGs, and our construction cannot capture the added expressive power of regular-form TAG.

As mentioned above, this approach is very similar to that of Evans and Weir (1997). One important difference is that they model TAG, while we model TIG. Another difference is that they use FSMs to encode the sequence of actions that need to be taken during a standard TAG parse (i.e., of the derived tree), while we encode the active valency of the lexical head in the FSM. A result, in retrieving the derivation tree, each item in the parse tree corresponds to an attachment of one word to another, and there are fewer items. Furthermore, our FSMs are built left-to-right, while Evans and Weir only explore FSMs constructed bottom-up from the lexical anchor of the tree. As a result, we can perform a strict left-to-right parse, which is not straightforwardly possible in TAG parsing using FSMs.

4. Adequacy of GDG and Initial Run-Time Results

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 (typically those cases where empty elements are found in the XTAG grammar, such as PRO subjects). 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 apposition). 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 (an empirical question), 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, following the TAG analysis, long-distance *wh*-movement is not in fact non-projective or us. However, the punctuation mark associated with the 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 (parsing using the correct supertag for each word, and no other supertag), which to our knowledge is significantly quicker than existing full-fledged TAG parsers.[6] We show the

5. 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.

6. Note that the extracted 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. (This is a parameter in the extraction algorithm.) Of course, features in TAG are always bounded in size anyway, and hence always equivalent to an extended label set.

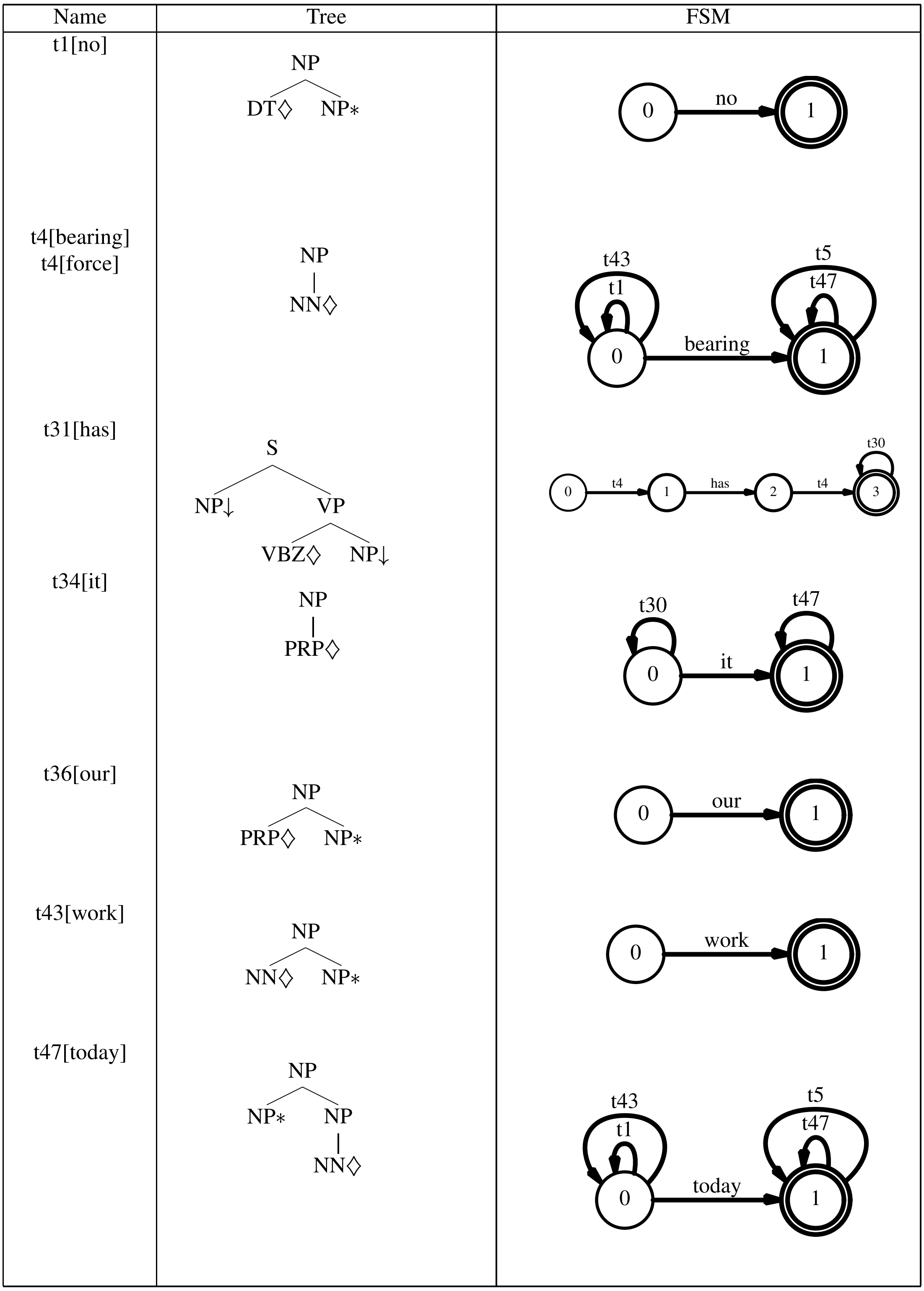

Figure 1: Bob subgrammar for sentence *It has no bearing on our workforce today* and set of FSMs that corresponds to that subgrammar

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).

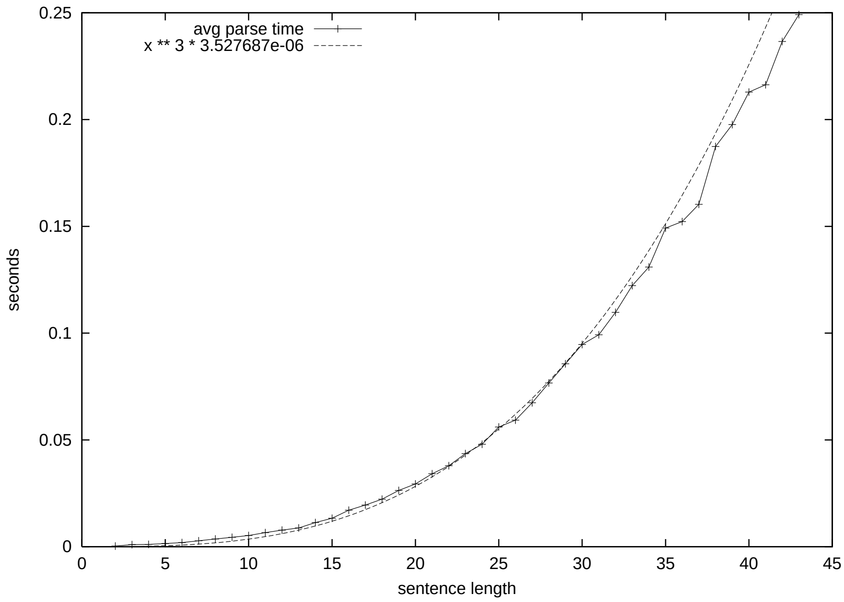

Figure 2: Average analysis time (in ms) against sentence length

5. Parsing Using Lexicalized Grammars

Recently, parsers using bilexical stochastic models which are derived from phrase-structure corpora with the aid of “head percolation tables” have been successfully developed (Magerman, 1995; Collins, 1997)). In some sense, these approaches use an unlexicalized formalism (CFG) and implicitly lexicalize it.[7] The use of the same tool (head percolation tables), often the same tables, in automatically extracting TAG grammars raises the question whether a stochastic TAG parser or TIG parser (such as (Chiang, 2000)) using bilexical stochastic models can ever outperform the implicitly lexicalized approaches, since they use the same kind of data, though packaged differently (David Chiang, personal communication). We suggest that the interest in pursuing parsing using explicitly lexicalized grammar formalisms such as TAG, TIG, or GDG lies not in a simple replication of the results obtained previously using head percolation. Rather, the lexicalized formalisms allow for a different approach, which is not possible with the implicitly lexicalized approaches: namely gathering as much syntactic information as possible about the syntactic properties of the words of the input sentence using very fast algorithms prior to (or interleaved with, but separate from) parsing.[8] The best-known such 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, parse quality, and parse time (but see initial results in Chen et al. (2002)). We intend to perform such investigations using the 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 disambiguation techniques which the implicitly lexicalized formalisms do not support.

7. Of course, this work can also be interpreted as a rediscovery of the core insight of TAG.
8. Some parsers make use of prior part-of-speech parsing, which provides some very shallow syntactic information.

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.

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.

Chen, John; Bangalore, Srinivas; Collins, Michael; and Rambow, Owen (2002). Reranking an n-gram SuperTagger. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+6)*, Venice, Italy.

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.

Evans, Roger and Weir, David (1997). Automaton-based parsing for lexicalized grammars. In *5th International Workshop on Parsing Technologies (IWPT97)*, pages 66–76.

Evans, Roger and Weir, David (1998). A structure-sharing parser for lexicalized grammars. In *36th Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING-ACL'98)*, pages 372–378, Montréal, Canada.

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

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.

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)*.

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.

Rogers, James (1994). Capturing cfls with Tree Adjoining Grammars. In *32nd Meeting of the Association for Computational Linguistics (ACL'94)*. ACL.

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 Shieber, Stuart (1994). An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 1(20):91–124.

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.