A Simple String-Rewriting Formalism for Dependency Grammar

Alexis NASR
Lattice, UMR 8094
Université Paris 7
2, place Jussieu
F-75005 Paris, France
alexis.nasr@linguist.jussieu.fr

Owen RAMBOW
Columbia University
Department of Computer Science
1214 Amsterdam Avenue
New York, NY 10027-7003, USA
rambow@cs.columbia.edu

Abstract

Recently, dependency grammar has gained renewed attention as empirical methods in parsing have emphasized the importance of relations between words, which is what dependency grammars model explicitly, but context-free phrase-structure grammars do not. While there has been much work on formalizing dependency grammar and on parsing algorithms for dependency grammars in the past, there is not a complete generative formalization of dependency grammar based on string-rewriting in which the derivation structure is the desired dependency structure. In this paper, we present a simple generative formalism for dependency grammars based on Extended Context-Free Grammar, along with a parser; the formalism captures the intuitions of previous formalizations while deviating minimally from the much-used Context-Free Grammar.

1 Introduction

Dependency grammar has a long tradition in syntactic theory, dating back to at least TesnieÁre’s work from the thirties. Recently, it has gained renewed attention as empirical methods in parsing have emphasized the importance of relations between words (see, e.g., (Collins, 1997)), which is what dependency grammars model explicitly, but context-free phrase-structure grammars do not. In this paper, we address an important issue in using grammar formalisms: the compact representation of the parse forest. Why is this an important issue? It is well known that for non-toy grammars and non-toy examples, a sentence can have a staggeringly large number of analyses (for example, using a context-free grammar (CFG) extracted from the Penn Treebank, a sentence of 25 words may easily have 1,000,000 or more analyses). By way of an example of an ambiguous sentence (though with only two readings), the two dependency representations for the ambiguous sentence (1) are given in Figure 1.

(1) Pilar saw a man with a telescope

It is clear that if we want to evaluate each possible analysis (be it using a probabilistic model or a different method, for example a semantic checker), we cannot ef£ciently do so if we enumerate all cases. We have two options: we can either use a greedy method for checking, which entails we may miss the optimal solution, or we perform our checking operation on a representation which encodes all options in a compact representation. This is possible because the exponential number of possible analyses (exponential in the length of the input sentence) share subanalyses, thus making a polynomial-size representation possible. This representation is called the *shared parse forest* and it has been extensively studied for CFGs (see, for example, (Billot and Lang, 1989)). To our knowledge, there has been no description of the notion of shared parse forest for dependency trees to date. In this paper, we propose a formalization which is very closely based on the shared parse forest for CFG. We achieve this by de£ning a generative string-rewriting formalism whose derivation trees are dependency trees. The formalism, and the corresponding shared parse forests, are used in a probabilistic chart parser for dependency grammar, which is described in (Nasr and Rambow, 2004).

While there has been much work on formalizing dependency grammar and on parsing algorithms for dependency grammars in the past, we observe that there is not, to our knowledge, a complete generative formalization[1] of dependency grammar based on string-rewriting in which the derivation structure is *exactly* the desired dependency structure. The most salient reason for the lack of such a generative dependency grammar is the absence of nonterminal symbols in a dependency tree, which prevents us from interpreting it as a derivation structure in a system that distinguishes between terminal and nonterminal symbols. The standard solution to this problem, proposed by Gaifman (1965),

[1] While the notions are related historically and conceptually, we refer to a type of mathematical formalization, not to the school of linguistics known as “Generative Grammar”.

is to introduce nonterminal symbols denoting lexical categories, as depicted in figure 2 (called the "labelled phrase-structure trees induced by a dependency tree" by Gaifman (1965)). Clearly, the "pure" dependency tree can be derived in a straightforward manner. The string rewriting system described in (Gaifman, 1965) generates as derivation structures this kind of trees.

There is however a deeper problem when considering dependency trees as derivation structures, following from the fact that in a dependency tree, modifiers[2] are direct dependents of the head they modify, and (in certain syntactic contexts) the number of modifiers is unbounded. Thus, if we wish to obtain a tree as shown in Figure 2, we need to have productions whose right-hand side is of unbounded size, which is not possible in a context-free grammar. Indeed, the formalization of dependency grammar proposed by Gaifman (1965) is unsatisfactory in that it does not allow for an unbounded number of modifiers!

In this paper, we follow a suggestion made by Abney (1996) and worked out in some detail in (Lombardo, 1996)[3] to extend Gaifman's notation with regular expressions, similar to the approach used in extended context-free grammars. The result is a simple generative formalism which has the property that the derivation structures are dependency trees, except for the introduction of preterminal nodes as shown in Figure 2. We do not mean to imply that our formalism is substantially different from previous formalizations of dependency grammar; the goal of this paper is to present a clean and easy-to-use generative formalism with a straightforward notion of parse forest. In particular, our formalism, Generative Dependency Grammar, allows for an unbounded number of daughter nodes in the derivation tree through the use of regular expressions in its rules. The parser uses the corresponding finite-state machines which straightforwardly allows for a binary-branching representation of the derivation structure for the purpose of parsing, and thus for a compact (polynomial and not exponential) representation of the parse forest. This formalism is based on previous work presented in (Kahane et al., 1998), which has been substantially reformulated in order to simplify it.[4] In particular, we do not address non-projectivity here, but acknowledge that for certain languages it is a crucial issue. We will extend our basic approach in the spirit of (Kahane et al., 1998) in future work.

The paper is structured as follows. We start out by surveying previous formalizations of dependency grammar in Section 2. In Section 3, we introduce several formalisms, including Generative Dependency Grammar. We present a parsing algorithm in Section 4, and then conclude.

2 Previous Formalizations of Dependency Grammar

We start out by observing that "dependency grammar" should be contrasted with "phrase structure grammar", not "CFG", which is a particular formalization of phrase structure grammar. Thus, just as it only makes sense to study the formal properties of a particular formalization of phrase structure grammar, the question about the formal properties of dependency grammar in general is not well defined, nor the question of a comparison of a dependency formalism with dependency grammar.

There have been (at least) four types of formalizations of dependency grammars in the past.[5] None of these approaches, to our knowledge, discuss the notion of shared parse forest. 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 Dependency Tree Grammar of Molina (see (Dikovsky and Mod-

[2] We use the term *modifier* in its linguistic sense as a type of syntactic dependency (another type being *argument*). We use *head* (or *mother*) and *dependent* (or *daughter*) to refer to nodes in a tree. Sometimes, in the formal and parsing literature, *modifier* is used to designate any dependent node, but we consider that usage confusing because of the related but different meaning of the term *modifier* that is well-established in the linguistic literature.

[3] In fact, much of our formalism is very similar to (Lombardo, 1996), who however do not discuss parsing (only recognition), nor the representation of the parse forest.

[4] Kahane et al. (1998) present three different types of rules, for subcategorization, modification, and linear precedence. In the formalism presented in this paper, they have been collapsed into one.

[5] We leave aside here work on tree rewriting systems such as Tree Adjoining Grammar, which, when lexicalized, have derivation structures which are very similar to dependency trees. See (Rambow and Joshi, 1997) for a discussion related to TAG, and see (Rambow et al., 2001) for the definition of a tree-rewriting system which can be used to develop grammars whose derivations faithfully mirror syntactic dependency.

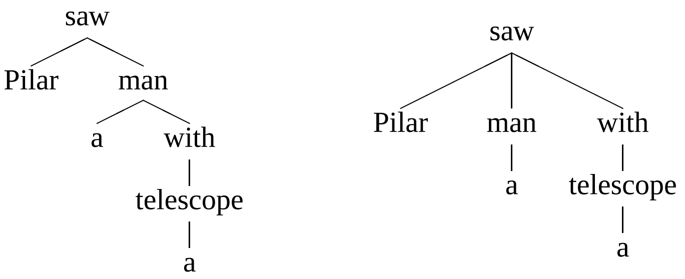

Figure 1: Two dependency trees

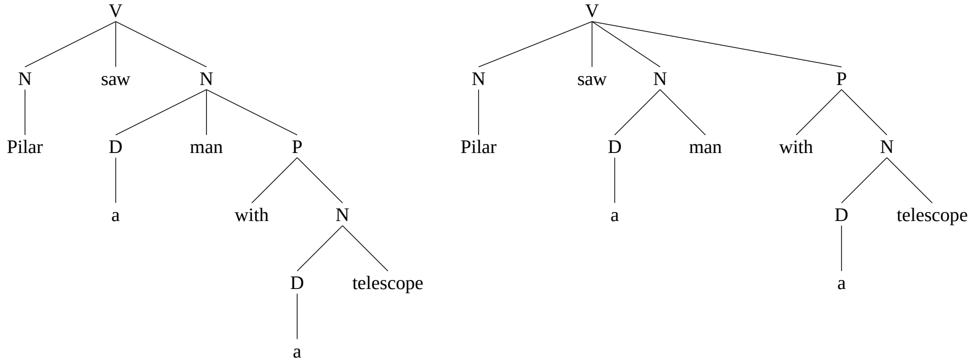

Figure 2: Two dependency trees with lexical categories

ina, 2000) for an overview), 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), also the Basic Dependency Grammar of Beletskij (1967) as cited in (Dikovsky and Modina, 2000). 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 speci£c 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 dependency 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 Formalism

In this section we £rst review the de£nition of Extended Context-Free Grammar and then show how we use it to model dependency derivations. An Extended Context-Free Grammar (or ECFG for short) is like a context-free grammar (CFG), except that the right-hand side is a regular expression over the terminal and nonterminal symbols of the grammar. At each step in a derivation, we £rst 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.

In the following, if G is a grammar and R a regular expression, then $L(G)$ denotes the language generated by the grammar and $L(R)$ the language denoted by the regular expression. If F is a class of grammars (such as CFG), then $\mathcal{L}$(F) denote the class of languages generated by the grammars in F. We now give a formal definition, which closely follows that given by Albert et al. (1999).[6]

A **Extended Context-Free Grammar** is a 4-tuple $(V_{\rm N}, V_{\rm T}, P, S)$, where:

- $V_{\rm N}$ is a finite set of nonterminal symbols,
- $V_{\rm T}$ is a finite set of terminal symbols (disjoint from $V_{\rm N}$),
- P is a finite set of rules, which are ordered pairs consisting of an element of $V_{\rm N}$ and a regular expression over $V_{\rm N} \cup V_{\rm T}$,
- S, a subset of $V_{\rm N}$, contains the possible start symbols.

We will use the traditional arrow notation ($\longrightarrow$) to write rules.

For $A \in V_{\rm N}$ and $u, v \in (V_{\rm N} \cup V_{\rm T})^*$ we say that uAv yields uwv (written $uAv \Longrightarrow uwv$) if $A \longrightarrow R$ is in P and w is in $L(R)$. The transitive closure of the yield relation (denoted $\overset{*}{\Longrightarrow}$) is defined in the usual manner.

The language generated by a Extended Context-Free Grammar is the set $\{w \in V_{\rm T}^* \mid A \overset{*}{\Longrightarrow} w, A \in S\}$.

We now define a restricted version of ECFG which we will use for defining dependency grammars. The only new formal requirement is that the rules be *lexicalized* in the sense of (Joshi and Schabes, 1991). 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. Linguistically speaking, this means that each rule is associated with exactly one lexical item (which may be multi-word). We will call this particular type of Extended Context-Free Grammar a **lexicalized Extended Context-Free Grammar** or, for obvious reasons, a Generative Dependency Grammar (GDG for short). 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.[7]

A **Generative Dependency Grammar** is a lexicalized ECFG.

It is sometimes useful to have dependency representations with labeled arcs (typically labeled with syntactic functions such as SUBJ for subject or ADJ for adjunct). There are different ways of achieving this goal; here, we discuss the use of feature structures in conjunction with the nonterminal symbols, for example `N[gf:subj]` instead of just `N`. Feature structures are of course useful for other reasons as well, such as expressing morphological features. In terms of our formalism, the use of bounded feature structures can be seen as a shorthand notation for an increased set of nonterminal symbols. The use of feature structures (rather than simple nonterminal symbols) allows for a more perspicuous representation of linguistic relations through the use of underspecification. Note that the use of underspecified feature structures in rules can potentially lead to an exponential increase (exponential in the size of the set of feature values) of the size of the set of rules if rules contain underspecified feature structures on the right-hand side. However, we note that the feature representing, grammatical function will presumably always be fully specified on the right-hand side of a rule (the head determines the function of its dependents). Underspecification in the left-hand side of a rule only leads to linear compactification of the rule set.

We now give a toy linguistic example. We let $G_{\rm Ling}$ be $(V_{\rm N}, V_{\rm T}, P, S)$ as follows:

- $V_{\rm N} = \{V, N, D, A, Adv\}$
- $V_{\rm T} = \{$ Pilar, saw, man, a, telescope, with, tall, very $\}$
- P consists of the following rules:

$$p_1 : V \longrightarrow N\ saw\ N\ P^*$$
$$p_2 : N \longrightarrow (Pilar \mid D\ (A)\ (man \mid telescope)\ P^*)$$
$$p_3 : P \longrightarrow with\ N$$
$$p_4 : A \longrightarrow Adv^*\ tall$$
$$p_5 : Adv \longrightarrow very$$
$$p_6 : D \longrightarrow a$$

- $S = \{V\}$

A derivation is shown in Figure 3; the corresponding derivation tree is shown in the right part

[6] ECFG has been around informally since the sixties (e.g., the Backus-Naur form); for a slightly different formalization, see (Madsen and Kristensen, 1976), whose definition allows for an infinite rule set.

[7] For practical purposes, we can separate the lexicon (which assigns lexical categories to lexemes) from the syntactic rules (which hold for all lexemes of a class), as does Gaifman (1965), resulting in a straightforward notational extension to our formalism.

$$
\begin{array}{l}
V \\
\xRightarrow{p_1} N\ saw\ N\ P \\
\xRightarrow{p_2 p_2 p_3} Pilar\ saw\ D\ man\ with\ N \\
\xRightarrow{p_6 p_2} Pilar\ saw\ a\ man\ with\ D\ telescope \\
\xRightarrow{p_6} Pilar\ saw\ a\ man\ with\ a\ telescope
\end{array}
$$

Figure 3: A sample GDG derivation

of Figure 2. As can be seen, the derivation structure is a dependency tree, except for the use of preterminals, as we desired.

The first part of the following theorem follows from the existence of a Greibach Normal Form for ECFG (Albert et al., 1999). The second part follows immediately from the closure of CFG under regular substitution.

$\mathcal{L}$(GDG) = $\mathcal{L}$(ECFG) = $\mathcal{L}$(CFG).

Of course, ECFG, GDG and CFG are not strongly equivalent in the standard sense for string rewriting systems of having the same sets of derivation trees. Clearly, ECFG can generate all sets of derivation trees that GDG can, while CFG cannot (because of the unbounded branching factor of ECFG and of GDG); ECFG can also generate all sets of derivation trees that CFG can, while GDG cannot (because of the lexicalization requirement). ECFG thus has a greater strong generative capacity than CFG and GDG, while those of GDG and CFG are incomparable.

It is interesting to notice the difference between the rewriting operation of a nonterminal symbol as defined for a ECFG or a GDG and the equivalent rewriting steps with a weakly equivalent CFG. A GDG rewriting operation of a symbol X using a rule r is decomposed in two stages, the first stage consists in choosing a string w which belongs to the set denoted by the right-hand side of r. During the second stage, X is replaced by w. These two stages are of a different nature, the first concerns the generation of CFG rules (and hence a CFG) using a GDG while the second concerns the generation of a string using the generated CFG. The equivalent rewriting operation ($X \Rightarrow w$) with a CFG does not distinguish the same two stages, both the selection of w and the rewriting of X as w are done at the same time.

4 Parsing Algorithm

The proposed parsing algorithm is a simple extension of the CKY algorithm. 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-**rule FSM**; its final states mark the completed recognition of a constituent of label C.

CKY-Style parsing algorithm for Extended Context-Free Grammars.

Input. A ECFG G and an input string $W = w_1 \cdots w_n$.

Output. The parse table T for W such that $t_{i,j}$ contains (M, q) iff M is a C-rule-FSM, q is one of the final states of M, and we have a derivation $C \overset{+}{\Longrightarrow} w_i \cdots w_j$. If $i = j$, $t_{i,j}$ also contains the input symbol w_i.

Method.

- **Initialization**: For each i, $1 \leq i \leq n$, add 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-rule-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. Add a single backpointer from (M', q') in $t_{i,j}$ to (M, q) or w in $t_{i,j}$.
- **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-rule-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. Add a double backpointer from (M_1, q) in $t_{i,j}$ to (M_1, q_1) in $t_{i,k}$ (left backpointer) and to either w or (M_2, q_2) in $t_{k+1,j}$ (right backpointer).

At the end of the parsing process, a packed parse forest has been built. The packed forest corresponding to the parse of sentence *Pilar saw a man with a telescope*, using the grammar of Section 3 is represented in Figure 5. The nonterminal nodes are labeled with pairs (M, q) where M is an rule FSM and q a state of this FSM. Three rules-FSM corresponding to rules p_1, p_2 and p_3 have been represented in Figure 4.

The parse forest can be used to extract the n most probable parses, as described in (Nasr and Rambow, 2004). The extracted parses are in the form of binary syntagmatic trees that can be transformed into dependency trees. The two trees that can be extracted from the forest of 5 are represented in Figure 6. The different nodes of each syntagmatic tree which are collapsed to a single node of the corresponding dependency tree have been circled. The edges that are contained in a circle will therefore disappear in the dependency tree while the edges

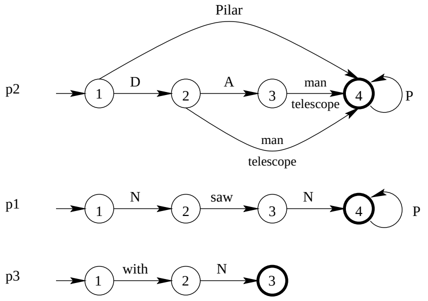

Figure 4: Three rules FSM m_1, m_2 and m_3. m_1 is a V-rule-FSM corresponding to rule p_1, m_2 is an N-rule-FSM which corresponds to rule p_2 and m_3 is a P-rule-FSM which corresponds to rule p_3

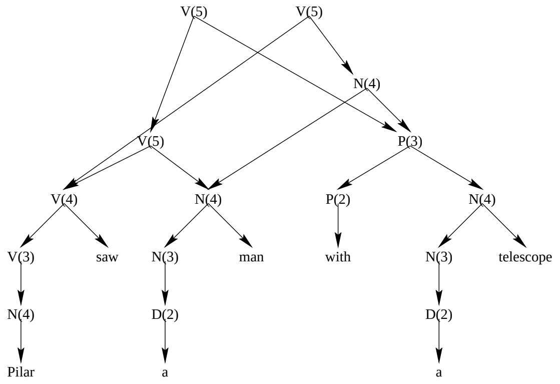

Figure 5: A packed parse forest

that link a circle to another one will give rise to dependencies.

5 Conclusion

We have presented a generative string-rewriting system, Extended Context-Free Grammar, whose derivation trees are dependency trees with unbounded branching factor. We have shown how we can reuse the representation of shared parse forests well-known from CFGs for Extended Context-Free Grammar. The question arises whether we can represent the shared parse forest in a manner more directly in the spirit of dependency. This question was investigated by (Nasr, 2003). He shows that the factoring realized in the shared forest presentend here and which is the key to the polynomial representation of a potentially exponential set, can be done directly on the dependency trees by introducing the notion of dependency sets.

References

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

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

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

Sylvie Billot and Bernard Lang. 1989. The structure of shared forests in ambiguous parsing. In *27th Meeting of the Association for Computational Linguistics (ACL'89)*, pages 143–151, Vancouver, Canada.

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

Alexander Dikovsky and Larissa Modina. 2000. Dependencies on the other side of the curtain. *Traitement Automatique des Langues*, 41(1):79–111.

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

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

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

Aravind K. Joshi and Yves Schabes. 1991. Tree-adjoining grammars and lexicalized grammars. In Maurice Nivat and Andreas Podelski, editors, *De£nability and Recognizability of Sets of Trees*. Elsevier.

Sylvain Kahane, Alexis Nasr, and Owen Rambow. 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.

Vincenzo Lombardo and Leonardo Lesmo. 2000. A formal theory of dependency syntax with

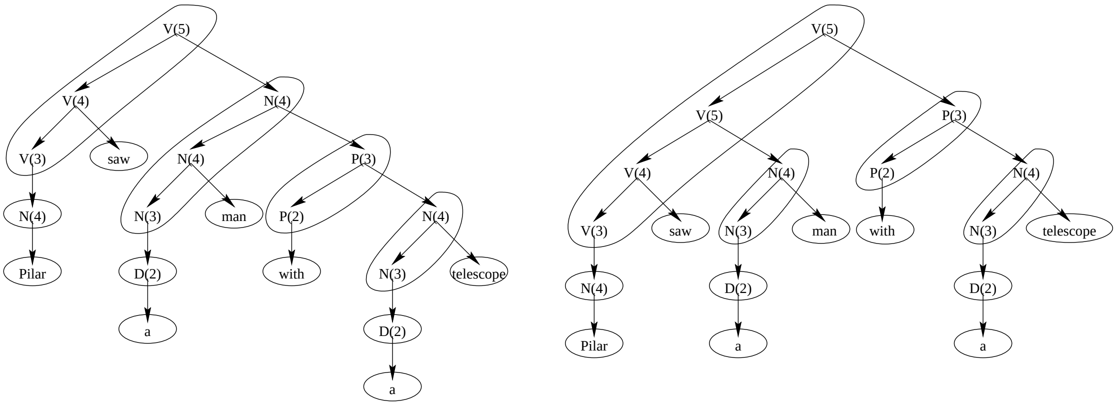

Figure 6: The extracted syntagmatic trees

empty units. *Traitement automatque des langues*, 41(1):179–209.

Vicenzo Lombardo. 1996. An Earley-style parser for dependency grammars. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING’96)*, Copenhagen.

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

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

Alexis Nasr and Owen Rambow. 2004. Supertagging and full parsing. In *7th International Workshop on Tree Adjoining Grammars and Related Frameworks*, Vancouver, Canada.

Alexis Nasr. 1996. *Un systÁme de reformulation automatique de phrases fondé sur la Théorie Sens-Texte : application aux langues contrôlées*. Ph.D. thesis, Université Paris 7.

Alexis Nasr. 2003. Factoring sufrace syntactic structures. In *First International Conference on Meaning-Text Theory*, pages 249–258, Paris, France.

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

Owen Rambow, K. Vijay-Shanker, and David Weir. 2001. D-Tree Substitution Grammars. *Computational Linguistics*, 27(1).

P. Sgall, E. Hajičová, and J. Panevová. 1986. *The meaning of the sentence and its semantic and pragmatic aspects*. Reidel, Dordrecht.

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

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