Parsing with Lexicalized Probabilistic Recursive Transition Networks

Alexis Nasr and Owen Rambow

[1] Lattice-CNRS (UMR 8094),
Université Paris 7, Paris, France
alexis.nasr@linguist.jussieu.fr
[2] Center for Computational Learning Systems,
Columbia University, New York, NY, USA
rambow@cs.columbia.edu

Abstract. We present a formalization of lexicalized Recursive Transition Networks which we call Automaton-Based Generative Dependency Grammar (GDG). We show how to extract a GDG from a syntactically annotated corpus, present a chart parser for GDG, and discuss different probabilistic models which are directly implemented in the finite automata and do not affect the parser.

1 Introduction

While finite-state methods are becoming ever more popular in natural language processing (NLP), parsing (as opposed to chunking) has resisted the use of finite-state methods, presumably because of the difficulty of properly modeling structure using only finite state methods (but see [1]). An early proposal to extend finite state methods for syntax were the Recursive Transition Networks (RTNs) of [2], which add a stack mechanism to a collection of finite-state automata (FSMs). RTNs have been used to implement context-free grammars.

In the field of syntax, there has been much interest since the 1990's in lexicalized formalisms, in which each elementary structure of a grammar formalism represents the syntactic behavior of a single lexical item. The question arises what happens if we add lexicalization to RTNs. In this paper, we present probabilistic lexicalized RTN, which we call *Probabilistic Automaton-Based Generative Dependency Grammar* or GDGP. A GDGP is a collection of weighted FSMs, such that in each of these FSMs, every path includes at least one lexical transition. As with all lexicalized generative formalisms, the derivation tree is a dependency tree. GDG as a formalization allows us to relate RTNs to Tree Adjoining Grammars (TAG), and thus to profit from work on extracting TAGs from treebanks. We show how to convert a TAG extracted from a treebank into a GDG. We also show we can vary the conversion algorithm to obtain different automata which represent different ways of probabilistically modeling multiple attachments of the same type (such as adjectives attaching to a noun). Thus, in our approach, the automata represent both the algebraic part of the grammar and the probabilistic model. As a result the same algorithms (for parsing and searching for the best parses) are used for different probabilistic models.

The outline of the paper is as follows. We start out by presenting related work in section 2, and then present our definitions in section3. We very briefly present a simple parsing algorithm for GDG in section 4. We then turn to the key contributions of this paper: we present probabilistic models of adjunction in section 5, and then show how to extract a GDGP from a treebank (section 6).

2 Related Work

This work is based on previous work in string rewriting systems for dependency grammars, as well as on the notion of Recursive Transition Networks [2]. In this section, we quickly review the literature on such string-rewriting systems. The formalism presented here can be seen as having some similarities with [3, 4], who proposed generative formalisms for string rewriting. These formalisms were basically context-free grammars in which there is, on the right-hand side of rules, at least one terminal symbol. To overcome the inadequacy of such formalisms, [5] suggests extending the notation of [4] with regular expressions in the right-hand side, similar to the approach used in extended context-free grammars (for example, [6]). This approach was worked out in some detail in [7], and in a similar manner in [8], who present a string-rewriting version of GDG.

There has been some work on modeling syntactic dependency between words using automata. [9] use cascaded head automata to derive dependency trees, but leave the nature of the cascading under-formalized. [10] provides a formalization of a system that uses two different automata to generate left and right children of a head. His formalism bears some similarity to the one we present.

3 Generative Dependency Grammars

3.1 Informal Definition

A GDG is a set of finite-state automata (FSMs) of a particular type, namely *lexicalized automata*. A lexicalized automaton with the anchor (word) m describes all possible dependents of m. Each automaton has a name, which defines not only the part-of-speech of m, but also the active valency of m (i.e., all word classes that can depend on it), as well as their linear order. Thus this name can be thought of as a *supertag* in the sense of [11], and we will adopt the name "supertag" here to avoid confusion with simple part-of-speech tags. A sample lexicalized automaton is shown in Figure 1[3]. For expository purposes, in these examples, the supertags are simply standard part-of-speech tags. The transitions of the automaton are labeled with pairs $\langle f, c \rangle$, where f is a grammatical

[3] The initial state of an automaton is labeled 0 while its accepting states are indicated in boldface. The empty transitions are represented in dotted lines.

function (subject, object, different types of adjuncts, etc.), and c is a supertag, or by pairs $\langle \texttt{LEX}, m\rangle$, where m is an anchor of the automaton. This automaton indicates that the verb *eat* had a dependent which is its subject, obligatory and non-repeatable, and whose category is noun or pronoun; a dependent which is its object that is optional and non-repeatable; and an adjunct prepositional phrase which is repeatable.

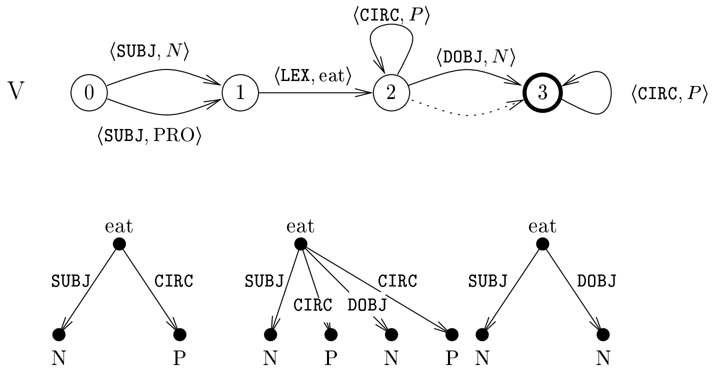

Fig. 1. A lexicalized automaton and three elementary trees that can be derived from it

Each word (in the formal language theory sense), i.e., each sentence (in the linguistic sense) accepted by an automaton is a sequence of pairs $\langle f, c\rangle$. Each such sequence corresponds to a dependency tree of depth one, which we will call an *elementary tree* of the grammar. Three sample elementary trees can be seen in the lower part of figure 1. The word corresponding to the leftmost tree is: $\langle \texttt{SUBJ}, N\rangle$ $\langle \texttt{LEX}, \mathrm{mange}\rangle$ $\langle \texttt{CIRC}, P\rangle$.

A GDG derivation is defined like a derivation in an RTN [2]. It uses a stack, which contains pairs $\langle c, e\rangle$ where c is the name of an automaton from the grammar, and e is a state of c. When $\langle c, e\rangle$ is on the top of the stack, and a transition of type $\langle f, c'\rangle$ goes from state e to state e' in automaton c, $\langle c, e\rangle$ is popped and $\langle c, e'\rangle$ is pushed as well as the machine c' in its initial state ($\langle c', q\rangle$). When we reach an accepting state q' in c', the pair $\langle c', q'\rangle$ is popped, uncovering $\langle c, e'\rangle$, and the traversal of automaton c resumes. We need to use a stack because, as we saw, during a derivation, several automata can be traversed in parallel, with one invoking the next recursively.

Since our automata are lexicalized, each traversal of a non-lexical arc (i.e., an arc of the form $\langle f, c\rangle$) corresponds to the establishment of a dependency between the lexical anchor of the automaton we are traversing and which we then put on the stack (as governor), and the lexical anchor of the new automaton which we start upon traversing the arc (as dependent). Thus, the result of a derivation can be seen as a sequence of transitions, which can be bijectively mapped to a dependency tree.

A probabilistic GDG, GDGP, is a GDG in which the automata of the grammar are weighted finite state automata. For each state in an automaton of the grammar, the weights of the outgoing arcs represent a probability distribution over possible transitions out of that state.

3.2 The Sites of an Automaton

The transitions of a lexicalized automaton do not all play the same role. We have already seen the lexical transitions which provide the words that anchor the automaton. In addition, we will distinguish the *argument transitions* which attach an argument as a dependent to the lexical anchor. All argument transitions which share the same grammatical function label constitute an *argument site* of the automaton. An example can be seen in Figure 2, where site 1 is the subject site, while site 4 is the object site. Note that since we consider in this example the grammatical object of *manger* to be optional, site 4 can be skipped using an ε -transition.

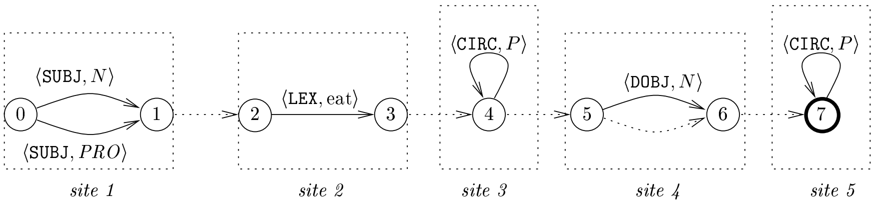

Fig. 2. Sites of the automaton in figure 1

The transitions associated with adjuncts are called *adjunct transitions*. They are grouped into *adjunct sites*, such as sites 3 and 5 in figure 2. Some adjunct sites are repeatable, while others (such as determiners in some languages) are not. When several dependencies are generated by the same repeatable adjunct site, we distinguish them by their *position*, which we mark with integers. The argument and adjunct sites are distinguished from the lexical transitions, which are called *lexical sites*.

4 Parsing with FSMs

The parsing algorithm is a simple extension of the chart parsing algorithm for context-free grammar (CFG). The difference is in the use of finite state machines in the items in the chart. In the following, we will call t-FSM an FSM M if its supertag is t. If T is the parse table for input sentence $W = w_1 \cdots w_n$ and GDG G, then $T_{i,j}$ contains (M, q) where M is a t-FSM and q is one of the accepting states of M, iff we have a complete derivation of substring $w_i \cdots w_j$ such that the root of the corresponding dependency tree is the lexical anchor of M with supertag t. If $T_{i,j}$ contains (M, q_1), if there is a transition in M from q_1 to q_2 labeled t, and if $T_{j+1,k}$ contains (M', q') where M' is a t-FSM and q' is an accepting state, then we add (M, q_2) to $T_{i,k}$. Note that because our grammars are lexicalized, each such step corresponds to one attachment of a lexical head to another as a dependent.

Before starting the parse, we create a tailored grammar by selecting those automata associated with the words in the input sentence. An important question is how to associate automata with words in a sentence; we do not discuss this issue in this paper, and refer to the literature on supertagging (for example, [11]). The parsing algorithm is extended to lattice input in the usual manner. The lattice represents several supertag sequences that can be associated to the sentence to parse. 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 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 phrase-structure trees is obtained from the packed forest and in a second stage, each phrase-structure tree is transformed into a dependency tree.

In order to deal with GDGP, we extend our parser by augmenting entries in the parse table with probabilities. The algorithm for extracting parses is augmented to choose the best parse (or n-best parses) in the usual manner.

5 Probabilistic Models

The parser introduced in Section 4 associates to a supertag sequence $S = S_1 \dots S_n$ one or several analyses. Each analysis $\mathcal{A}$ can be seen as a set of $n-1$ attachment operations (either adjunction or substitution) and the selection of one supertag token as the root of the analysis (the single supertag that is not attached in another supertag). For the sake of uniformity, we will consider the selection of the root as a special kind of attachment, $\mathcal{A}$ is therefore of cardinality n.

From a probabilistic point of view, each attachment operation is considered as an event and an analysis $\mathcal{A}$ as the joint event $A_1, \dots, A_n$. A large range of different models can be used to compute such a joint probability, from the simplest which considers that all events are independent to the model that considers that they are all dependent. The two models that we describe in this section vary in the way they model multiple adjuncts attaching at the same adjunct site. Put differently, the internal structure of repeatable adjunct sites is the only difference between the models. The reason to focus on this phenomenon comes from the fact that it is precisely at this level that much of the structural ambiguity occurs. The two models described below consider that attachments at argument sites are independent of all the other attachments that make up an analysis.

What is important is that the models we present in this section change the automata, but the changes are fully within sites; if we abstract to the level of sites, the automata are identical. Note that this hypothesis is not entailed by the GDGP formalism, one can produce a GDGP which changes the topology of the automata.

The two models for adjunction will be illustrated on a simple example where two automata c_1 and c_2 are candidates for attachment at a given repeatable adjunct site (which we will simply refer to as a "site"). Both sites can generate $(c_1|c_2)*$ but associate different probabilities to the generated strings. Recall that when several adjunctions occur at the same site, the first one is said to be of order 1, the second of order 2 and so on. The two models described below differ mainly in the fact the the first one (the positional model) focuses on the nature of the attachment at order i (how probable is it to have a prepositional attachment at order i) as well as on the number of attachments (how probable is it to have n attachments on this site). The second model (the bigram model) focuses on the dependency between an attachement and the preceding one (how probable is it to have a prepositional attachment following another prepositional attachment or an adjectival one). Both models have been used extensively in probabilistic models for parsing, but our use is slightly different as we only use these models for ordering within the same adjunct site. In the context of standard probabilistic context-free grammars, these models are ususally used to model the entire right-hand side of context-free rules.

5.1 Model 1: Positional Model

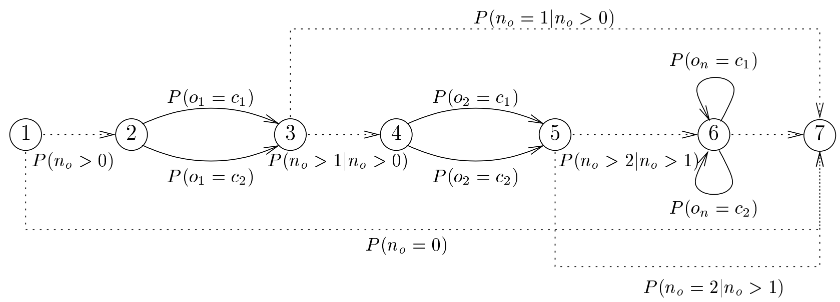

Fig. 3. Repeatable site with two distinguished positions

The automaton for a repeatable site with two positions is shown in Figure 3. It is made of a series of transitions between consecutive pairs of states. The first "bundle" of transitions models the first attachment at the site, the second bundle, the second attachment, and so on, until the maximum number of explicitly modeled attachments is reached, which is a parameter of the model. This limit on the number of attachments concerns only the probabilistic part of the automaton, more attachments can occur on this node, but their probabilities will not be distinguished. These additional attachments correspond to the loops on state 6 of the automaton. ϵ-transitions allow the attachments to stop at any moment by transitioning to state 7. Under Model 1, the probability of the sequences c_1c_2 and $c_1c_2c_1c_2$ being adjoined are:

$$P(c_1c_2) = P(o_1 = c_1) \times P(o_2 = c_2) \times P(n_o = 2)$$

$$P(c_1c_2c_1c_2) = P(o_1 = c_1) \times P(o_2 = c_2) \times P(o_n = c_1) \times P(o_n = c_2) \times P(n_o > 2)$$

Where variables o_1 and o_2 represent the first and second order adjunctions. Variable o_n represents adjunctions of order higher than 2. Variable n_o represents the total number of adjunctions.

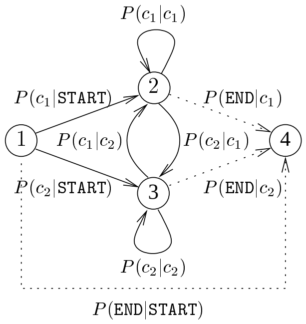

Fig. 4. Repeatable site with bigram modeling

5.2 Model 2: N-Gram Model

The previous model takes into account the nature of an attachment at a given order as well as the total number of attachements but disregards the nature of the attachments that happened before (or after) a given attachment. The model described here is, in a sense, complementary to the previous one since it takes into account, in the probability of an attachment, the nature of the $n-1$ attachments that occurred before and ignores the order of the current attachment. The probability of a series of attachments on the same side of the same node will be computed by an order-n Markov chain. The order of the Markov chain is a parameter of the model. The first order Markov chain for the repeatable site discussed is represented as a finite state automaton in Figure 4. The transitions with probabilities $P(x|\texttt{START})$ (respect. $P(\texttt{END}|x)$) correspond to the occurrence of automaton x as the first (respectively the last) attachment at this node and the transition with probability $P(\texttt{END}|\texttt{START})$ corresponds to the null adjunction (the probability that no adjunction occurs at a node). The probability of the sequence $c_1c_2c_1c_2$ being adjoined is now:

$$P(c_1c_2c_1c_2) = P(c_1|\texttt{START}) \times P(c_2|c_1) \times P(c_1|c_2) \times P(c_2|c_1) \times P(\texttt{END}|c_2)$$

6 Extracting a GDGP From a Corpus

We first describe the algebraic part of the extraction process, then briefly describe the estimation of the parameters of the probabilistic models.

6.1 Basic Approach

To extract a GDG (i.e., a lexicalized RTN) from the Penn Treebank (PTB), we first extract a TAG, and then convert it to a GDG. We make the detour via TAG for the following reason: we must extract an intermediate representation first in any case, as the automata in the GDG may refer in their transitions to any other automaton in the grammar. Thus, we cannot construct the automata until we have done a first pass through the treebank. We use TAG as the result of the first pass because this work has already been done, and we can reuse previous work, specifically the approach of [12] (which is similar to [13] and [14]). Note that the different models discussed in Section 5 only affect the manner in which the TAG grammar extracted from the corpus is converted to an FSM; the parsing algorithm (and code) is always the same.

We first briefly describe the work on TAG extraction, but refer the reader to the just cited literature for details. However, we optimize the head percolation in the grammar extraction module to create meaningful dependency structures, rather than (for example) maximally simple elementary tree structures. For example, we include long-distance dependencies (*wh*-movement, relativization) in elementary trees, we distinguish passive transitives without *by*-phrase from active intransitives, and we include strongly governed prepositions (as determined in the PTB annotation, including passive *by*-phrases) in elementary verbal trees as secondary lexical heads. Generally, function words such as auxiliaries or determiners are dependents of the lexical head,[4] conjunctions (including punctuation functioning as conjunction) are dependent on the first conjunct and take the second conjunct as their argument, and conjunction chains are represented as right-branching rather than flat.

In the second step, we directly compile this TAG into a set of FSMs which constitute the GDG and which are used in the parser. An FSM is built for each elementary tree of the TAG, during its depth-first traversal. For predicative auxiliary trees which are left auxiliary trees (in the sense of [15], i.e., all nonempty frontier nodes are to the left of the footnode), the traversal ends at the footnode.

[4] This is a linguistic choice and not forced by the formalism or the PTB. We prefer this representation as the resulting dependency tree is closer to predicate-argument structure.

For right auxilary predicative trees (which do not occur for English), the traversal would start at the footnode. In all other cases, the tree traversal goes from the root to the root (but excluding the root and foot nodes of adjunct auxiliary trees).

Non-leaf nodes are visited twice: first during the downward traversal, second during upward traversal. Each time a node is visited, a site is built in the corresponding automaton. Each transition in the site corresponds to an attachment that can be performed on the node. If the node visited is a substitution node of category X, a substitution site will be created in the FSM. The transitions in the substitution site are labeled with all the initial trees of the TAG whose root has category X. If the node is the lexical root of the elementary tree, a lexical site is created with one transition, labelled with the lexical anchor, if the elementary tree is lexicalized, or with the special symbol `HEAD` in the case of a tree schema. Internal nodes of the elementary tree give rise to adjunction sites in the automaton.

Finally, in the basic model in which adjunctions are modeled as independent, we proceed as follows for non-leaf nodes. (In Section 5, we discussed two other models that treat non-leaf nodes in a more complex manner.) 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. There are no other types of leaf nodes since we do not traverse the passive valency structure of adjunct auxiliary tees. 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. When an internal node of category X is visited during the downward traversal (upward traversal, respectively), we list all auxiliary trees that adjoin from the left (right) at a node of category X. For each of them, transitions are added to the adjunction site. Depending on the topology of the adjunction site (positionnal v/s N-gram), this addition of a transition will be realized differently.

We give a sample grammar in 5 and the result of converting one of its trees to an FSM in Figure 6[5].

6.2 Parameter estimation

During the production of the TAG, three kinds of counts are collected for each elementary tree schema T of the grammar: the number of times T has been selected as a root of a derivation tree, the number of substitutions of another elementary tree schema at the different substitution nodes of T, and the number of adjunctions of other elementary tree schemas at the internal nodes of T. Along with the last type of counts, the direction of the adjunction (left or right) is specified, as well as the order of the adjunction and the n preceding adjunctions of the same direction at this node.

[5] Due to space scarseness, we do not label the arcs of the automata of figure 6 with both probabilities and function, supertag pairs.

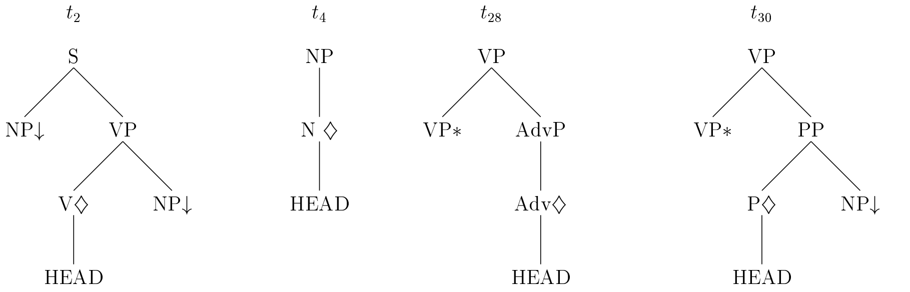

Fig. 5. Sample small grammar: trees for a transitive verb, a nominal argument, and two VP adjuncts from the right

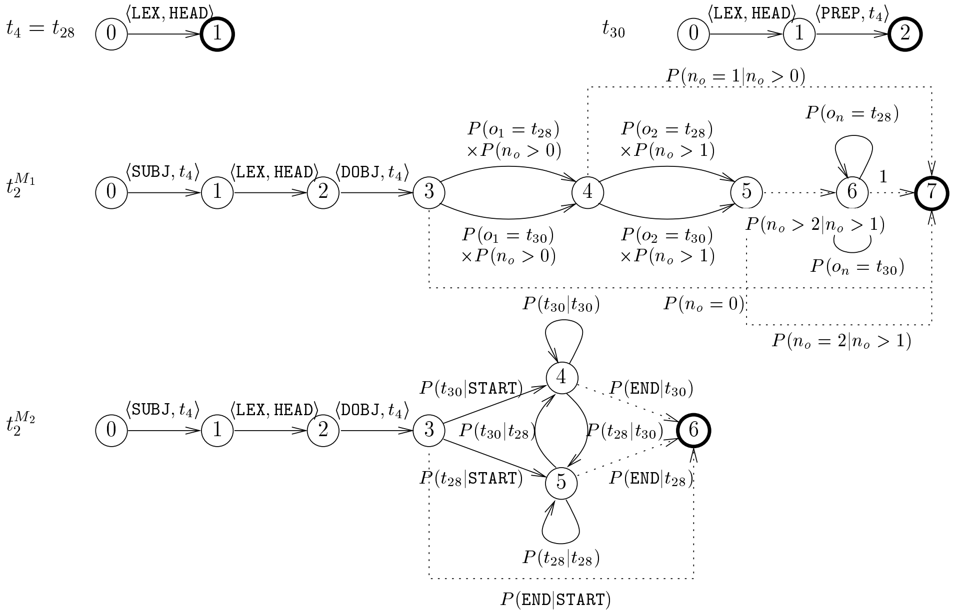

Fig. 6. FSMs derived from the grammar in figure 5. Two versions of tree t_2 has been built, corresponding to models 1 and 2.

These counts are used to estimate the root selection probabilities of the automata (the probability that an elementary tree schema constitues the root of a derivation tree) as well as the probabilities of their transitions. The probability of the initial probabilities, the substitution transitions and the adjunction probabilities in the positional models are estimated using simple add-one smoothing (actually, add-X smoothing with X tuned to 0.00001 on a devvelopment corpus), with the quantities added to the counts optimized on a developpment corpus. The adjunction probabilities in the N-gram models are smoothed using linear interpolation with lower order N-grams.

7 Conclusion

We have presented a probabilistic generative formalism for dependency grammars which can be seen as a probabilistic lexicalized version of Recursive Transition Networks. The topology of the automata that constitue the grammars can be modified in order to account for different probabilistic models. Two such models have been discussed. We showed how GDGP can be extracted from a treebank. Empirical results using GDG on the Penn Treebank have been presented in [8] and results on a French treebank can be found in [16].

Further work on this topic will focus on the coupling of a supertagger with the parser and the developpment of other probabilistic models.

References

1. Rambow, O., Bangalore, S., Butt, T., Nasr, A., Sproat, R.: Creating a finite-state parser with application semantics. In: Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002), Taipei, Republic of China (2002)
2. Woods, W.A.: Transition network grammars for natural language analysis. Commun. ACM **3** (1970) 591–606
3. Hays, D.G.: Dependency theory: A formalism and some observations. Language **40** (1964) 511–525
4. Gaifman, H.: Dependency systems and phrase-structure systems. Information and Control **8** (1965) 304–337
5. Abney, S.: A grammar of projections. Unpublished manuscript, Universität Tübingen (1996)
6. Madsen, O., Kristensen, B.: *LR*-parsing of extended context-free grammars. Acta Informatica **7** (1976) 61–73
7. Lombardo, V.: An Earley-style parser for dependency grammars. In: Proceedings of the 16th International Conference on Computational Linguistics (COLING'96), Copenhagen (1996)
8. Nasr, A., Rambow, O.: Supertagging and full parsing. In: Proceedings of the Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+7), Vancouver, BC, Canada (2004)
9. Alshawi, H., Bangalore, S., Douglas, S.: Learning dependency translation models as collections of finite-state head transducers. cl **26** (2000) 45–60

10. Eisner, J.M.: Three new probabilistic models for dependency parsing: An exploration. In: COLING'96, Copenhagen (1996)
11. Bangalore, S., Joshi, A.: Supertagging: An approach to almost parsing. Computational Linguistics **25** (1999) 237–266
12. Chen, J.: Towards Efficient Statistical Parsing Using Lexicalized Grammatical Information. PhD thesis, University of Delaware (2001)
13. Xia, F., Palmer, M., Joshi, A.: A uniform method of grammar extraction and its applications. In: Proc. of the EMNLP 2000, Hong Kong (2000)
14. Chiang, D.: Statistical parsing with an automatically-extracted tree adjoining grammar. In: 38th Meeting of the Association for Computational Linguistics (ACL'00), Hong Kong, China (2000) 456–463
15. Schabes, Y., Waters, R.C.: Tree Insertion Grammar: A cubic-time, parsable formalism that lexicalizes Context-Free Grammar without changing the trees produced. Computational Linguistics **21** (1995) 479–514
16. Nasr, A.: Analyse syntaxique probabiliste pour grammaires de dépendances extraites automatiquement. Habilitation à diriger des recherches, Université Paris 7 (2004)