Pseudo-Projectivity: A Polynomially Parsable Non-Projective Dependency Grammar

Sylvain Kahane[*] and **Alexis Nasr**[†] and **Owen Rambow**[‡]
∗ TALANA Université Paris 7 (sk@ccr.jussieu.fr)
† LIA Université d'Avignon (alexis.nasr@lia.univ-avignon.fr)
‡CoGenTex, Inc. (owen@cogentex.com)

1 Introduction

Dependency grammar has a long tradition in syntactic theory, dating back to at least Tesnière's work from the thirties.[1] Recently, it has gained renewed attention as empirical methods in parsing are discovering the importance of relations between words (see, e.g., (Collins, 1997)), which is what dependency grammars model explicitly do, but context-free phrase-structure grammars do not. One problem that has posed an impediment to more wide-spread acceptance of dependency grammars is the fact that there is no computationally tractable version of dependency grammar which is not restricted to *projective* analyses. However, it is well known that there are some syntactic phenomena (such as *wh*-movement in English or clitic climbing in Romance) that require non-projective analyses. In this paper, we present a form of projectivity which we call *pseudo-projectivity*, and we present a generative string-rewriting formalism that can generate pseudo-projective analyses and which is polynomially parsable.

The paper is structured as follows. In Section 2, we introduce our notion of pseudo-projectivity. We briefly review a previously proposed formalization of projective dependency grammars in Section 3. In Section 4, we extend this formalism to handle pseudo-projectivity. We informally present a parser in Section 5.

2 Linear and Syntactic Order of a Sentence

2.1 Some Notation and Terminology

We will use the following terminology and notation in this paper. The **hierarchical order** (dominance) between the nodes of a **tree** T will be represented with the symbol $\prec^T$ and $\preceq^T$. Whenever they are unambiguous, the notations $\prec$ and $\preceq$ will be used. When $x \prec y$, we will say that x is a **descendent** of y and y an **ancestor** of x. The **projection** of a node x, belonging to a tree T, is the set of the nodes y of T such that $y \preceq^T x$. An **arc** between two nodes y and x of a tree T, directed from y to x will be noted either (y, x) or $\overleftarrow{x}$. The node x will be referred to as the **dependent** and y as the **governor**. The latter will be noted, when convenient, x^{+T} (x^+ when unambiguous). The notations $\overleftarrow{x}$ and x^+ are unambiguous because a node x has at most one governor in a tree. As usual, an **ordered tree** is a tree enriched with a linear order over the set of its nodes. Finally, if l is an arc of an ordered tree T, then *Supp(l)* represents the **support** of l, i.e. the set of the nodes of T situated between the extremities of l, extremities included. We will say that the elements of *Supp(l)* are **covered** by l.

2.2 Projectivity

The notion of projectivity was introduced by (Lecerf, 1960) and has received several different definitions since then. The definition given here is borrowed from (Marcus, 1965) and (Robinson, 1970):

Definition: An arc $\overleftarrow{x}$ is **projective** if and only if for every y covered by $\overleftarrow{x}$, $y \preceq x^+$. A tree T is **projective** if and only if every arc of T is projective

A projective tree has been represented in Figure 1.

A projective dependency tree can be associated with a phrase structure tree whose constituents are the projections of the nodes of the dependency tree. Projectivity is therefore equivalent, in phrase structure markers, to con-

[1] The work presented in this paper is collective and the order of authors is alphabetical.

Figure 1: A projective sub-categorization tree

tinuity of constituent.

The strong constraints introduced by the projectivity property on the relationship between hierarchical order and linear order allow us to describe word order of a projective dependency tree at a local level: in order to describe the linear position of a node, it is sufficient to describe its position towards its governor and sister nodes. The domain of locality of the linear order rules is therefore limited to a subtree of depth equal to one. It can be noted that this domain of locality is equal to the domain of locality of sub-categorization rules. Both rules can therefore be represented together as in (Gaifman, 1965) or separately as will be proposed in 3.

2.3 Pseudo-Projectivity

Although most linguistic structures can be represented as projective trees, it is well known that projectivity is too strong a constraint for dependency trees, as shown by the example of Figure 2, which includes a non-projective arc (marked with a star).

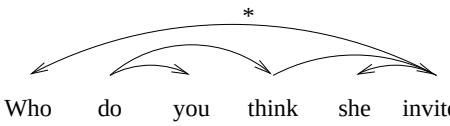

Figure 2: A non projective sub-categorization tree

The non projective structures found in linguistics represent a small subset of the potential non projective structures. We will define a property (more exactly a family of properties), weaker than projectivity, called **pseudo-projectivity**, which describes a subset of the set of ordered dependency trees, containing the non-projective linguistic structures.

In order to define pseudo-projectivity, we introduce an operation on dependency trees called **lifting**. When applied to a tree, this operation leads to the creation of a second tree, a **lift** of the first one. An ordered tree T' is a **lift** of the ordered tree T if and only if T and T' have the same nodes in the same order and for every node x, $x^{+T} \preceq^{T} x^{+T'}$. We will say that the node x has been lifted from x^{+T} (its **syntactic governor**) to $x^{+T'}$ (its **linear governor**).

Recall that the linear position of a node in a projective tree can be defined relative to its governor and its sisters. In order to define the linear order in a non projective tree, we will use a projective lift of the tree. In this case, the position of a node can be defined only with regards to its governor and sisters in the lift, i.e., its *linear* governor and sisters.

Definition: An ordered tree T is said **pseudo-projective** if there exists a lift T' of tree T which is projective.

If there is no restriction on the lifting, the previous definition is not very interesting since we can in fact take any non-projective tree and lift all nodes to the root node and obtain a projective tree.

We will therefore constrain the lifting by a set of rules, called *lifting rules*. Consider a set of (syntactic) categories. The following definitions make sense only for trees whose nodes are labeled with categories.[2]

The lifting rules are of the following form (LD, SG and LG are categories and w is a regular expression on the set of categories):

$$LD \uparrow SG\ w\ LG \tag{1}$$

This rule says that a node of category LD can be lifted from its syntactic governor of category SG to its linear governor of category LG through a path consisting of nodes of category $C_1, \ldots, C_n$, where the string $C_1 \ldots C_n$ belongs to $L(w)$. Every set of lifting rules defines a particular property of pseudo-projectivity by imposing particular constraints on the lifting. A

[2]It is possible to define pseudo-projectivity purely structurally (i.e. without referring to the labeling). For example, we can impose that each node x is lifted to the highest ancestor of x covered by $\overleftarrow{x}$ ((Nasr, 1996)). The resulting pseudo-projectivity is a fairly weak extension to projectivity, which nevertheless covers major non-projective linguistic structures. However, we do not pursue a purely structural definition of pseudo-projectivity in this paper.

linguistic example of lifting rule is given in Section 4.

The idea of building a projective tree by means of lifting appears in (Kunze, 1968) and is used by (Hudson, 1990) and (Hudson, unpublished). This idea can also be compared to the notion of word order domain (Reape, 1990; Bröker and Neuhaus, 1997), to the Slash feature of GPSG and HPSG, to the functional uncertainty of LFG, and to the Move-α of GB theory.

3 Projective Dependency Grammars Revisited

We (informally) define a projective Dependency Grammar as a string-rewriting system[3] by giving a set of *categories* such as N, V and Adv,[4] a set of distinguished start categories (the root categories of well-formed trees), a mapping from strings to categories, and two types of rules: **dependency rules** which state hierarchical order (dominance) and **LP rules** which state linear order. The dependency rules are further subdivided into subcategorization rules (or *s-rules*) and modification rules (or *m-rules*). Here are some sample s-rules:

$$d_1 : V_{\text{trans}} \longrightarrow N_{\text{nom}}, N_{\text{obj}}, \tag{2}$$

$$d_2 : V_{\text{clause}} \longrightarrow N_{\text{nom}}, V \tag{3}$$

Here is a sample m-rule.

$$d_3 : V \longrightarrow Adv \tag{4}$$

LP rules are represented as regular expressions (actually, only a limited form of regular expressions) associated with each category. We use the hash sign (#) to denote the position of the governor (head). For example:

$$p_1{:}V_{\text{trans}} = (Adv)N_{\text{nom}}(Aux)Adv^*\#N_{\text{obj}}Adv^*V_{\text{trans}} \tag{5}$$

[3]We follow (Gaifman, 1965) throughout this paper by modeling a dependency grammar with a string-rewriting system. However, we will identify a derivation with its representation as a tree, and we will sometimes refer to symbols introduced in a rewrite step as "dependent nodes". For a model of a DG based on tree-rewriting (in the spirit of Tree Adjoining Grammar (Joshi et al., 1975)), see (Nasr, 1995).

[4]In this paper, we will allow finite feature structures on categories, which we will notate using subscripts; e.g., V_{trans}. Since the feature structures are finite, this is simply a notational variant of a system defined only with simple category labels.

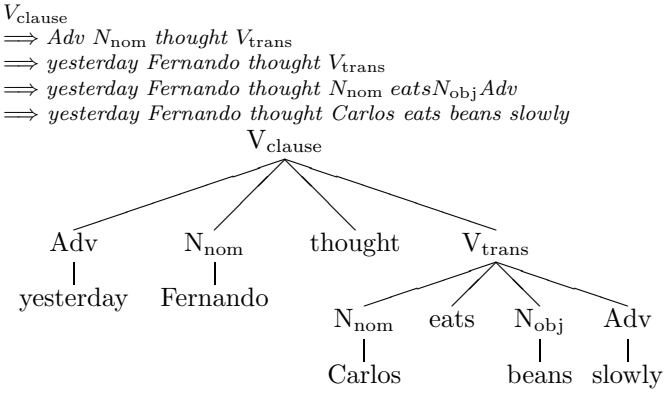

Figure 3: A sample GDG derivation

We will call this system **generative dependency grammar** or GDG for short.

Derivations in GDG are defined as follows. In a rewrite step, we choose a multiset of dependency rules (i.e., a set of instances of dependency rules) which contains exactly one s-rule and zero or more m-rules. The left-hand side nonterminal is the same as that we want to rewrite. Call this multiset the *rewrite-multiset*. In the rewriting operation, we introduce a multiset of new nonterminals and exactly one terminal symbol (the head). The rewriting operation then must meet the following three conditions:

- There is a bijection between the set of dependents of the instances of rules in the rewrite-multiset and the set of newly introduced dependents.
- The order of the newly introduced dependents is consistent with the LP rule associated with the governor.
- The introduced terminal string (head) is mapped to the rewritten category.

As an example, consider a grammar containing the three dependency rules d_1 (rule 2), d_2 (rule 3), and d_3 (rule 4), as well as the LP rule p_1 (rule 5). In addition, we have some lexical mappings (they are obvious from the example), and the start symbol is $V_{\text{finite:+}}$. A sample derivation is shown in Figure 3, with the sentential form representation on top and the corresponding tree representation below.

Using this kind of representation, we can derive a bottom-up parser in the following

straightforward manner.[5] Since syntactic and linear governors coincide, we can derive deterministic finite-state machines which capture both the dependency and the LP rules for a given governor category. We will refer to these FSMs as **rule-FSMs**, and if the governor is of category C, we will refer to a C-rule-FSM. In a rule-FSM, the transitions are labeled by categories, and the transition corresponding to the governor labeled by its category and a special mark (such as #). This transition is called the "head transition".

The entries in the parse matrix M are of the form (m, q), where m is a rule-FSM and q a state of it, except for the entries in squares $M(i, i)$, $1 \leq i \leq n$, which also contain category labels. Let $w_0 \cdots w_n$ be the input word. We initialize the parse matrix as follows. Let C be a category of word w_i. First, we add C to $M(i, i)$. Then, we add to $M(i, i)$ every pair (m, q) such that m is a rule-FSM with a transition labeled C from a start state and q the state reached after that transition.[6]

Embedded in the usual three loops on i, j, k, we add an entry (m_1, q) to $M(i, j)$ if (m_1, q_1) is in $M(k, j)$, (m_2, q_2) is in $M(i, k+1)$, q_2 is a final state of m_2, m_2 is a C-rule-FSM, and m_1 transitions from q_1 to q on C (a non-head transition). There is a special case for the head transitions in m_1: if $k = i - 1$, C is in $M(i, i)$, m_1 is a C-rule-FSM, and there is a head transition from q_1 to q in m_1, then we add (m_1, q) to $M(i, j)$.

The time complexity of the algorithm is $O(n^3|G|Q_{\max})$, where G is the number of rule-FSMs derived from the dependency and LP rules in the grammar and $Q_{\max}$ is the maximum number of states in any of the rule-FSMs.

4 A Formalization of PP-Dependency Grammars

Recall that in a pseudo-projective tree, we make a distinction between a *syntactic governor* and a *linear governor*. A node can be "lifted" along a lifting path from being a dependent of its syntactic governor to being a dependent of its linear governor, which must be an ancestor of the governor. In defining a formal rewriting system for pseudo-projective trees, we will not attempt to model the "lifting" as a transformational step in the derivation. Rather, we will directly derive the "lifted" version of the tree, where a node is dependent of its linear governor. Thus, the derived structure resembles more a unistratal dependency representation like those used by (Hudson, 1990) than the multistratal representations of, for example, (Mel'čuk, 1988). However, from a formal point of view, the distinction is not significant.

In order to capture pseudo-projectivity, we will interpret rules of the form (2) (for subcategorization of arguments by a head) and (4) (for selection of a head by an adjunct) as introducing syntactic dependents which may lift to a higher linear governor. An LP rule of the form (5) orders all *linear* dependents of the linear governor, no matter whose syntactic dependents they are.

In addition, we need a third type of rule, namely a *lifting rule*, or *l-rule* (see 2.3). The l-rule (1) can be rewrited on the following form:

$$l_1 : LG \longrightarrow LD \; \{LG \cdot w \; SG \; LD\} \qquad (6)$$

This rule resembles normal dependency rules but instead of introducing syntactic dependents of a category, it introduces a lifted dependent. Besides introducing a linear dependent LD, a l-rule should make sure that the syntactic governor of LD will be introduced at a later stage of the derivation, and prevent it to introduce LD as its syntactic dependent, otherwise non projective nodes would be introduced twice, a first time by their linear governor and a second time by their syntactic governor. This condition is represented in the rule by means of a constraint on the categories found along the lifting path. This condition, which we call *the lifting condition*, is represented by the regular expression $LG \cdot w \; SG$. The regular expression representing the lifting condition is enriched with a dot separating, on its left, the part of the lifting path which has already been introduced during the rewriting and on its right the part which is still to be introduced for the rewriting to be valid. The dot is an unperfect way of representing the current state in a finite state automaton equivalent to the regular expression. We can further notice that the lifting condition ends with a rep-

[5] This type of parser has been proposed previously. See for example (Lombardo, 1996; Eisner, 1996), who also discuss Early-style parsers for projective dependency grammars.

[6] We can use pre-computed top-down prediction to limit the number of pairs added.

etition of LD for reasons which will be made clear when discussing the rewriting process.

A sentential form contains terminal strings and categories paired with a multiset of lifting conditions, called the *lift multiset*. The lift multiset associated to a category C contains 'transiting' lifting conditions: introduced by ancestors of C and passing across C.

Three cases must be distinguished when rewriting a category C and its lifting multiset LM:

- LM contains a single lifting condition which dot is situated to its right: $LG\ w\ SG\ C\cdot$. In such a case, C must be rewritten by the empty string. The situation of the dot at the right of the lifting condition indicates that C has been introduced by its syntactic governor although it has already been introduced by its linear governor earlier in the rewriting process. This is the reason why C has been added at the end of the lifting condition.
- LM contains several lifting conditions one of which has its dot to the right. In such a case, the rewriting fails since, in accordance with the preceding case, C must be rewritten by the empty string. Therefore, the other lifting conditions of LM will not be satisfied. Furthermore, a single instance of a category cannot anchor more than one lifting condition.
- LM contains several lifting conditions none of which having the dot to their right. In this case, a rewrite multiset of dependency rules and lifting rules, both having C as their left hand side, is selected. The result of the rewriting then must meet the following conditions:

  1. The order of the newly introduced dependents is consistent with the LP rule associated with C.
  2. The union[7] of the lift multisets associated with all the newly introduced (instances of) categories is equal to the union of the lift multiset of C and the multiset composed of the lift condition of the l-rules used in the rewriting operation.
  3. The lifting conditions contained in the lift multiset of all the newly introduced dependents D should be compatible with D, with the dot advanced appropriately.

In addition, we require that, when we rewrite a category as a terminal, the lift multiset is empty.

Let us consider an example. Suppose we have have a grammar containing the dependency rules d_1 (rule 2), d_2 (rule 3), and d_3 (rule 4); the LP rule p_1 (rule 5) and p_2:

$$p_2{:}V_{\text{clause}} = (N_{\text{top:+}}|N_{\text{wh:+}})(Adv)N_{\text{nom}}(Aux)Adv^*\#Adv^*V_{\text{trans}}$$

Furthermore, we have the following l-rule:

$$l_1{:}V_{\text{bridge:+}} \longrightarrow N_{\text{case:obj top:+}}\{\cdot V^*_{\text{bridge:+}} V N_{\text{case:obj top:+}}\}$$

This rule says that an objective *wh*-noun with feature top:+ which depends on a verb with no further restrictions (the third V in the lifting path) can raise to any verb that dominates its immediate governor as long as the raising paths contains only verb with feature bridge:+, i.e., bridge verbs.

V_{clause}
$\Longrightarrow N_{\text{obj}}\ N_{\text{nom}}$ *thought Adv* $V\{\cdot V^*_{\text{bridge:+}}\ V\ N_{\text{case:obj top:+}}\}$
$\Longrightarrow$ *beans Fernando thought yesterday* $V\{\cdot V^*_{\text{bridge:+}}\ V\ N_{\text{case:obj top:+}}\}$
$\Longrightarrow$ *beans Fernando thought yesterday* N_{nom} *claims* $V\{\cdot V^*_{\text{bridge:+}}\ V\ N_{\text{case:obj top:+}}\}$
$\Longrightarrow$ *beans Fernando thought yesterday Milagro claims* $V\{\cdot V^*_{\text{bridge:+}}\ V\ N_{\text{case:obj top:+}}\}$
$\Longrightarrow$ *beans yesterday Fernando thought yesterday Milagro claims* N_{nom} *eats* $N\{V^*_{\text{bridge:+}} V N_{\text{case:obj top:+}}\cdot\}$ *Adv*
$\Longrightarrow$ *beans Fernando thought yesterday Milagro claims Carlos eats slowly*

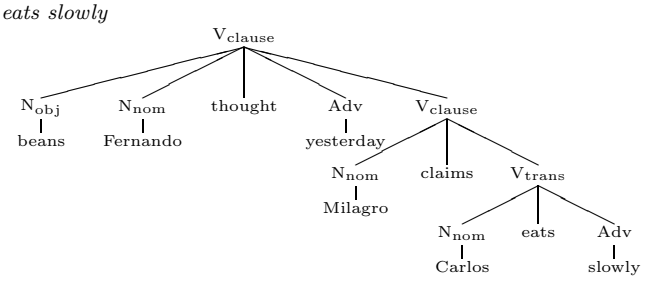

Figure 4: A sample PP-GDG derivation

A sample derivation is shown in Figure 4, with the sentential form representation on top

[7]When discussing set operations on multisets, we of course mean the corresponding multiset operations.

and the corresponding tree representation below. We start our derivation with the start symbol V_{clause} and rewrite it using dependency rules d_2 and d_3, and the lifting rule l_1 which introduces an objective NP argument. The lifting condition of l_1 is passed to the V dependent but the dot remains at the left of $V_{\text{bridge:+}}$ because of the Kleene star. When we rewrite the embedded V, we choose to rewrite again with V_{clause}, and the lifting condition is passed on to the next verb. This verb is a V_{trans} which requires a N_{obj}. The lifting condition is passed to N_{obj} and the dot is moved to the right of the regular expression, therefore N_{obj} is rewritten as the empty string.

5 A Polynomial Parser for PP-GDG

In this section, we show that pseudo-projective dependency grammars as defined in Section 2.3 are polynomially parsable.

We can extend the bottom-up parser for GDG to a parser for PP-GDG in the following manner. In PP-GDG, syntactic and linear governors do not necessarily coincide, and we must keep track separately of linear precedence and of lifting (i.e., "long distance" syntactic dependence).

The entries in the parse matrix M are of the form (m, q, LM), where m is a rule-FSM, q a state of m, and LM is a multiset of lifting conditions as defined in Section 4. An entry (m, q, LM) in a square $M(i, j)$ of the parse matrix means that the sub-word $w_i \cdots w_j$ of the entry can be analyzed by m up to state q (i.e., it matches the beginning of an LP rule), but that nodes corresponding to the lifting rules in LM are being lifted from the subtrees spanning $w_i \cdots w_j$. Put differently, in this bottom-up view LM represents the set of nodes which have a syntactic governor in the subtree spanning $w_i \cdots w_j$ and a lifting rule, but are still looking for a linear governor.

Suppose we have an entry in the parse matrix M of the form (m, q, L). As we traverse the C-rule-FSM m, we recognize one by one the linear dependents of a node of category C. Call this governor η. The action of adding a new entry to the parse matrix corresponds to adding a single new linear dependent to η. (While we are working on the C-rule-FSM m and are not yet in a final state, we have not yet recognized η itself.) Each new dependent η' brings with it a multiset of nodes being lifted from the subtree it is the root of. Call this multiset LM'. The new entry will be $(m, q', LM \cup LM')$ (where q' is the state that m transitions to when η' is recognized as the next linear dependent.

When we have reached a final state q of the rule-FSM m, we have recognized a complete subtree rooted in the new governor, η. Some of the dependent nodes of η will be both syntactic and linear dependents of η, and the others will be linear dependents of η, but lifted from a descendent of η. In addition, η may have syntactic dependents which are not realized as its own linear dependent and are lifted away. (No other options are possible.) Therefore, when we have reached the final state of a rule-FSM, we must connect up all nodes and lifting conditions before we can proceed to put an entry (m, q, L) in the parse matrix. This involves these steps:

1. For every lifting condition in LM, we ensure that it is compatible with the category of η. This is done by moving the dot leftwards in accordance with the category of η. (The dot is moved leftwards since we are doing bottom-up recognition.)

   The obvious special provisions deal with the Kleene star and optional elements. If the category matches a catgeory with Kleene start in the lifting condition, we do not move the dot. If the category matches a category which is to the left of an optional category, or to the left of category with Kleene star, then we can move the dot to the left of that category.

   If the dot cannot be placed in accordance with the category of η, then no new entry is made in the parse matrix for η.

2. We then choose a multiset of s-, m-, and l-rules whose left-hand side is the category of η. For every dependent of η introduced by an l-rule, the dependent must be compatible with an instance of a lifting condition in LM (whose dot must be at its beginning, or seperated from the beginning by optional or categories only); the lifting condition is then removed from L.

3. If, after the above repositioning of the dot and the linking up of all linear dependents to lifting conditions, there are still lifting

conditions in LM such that the dot is at the beginning of the lifting condition, then no new entry is made in the parse matrix for η.

4. For every syntactic dependent of η, we determine if it is a linear dependent of η which has not yet been identified as lifted. For each syntactic dependents which is not also a linear dependent, we check whether there is an applicable lifting rule. If not, no entry is made in the parse matrix for η. If yes, we add the lifting rule to LM.

This procedure determines a new multiset LM so we can add entry (m, q, LM) in the parse matrix. (In fact, it may determine several possible new multisets, resulting in multiple new entries.) The parse is complete if there is an entry $(m, q_m, \emptyset)$ in square $M(n, 1)$ of the parse matrix, where m is a C-rule-FSM for a start category and q_m is a final state of m. If we keep backpointers at each step in the algorithm, we have a compact representation of the parse forest.

The maximum number of entries in each square of the parse matrix is $O(GQn^L)$, where G is the number of rule-FSMs corresponding to LP rules in the grammar, Q is the maximum number of states in any of the rule-FSMs, and L is the maximum number of states that the lifting rules can be in (i.e., the number of lifting conditions in the grammar multiplied by the maximum number of dot positions of any lifting condition). Note that the exponent is a grammar constant, but this number can be rather small since the lifting rules are not lexicalized – they are construction-specific, not lexeme-specific. The time complexity of the algorithm is therefore $O(GQn^{3+2|L|})$.

References

Norbert Bröker and Peter Neuhaus. 1997. The complexity of recognition of linguistically adequate dependency grammars. In *ACL'97*, Madrid, Spain.

M. Collins. 1997. Three generative, lexicalised models for statistical parsing. In *ACL'97*, Madrid, Spain.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *COLING'96*, Copenhagen.

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

Richard Hudson. 1990. *English Word Grammar*. Basil Blackwell, Oxford, RU.

Richard Hudson. unpublished. Discontinuity. e-preprint (ftp.phon.ucl.ac.uk).

Aravind Joshi, Leon Levy, and M Takahashi. 1975. Tree adjunct grammars. *J. Comput. Syst. Sci.*, 10:136–163.

Jürgen Kunze. 1968. The treatment of non-projective structures in the syntactic analysis and synthesis of english and german. *Computational Linguistics*, 7:67–77.

Yves Lecerf. 1960. Programme des conflits, modèle des conflits. *Bulletin bimestriel de l'*ATALA, 4,5.

Vicenzo Lombardo. 1996. An earley-style parser for dependency grammars. In *COLING'96*, Copenhagen.

Solomon Marcus. 1965. Sur la notion de projectivité. *Zeitschr. f. math. Logik und Grundlagen d. Math.*, 11:181–192.

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

Alexis Nasr. 1995. A formalism and a parser for lexicalised dependency grammars. In *4th International Workshop on Parsing Technologies*, pages 186–195, Prague.

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

Michael Reape. 1990. Getting things in order. In *Proceedings of the Symposium on Discontinuous Constituents*, Tilburg, Holland.

Jane J. Robinson. 1970. Dependency structures and transformational rules. *Language*, 46(2):259–285.