Lexicalized Grammar 101

Matthew Stone
Department of Computer Science and Center for Cognitive Science
Rutgers, the State University of New Jersey
Piscataway NJ 08854-8019 USA
http://www.cs.rutgers.edu/~mdstone
mdstone@cs.rutgers.edu

Abstract

This paper presents a simple and versatile tree-rewriting lexicalized grammar formalism, TAGLET, that provides an effective scaffold for introducing advanced topics in a survey course on natural language processing (NLP). Students who implement a strong competence TAGLET parser and generator simultaneously get experience with central computer science ideas and develop an effective starting point for their own subsequent projects in data-intensive and interactive NLP.

1 Introduction

This paper is particularly addressed to readers at institutions whose resources and organization rule out extensive formal course-work in natural language processing (NLP). This is typical at universities in North America. In such places, NLP teaching must be ambitious but focused; courses must quickly acquaint a broad range of students to the essential concepts of the field and sell them on its current research opportunities and challenges. This paper presents one resource that may help. Specifically, I outline a simple and versatile lexicalized formalism for natural language syntax, semantics and pragmatics, called TAGLET, and draw on my experience with CS 533 (NLP) at Rutgers to motivate the potential role for TAGLET in a broad NLP class whose emphasis is to introduce topics of current research. Notes, assignments and implementations for TAGLET are available on the web.

I begin in Section 2 by describing CS 533—situating the course within the university and outlining its topics, audience and goals. I then describe the specific goals for teaching and implementing grammar formalisms within such a course, in Section 3. Section 4 gives an informal overview of TAGLET, and the algorithms, specifications and assignments that fit TAGLET into a broad general NLP class.

In brief, TAGLET is a context-free tree-rewriting formalism, defined by the usual complementation operation and the simplest imaginable modification operation. By implementing a strong competence TAGLET parser and generator students simultaneously get experience with central computer science ideas—data structures, unification, recursion and abstraction—and develop an effective starting point for their own subsequent projects. Two noteworthy directions are the construction of interactive applications, where TAGLET's relatively scalable and reversible processing lets students easily explore cutting-edge issues in dialogue semantics and pragmatics, and the development of linguistic specifications, where TAGLET's ability to lexicalize tree-bank parses introduces a modern perspective of linguistic intuitions and annotations as programs. Section 5 briefly summarizes the advantages of TAGLET over the many alternative formalisms that are available; an appendix to the paper provides more extensive technical details.

2 CS 533

NLP at Rutgers is taught as part of the graduate artificial intelligence (AI) sequence in the computer science department. As a prerequisite, computer science students are expected to be familiar with prob-

abilistic and decision-theoretic modeling (including statistical classification, hidden Markov models and Markov decision processes) from the graduate-level AI foundations class. They might take NLP as a preliminary to research in dialogue systems or in learning for language and information—or simply to fulfill the breadth requirement of MS and PhD degrees.

Students from a number of other departments frequently get involved in natural language research, however, and are also welcome in 533; on average, only about half the students in 533 come from computer science. Students from the linguistics department frequently undertake computational work as a way of exploring practical learnability as a constraint on universal grammar, or practical reasoning as a constraint on formal semantics and pragmatics. The course also attracts students from Rutgers's library and information science department, its primary locus for research in information retrieval and human-computer interaction. Ambitious undergraduates can also take 533 their senior year; most participate in the interdisciplinary cognitive science undergraduate major. 533 is the only computational course in natural language at Rutgers.

Overall, the course is structured into three modules, each of which represents about fifteen hours of in-class lecture time.

The first module gives a general overview of language use and dialogue applications. Lectures follow (Clark, 1996), but instill the practical methodology for specifying and constructing knowledge-based systems, in the style of (Brachman et al., 1990), into the treatment of communication. Concurrently, students explore precise descriptions of their intuitions about language and communication through a series of short homework exercises.

The second module focuses on general techniques for linguistic representation and implementation, using TAGLET. With an extended TAGLET project, conveniently implemented in stages, we use basic tree operations to introduce Prolog programming, including data structures, recursion and abstraction much as outlined in (Sterling and Shapiro, 1994); then we write a simple chart parser with incremental interpretation, and a simple communicative-intent generator scaled down after (Stone et al., 2001).

The third module explores the distinctive problems of specific applications in NLP, including spoken dialogue systems, information retrieval and text classification, spelling correction and shallow tagging applications, and machine translation. Jurafsky and Martin (2000) is our source-book. Concurrently, students pursue a final project, singly or in cross-disciplinary teams, involving a more substantial and potentially innovative implementation.

In its overall structure, the course seems quite successful. The initial emphasis on clarifying intuitions about communication puts students on an even footing, as it highlights important ideas about language use without too much dependence on specialized training in language or computation. By the end of the class, students are able to build on the more specifically computational material to come up with substantial and interesting final projects. In Spring 2002 (the first time this version of 533 was taught), some students looked at utterance interpretation, response generation and graphics generation in dialogue interaction; explored statistical methods for word-sense disambiguation, summarization and generation; and quantified the potential impact of NLP techniques on information tasks. Many of these results represented fruitful collaborations between students from different departments.

Naturally, there is always room for improvement, and the course is evolving. My presentation of TAGLET here, for example, represents as much a project for the next run of 533 as a report of this year's materials; in many respects, TAGLET actually emerged during the semester as a dynamic reaction to the requirements and opportunities of a six-week module on general techniques for linguistic representation and implementation.

3 Language and Computation in NLP

In a survey course for a broad, research-oriented audience, like CS 533 at Rutgers, a module on linguistic representation must orient itself to central ideas about computation. 533 may be the first and last place linguistics or information science students encounter concepts of specification, abstraction, complexity and search in class-work. The students who attack interdisciplinary research with success will be the ones who internalize and draw on these concepts, not those who merely hack proficiently. At the same time, computer scientists also can benefit from an

emphasis on computational fundamentals; it means that they are building on and reinforcing their expertise in computation in exploring its application to language. Nevertheless, NLP is not compiler construction. Programming assignments should always underline a worthwhile linguistic lesson, not indulge in implementation for its own sake.

This perspective suggests a number of desiderata for the grammar formalism for a survey course in NLP.

Tree rewriting. Students need to master recursive data-structures and programming. NLP directs our attention to the recursive structures of linguistic syntax. In fact, by adopting a grammar formalism whose primitives operate on these structures as first-class objects, we can introduce a rich set of relatively straightforward operations to implement, and motivate them by their role in subsequent programs.

Lexicalization. Students need to distinguish between specification and implementation, and to understand the barriers of abstraction that underlie the distinction. Lexicalized grammars come with a ready notion of abstraction. From the outside, abstractly, a lexicalized grammar analyzes each sentence as a simple combination of atomic elements from a lexicon of options. Simultaneously, a concrete implementation can assign complex structures to the atomic elements (elementary trees) and implement complex combinatory operations.

Strong competence implementation. Students need to understand how natural language must and does respond to the practical logic of physical realization, like all AI (Agre, 1997). Mechanisms that use grammars face inherent computational problems and natural grammars in particular must respond to these problems: students should undertake implementations which directly realize the operations of the grammar in parsing and generation. But these must be effective programs that students can build on—our time and interest is too scarce for extensive reimplementations.

Simplicity. Where possible, linguistic proposals should translate readily to the formalism. At the same time, students should be able to adapt aspects of the formalism to explore their own judgments and ideas. Where possible, students should get intuitive and satisfying results from straightforward algorithms implemented with minimal bookkeeping and case analysis. At the same time, there is no reason why the formalism should not offer opportunities for meaningful optimization.

We cannot expect any formalism to fare perfectly by all these criteria—if any does, it is a deep fact about natural language! Still, it is worth remarking just how badly these criteria judge traditional unification-based context-free grammars (CFGs), as presented in say (Pereira and Shieber, 1987). Data-structures are an afterthought in CFGs; CFGs cannot in principle be lexicalized; and, whatever their merits in parsing or recognition, CFGs set up a positively abysmal search space for meaningful generation tasks.

4 TAGLET

TAGLET[1] is my response to the objectives motivated in Section 2 and outlined in Section 3. TAGLET represents my way of distilling the essential linguistic and computational insights of lexicalized tree-adjoining grammar—LTAG (Joshi et al., 1975; Schabes, 1990)—into a form that students can easily realize in end-to-end implementations.

4.1 Overview

Like LTAG, TAGLET analyzes sentences as a complex of atomic elements combined by two kinds of operations, *complementation* and *modification*. Abstractly, *complementation* combines a *head* with an *argument* which is syntactically obligatory and semantically dependent on the head. Abstractly, *modification* combines a head with an *adjunct* which is syntactically optional and need not involve any special semantic dependence. Crucially for generation, in a derivation, modification and complementation operations can apply to a head in any order, often yielding identical structures in surface syntax. This means the generator can provide required material first, then elaborate it, enabling use of grammar in high-level tasks such as the planning of referring expressions or the "aggregation" of related semantic material into a single complex sentence.

Concretely, TAGLET operations are implemented by operations that rewrite trees. Each lexical element is associated with a fragmentary phrase-

[1] If the acronym must stand for something, "Tree Assembly Grammar for LExicalized Teaching" will do.

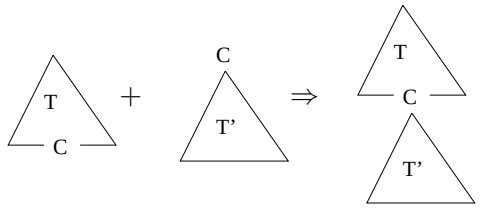


Figure 1: Substitution (complementation).

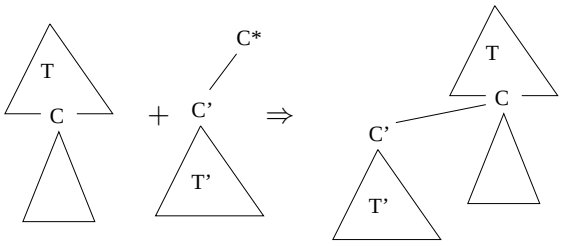


Figure 2: Forward sister-adjunction (modification.)

structure tree containing a distinguished word called the *anchor*. For complementation, TAGLET adopts TAG's *substitution* operation; substitution replaces a leaf node in the head tree with the phrase structure tree associated with the complement. See Figure 1. For modification, TAGLET adopts the the *sister-adjunction* operation defined in (Rambow et al., 1995); sister-adjunction just adds the modifier subtree as a child of an existing node in the head tree—either on the left of the head (*forward* sister-adjunction) as in Figure 2, or on the right of the head (*backward* sister-adjunction). I describe TAGLET formally in Appendix A.

TAGLET is equivalent in weak generative power to context-free grammar. That is, any language defined by a TAGLET also has a CFG, and any language defined by a CFG also has a TAGLET. On the other hand context-free languages can have derivations in which all lexical items are arbitrarily far from the root; TAGLET derived structures always have an anchor whose path to the root of the sentence has a fixed length given by a grammatical element. See Appendix B. The restriction seems of little linguistic significance, since any tree-bank parse induces a unique TAGLET grammar once you label which child of each node is the head, which are complements and which are modifiers. Indeed, since TAGLET thus induces bigram dependency structures from trees, this invites the estimation of probability distributions on TAGLET derivations based on

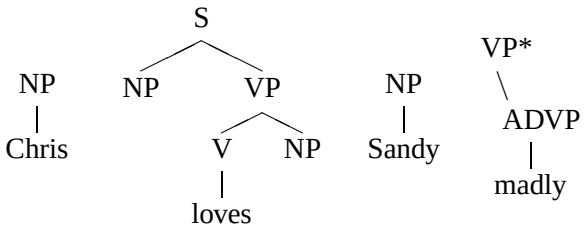


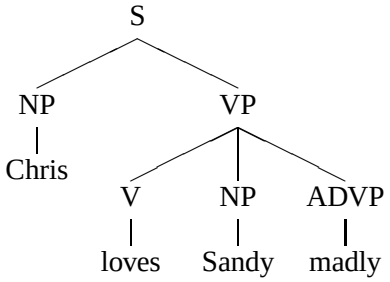
Figure 3: Parallel analysis in TAGLET and TAG.

observed bigram dependencies; see (Chiang, 2000).

To implement an effective TAGLET generator, you can perform a greedy head-first search of derivations guided by heuristic progress toward achieving communicative goals (Stone et al., 2001). Meanwhile, because TAGLET is context-free, you can easily write a CKY-style dynamic programming parser that stores structures recognized for spans of text in a chart, and iteratively combines structures in adjacent spans until the analyses span the entire sentence. (More complexity would be required for multiply-anchored trees, as they induce discontinuous constituents.) The simple requirement that operations never apply inside complements or modifiers, and apply left-to-right within a head, suffices to avoid spurious ambiguity. See Appendix C.

4.2 Examples

With TAGLET, two kinds of examples are instructive: those where TAGLET can mirror TAG, and those where it cannot. For the first case, consider an analysis of *Chris loves Sandy madly* by the trees of Figure 3. The final structure is:



For the second case, consider the embedded question *who Chris thinks Sandy likes*. The usual TAG analysis uses the full power of adjunction. TAGLET requires the use of one of the familiar context-free filler-gap analyses, as perhaps that suggested by the trees in Figure 4, and their composition:

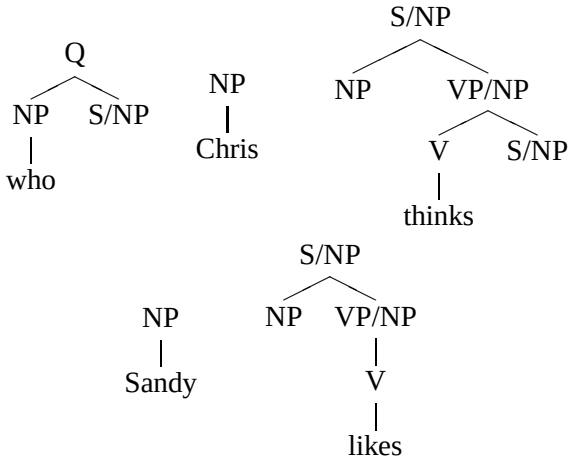
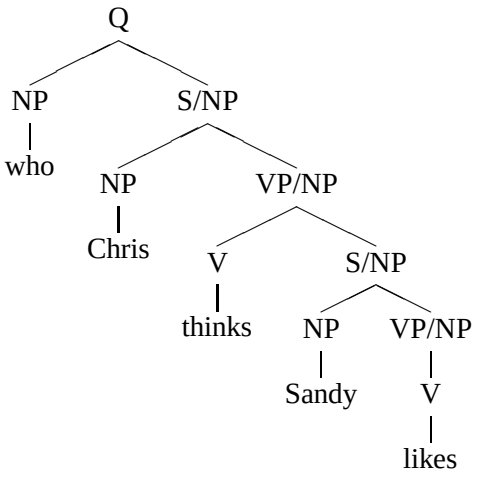
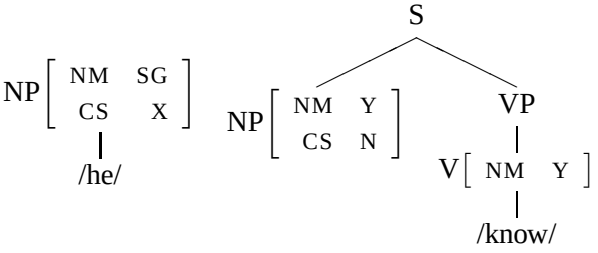


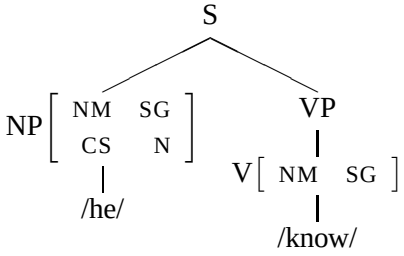
Figure 4: TAGLET requires a gap-threading analysis of extraction (or another context-free analysis).



The use of syntactic features amounts to an intermediate case. In TAGLET derivations (unlike in TAG) nodes accrete children during the course of a derivation but are never rewritten or split. Thus, we can decorate any TAGLET node with a single set of syntactic features that is preserved throughout the derivation. Consider the trees for *he knows* below:



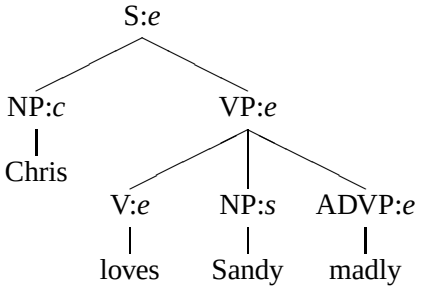
When these trees combine, we can immediately unify the number Y of the verb with the pronoun's singular; we can immediately unify the case X of the pronoun with the nominative assigned by the verb:



The feature values will be preserved by further steps of derivation.

4.3 Building on TAGLET

Semantics and pragmatics are crucial to NLP. TAGLET lets students explore meaty issues in semantics and pragmatics, using the unification-based semantics proposed in (Stone and Doran, 1997). We view constituents as referential, or better, indexical; we link elementary trees with constraints on these indices and conjoin the constraints in the meaning of a compound structure. This example shows how the strategy depends on a rich ontology:



$$chris(c) \wedge sandy(s) \wedge love(e,c,s) \wedge mad(e)$$

The example also shows how the strategy lets us quickly implement, say, the constraint-satisfaction approaches to reference resolution or the plan-recognition approaches to discourse integration described in (Stone and Webber, 1998).

4.4 Lectures and Assignments

Here is a plan for a six-week TAGLET module. The first two weeks introduce data structures and recursive programming in Prolog, with examples drawn from phrase structure trees and syntactic combination; and discuss dynamic-programming parsers, with an aside on convenient implementation using Prolog assertion. As homework, students implement simple tree operations, and build up to definitions of

substitution and modification for parsing and generation; they use these combinatory operations to write a CKY TAGLET parser.

The next two weeks begin with lectures on the lexicon, emphasizing abstraction on the computational side and the idiosyncrasy of lexical syntax and the indexicality of lexical semantics on the linguistic side; and continue with lectures on semantics and interpretation. Meanwhile, students add reference resolution to the parser, and implement routines to construct grammars from tree-bank parses.

The final two weeks cover generation as problem-solving, and search through the grammar. Students reuse the grammar and interpretation model they already have to construct a generator.

5 Conclusion

Important as they are, lexicalized grammars can be forbidding. Versions of TAG and combinatory categorial grammars (CCG) (Steedman, 2000), as presented in the literature, require complex bookkeeping for effective computation. When I wrote a CCG parser as an undergraduate, it took me a whole semester to get an implemented handle on the metatheory that governs the interaction of (crossing) composition or type-raising with spurious ambiguity; I still have never written a TAG parser or a CCG generator. Variants of TAG like TIG (Schabes and Waters, 1995) or D-Tree grammars (Rambow et al., 1995) are motivated by linguistic or formal considerations rather than pedagogical or computational ones. Other formalisms come with linguistic assumptions that are hard to manage. Link grammar (Sleator and Temperley, 1993) and other pure dependency formalisms can make it difficult to explore rich hierarchical syntax and the flexibility of modification; HPSG (Pollard and Sag, 1994) comes with a commitment to its complex, rather bewildering regime for formalizing linguistic information as feature structures. Of course, you probably could refine any of these theories to a simple core—and would get something very like TAGLET.

I strongly believe that this distillation is worth the trouble, because lexicalization ties grammar formalisms so closely to the motivations for studying language in the first place. For linguistics, this philosophy invites a fine-grained description of sentence syntax, in which researchers document the diversity of linguistic constructions within and across languages, and at the same time uncover important generalizations among them. For computation, this philosophy suggests a particularly concrete approach to language processing, in which the information a system maintains and the decisions it takes ultimately always just concern words. In taking TAGLET as a starting point for teaching implementation in NLP, I aim to expose a broad range of students to a lexicalized approach to the cognitive science of human language that respects and integrates both linguistic and computational advantages.

Acknowledgments

Thanks to the students of CS 533, particularly David DeVault, and four anonymous reviewers for helping to disabuse me of numerous preconceptions.

References

Philip E. Agre. 1997. *Computation and Human Experience*. Cambridge.

Ronald Brachman, Deborah McGuinness, Peter Patel Schneider, Lori Alperin Resnick, and Alexander Borgida. 1990. Living with CLASSIC: when and how to use a KL-ONE-like language. In J. Sowa, editor, *Principles of Semantic Networks*. Morgan Kaufmann.

David Chiang. 2000. Statistical parsing with an automatically-extracted tree adjoining grammar. In *ACL*, pages 456–463.

Herbert H. Clark. 1996. *Using Language*. Cambridge University Press, Cambridge, UK.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2000. *Introduction to automata theory, languages and computation*. Addison-Wesley, second edition.

Aravind K. Joshi, L. Levy, and M. Takahashi. 1975. Tree adjunct grammars. *Journal of the Computer and System Sciences*, 10:136–163.

Daniel Jurafsky and James H. Martin. 2000. *Speech and Language Processing: An introduction to natural language processing, computational linguistics and speech recognition*. Prentice-Hall.

Fernando C. N. Pereira and Stuart M. Shieber. 1987. *Prolog and Natural Language Analysis*. CSLI, Stanford CA.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago.

Owen Rambow, K. Vijay-Shanker, and David Weir. 1995. D-Tree grammars. In *ACL*, pages 151–158.

Yves Schabes and Richard C. Waters. 1995. Tree-insertion grammar: A cubic-time parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Computational Linguistics*, 21:479–513.

Yves Schabes. 1990. *Mathematical and Computational Aspects of Lexicalized Grammars*. Ph.D. thesis, Computer Science Department, University of Pennsylvania.

Daniel Sleator and Davy Temperley. 1993. Parsing English with a link grammar. In *Third International Workshop on Parsing Technologies*.

Mark Steedman. 2000. *The Syntactic Process*. MIT.

Leon Sterling and Ehud Shapiro. 1994. *The Art of Prolog*. MIT, second edition.

Matthew Stone and Christine Doran. 1997. Sentence planning as description using tree-adjoining grammar. In *Proceedings of ACL*, pages 198–205.

Matthew Stone and Bonnie Webber. 1998. Textual economy through close coupling of syntax and semantics. In *Proceedings of International Natural Language Generation Workshop*, pages 178–187.

Matthew Stone, Christine Doran, Bonnie Webber, Tonia Bleam, and Martha Palmer. 2001. Microplanning with communicative intentions: The SPUD system. *Under review*.

A Definitions

I define TAGLET in terms of *primitive trees*. The definitions require a set V_T of *terminal categories*, corresponding to our lexical items, and a disjoint set V_N of *nonterminal categories*, corresponding to constituent categories. TAGLET uses trees labeled by these categories both as representations of the syntactic structure of sentences and as representations of the grammatical properties of words:

- A *syntactic tree* is a tree whose nodes are each assigned a unique label in $V_N \cup V_T$, such that only leaf nodes are assigned a label in V_T.
- A *lexical tree* is a syntactic tree in which exactly one node, called the *anchor*, is assigned a label in V_T. The path through such a tree from the root to the anchor is called the *spine*.

A *primitive tree* is lexical tree in which every leaf is the child of a node on the spine. See Figures 3 and 4.

A TAGLET *element* is a pair $\langle T, O \rangle$ consisting of primitive tree together with the specification of the operation for the tree; the allowable operations are complementation, indicated by α; premodification at a specified category $C \in V_N$, indicated by $\beta^{\rightarrow}(C)$ and postmodification at a specified category $C \in V_N$, indicated by $\beta^{\leftarrow}(C)$.

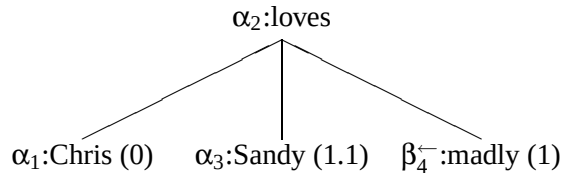
Formally, then, a TAGLET *grammar* is a tuple $G = \langle V_T, V_N, \Gamma \rangle$ where V_T gives the set of terminal categories, V_N gives the set of nonterminal categories, and Γ gives a set of TAGLET elements for V_T and V_N. Given a TAGLET grammar G, the set of *derived trees* for G is defined as the smallest set closed under the following operations:

- (Initial) Suppose $\langle T, O \rangle \in \Gamma$. Then $\langle T, O \rangle$ is a derived tree for G.
- (Substitution) Suppose $\langle T, O \rangle$ is a derived tree for G where T contains leaf node n with label $C \in V_N$; and suppose $\langle T', \alpha \rangle$ is a derived tree for G where the root of T' also has label C. Then $\langle T'', O \rangle$ is a derived tree for G where T'' is obtained from T by identifying node n with the root of T'.
- (Premodification) Suppose $\langle T, O \rangle$ is a derived tree for G where T contains node n with label $C \in V_N$, and suppose $\langle T', \beta^{\rightarrow}(C) \rangle$ is a derived tree for G. Then $\langle T'', O \rangle$ is a derived tree for G where T'' is obtained from T by adding T' as the first child of node n.
- (Postmodification) Suppose $\langle T, O \rangle$ is a derived tree for G where T contains node n with label $C \in V_N$, and suppose $\langle T', \beta^{\leftarrow}(C) \rangle$ is a derived tree for G. Then $\langle T'', O \rangle$ is a derived tree for G where T'' is obtained from T by adding T' as the last child of node n.

A *derivation* for G is a derived tree $\langle T, \alpha \rangle$ for G, in which all the leaves of T are elements of V_T. The *yield* of a derivation $\langle T, \alpha \rangle$ is the string consisting of the leaves of T in order. A string σ is in the *language* generated by G just in case σ is the yield of some derivation for G.

B Properties

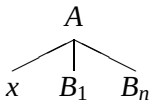
Each node in a TAGLET derived tree T is first contributed by a specific TAGLET element, and so indirectly by a particular anchor. Accordingly, we can construct a lexicalized *derivation tree* corresponding to T. Nodes in the derivation tree are labeled by the elements used in deriving T. An edge leads from parent E to child E' if T includes a step of derivation in which E' is substituted or sister-adjoined at a node first contributed by E. To make the derivation unambiguous, we record the address of the node in E at which the operation applies, and we order the edges in the derivation tree in the same order that the corresponding operations are applied in T. For Figure 3, we have:



Let L be a CFL. Then there is a grammar G for L in Greibach normal form (Hopcroft et al., 2000), where each production has the form

$$A \rightarrow xB_1 \ldots B_n$$

where $x \in V_T$ and $B_i \in V_N$. For each such production, create the TAGLET element which allows complementation with a tree as below:



An easy induction transforms any derivation in G to a derivation in this TAGLET grammar, and vice versa. So both generate the same language L.

Conversely, we can build a CFG for a TAGLET by creating nonterminals and productions for each node in a TAGLET elementary structure, taking into account the possibilities for optional premodification and postmodification as well as complementation.

C Parsing

Suppose we make a bottom-up traversal of a TAGLET derivation tree to construct the derived tree. After we finish with each node (and all its children), we obtain a subtree of the final derived tree. This subtree represents a complete constituent that must appear as a subsequence of the final sentence. A CKY TAGLET parser just reproduces this hierarchical discovery of constituents, by adding completed constituents for complements and modifiers into an open constituent for a head.

The only trick is to preserve linear order; this means adding each new complement and modifier at a possible "next place", without skipping past missing complements or slipping under existing modifiers. To do that, we only apply operations that add completed constituents T_2 along what is known as the *frontier* of the head tree T_1, further away from the head than previously incorporated material. This concept, though complex, is essential in any account of incremental structure-building. To avoid spurious ambiguities, we also require that operations to the left frontier must precede operations to the right frontier. This gives a relation $\text{COMBINE}(T_1, T_2, T_3)$.

The parser analyses a string of length N using a dynamic-programming procedure to enumerate all the analyses that span contiguous substrings, shortest substrings first. We write $T \in (i, j)$ to indicate that object T spans position i to j. The start of the string is position 0; the end is position N. So we have:

for word $w \in (i, i+1)$, T with anchor w
  add $T \in (i, i+1)$

for $k \leftarrow 2$ up to N
  for $i \leftarrow k-2$ down to 0
    for $j \leftarrow i+1$ up to $k-1$
      for $T_1 \in (i, j)$ and $T_2 \in (j, k)$
        for T_3 with $\text{COMBINE}(T_1, T_2, T_3)$
          add $T_3 \in (i, k)$

Now, any parser that delivers possible analyses exhaustively will be prohibitively expensive in the worst-case; analyses of ambiguities multiply exponentially. At the cost of a strong-competence implementation, one can imagine avoiding the complexity by maintaining TAGLET derivation forests. This enables $O(N^3)$ recognition, since TAGLET parsing operations apply within spans of the spine of single elementary trees and therefore the number of COMBINE results for T_1 and T_2 is independent of N.