Parsing Mildly Non-projective Dependency Structures*

Carlos Gomez-Rodríguez[†]
Departamento de Computacion
Universidade da Coruña, Spain
cgomezr@udc.es

David Weir and John Carroll
Department of Informatics
University of Sussex, United Kingdom
{davidw,johnca}@sussex.ac.uk

December 19, 2008

Abstract

We present novel parsing algorithms for several sets of mildly non-projective dependency structures.

First, we define a parser for well-nested structures of gap degree at most 1, with the same complexity as the best existing parsers for constituency formalisms of equivalent generative power. We then extend this algorithm to handle all well-nested structures with gap degree bounded by any constant k.

Finally, we define a parsing algorithm for a new class of structures with gap degree up to k that includes some ill-nested structures. This set of structures, which we call mildly ill-nested, includes all the gap degree k structures in a number of dependency treebanks.

1 Introduction

of predicate argument structure. We take dependency structures to be directed trees, where each node corresponds to a word and the root of the tree marks the syntactic head of the sentence. For reasons of e ciency, many practical implementations of dependency parsing are restricted to *projective* structures, in which the subtree rooted at each word covers a contiguous substring of the sentence. However, while free word order languages such as Czech do not satisfy this constraint, parsing without the projectivity constraint is computationally complex. Although it is possible to parse non-projective structures in quadratic time under a model in which each dependency decision is independent of all the others (McDonald et al., 2005), the problem is intractable in the absence of this assumption (McDonald and Satta, 2007).

Nivre and Nillson (2005) observe that most non-projective dependency structures appearing in practice are \close" to being projective, since they contain only a small proportion of non-projective arcs. This has led to the study of classes of dependency structures that lie between projective and unrestricted non-projective structures (Kuhlmann and Nivre, 2006; Havelka, 2007). Kuhlmann (2007) investigates several such classes, based on well-nestedness and gap degree constraints (Bodirsky et al., 2005), relating them to lexicalised constituency grammar formalisms. Speci cally, he shows that: linear context-free rewriting systems (LCFRS) with fan-out k (Vijay-Shanker et al., 1987; Satta, 1992) induce the set of dependency structures with gap degree at most $k-1$; coupled context-free grammars in which the maximal rank of a nonterminal is k (Hotz and Pitsch, 1996) induce the set of well-nested dependency structures with gap degree at most $k-1$; and LTAGs (Joshi and Schabes, 1997) induce the set of well-nested dependency structures with gap degree at most 1.

These results establish that there must be polynomial-time dependency parsing algorithms for well-nested structures with bounded gap degree, since such parsers exist for their corresponding lexicalised constituency-based formalisms. However, since most of the non-projective structures in treebanks are well-nested and have a small gap degree (Kuhlmann and Nivre, 2006), developing e cient dependency parsing strategies for these sets of structures has considerable practical interest, since we would be able to parse directly with dependencies in a data-driven manner, rather than indirectly by constructing intermediate constituency grammars and extracting dependencies from constituency parses.

We address this problem with the following contributions:

- We de ne a parsing algorithm for well-nested dependency structures of gap degree 1, and prove its correctness. The parser runs in time $O(n^7)$, the same complexity as the best existing algorithms for LTAG (Eisner and Satta, 2000), and can be optimised to $O(n^6)$ in the non-lexicalised case.

- We generalise the previous algorithm to any well-nested dependency structure with gap degree at most k in time $O(n^{5+2k})$.

- We generalise the previous parsers to be able to analyse not only well-nested structures, but also ill-nested structures with gap degree at most k satisfying certain constraints[1], in time $O(n^{4+3k})$.

- We characterise the set of structures covered by this parser, which we call *mildly ill-nested* structures, and show that it includes all the trees present in a number of dependency treebanks.

2 Preliminaries

A *dependency graph* for a string $w_1 \dots w_n$ is a graph $G = (V, E)$, where $V = \{w_1$

well-nested is said to be **ill-nested**. Note that projective trees are always well-nested, but well-nested trees are not always projective.

2.2 Dependency parsing schemata

The framework of parsing schemata (Sikkel, 1997) provides a uniform way to describe, analyse and compare parsing algorithms. Parsing schemata were initially de ned for constituency-based grammatical formalisms, but Gomez-Rodr guez et al. (2008) de ne a variant of the framework for dependency-based parsers. We use these *dependency parsing schemata* to de ne parsers and prove their correctness. We will now provide brief outlines of the main concepts behind dependency parsing schemata.

The parsing schema approach considers parsing as deduction, generating intermediate results called *items*. An initial set of items is obtained from the input sentence, and the parsing process involves *deduction steps*

3 The WG_1 parser

3.1 Parsing schema for WG_1

We de ne WG_1, a parser for well-nested dependency structures of gap degree ≤ 1, as follows:

The item set is $\mathcal{I}_{WG1} = \mathcal{I}_1 \cup \mathcal{I}_2$, with

$$\mathcal{I}_1 = \{[i, j, h, \diamond, \diamond] \mid i, j, h \in \mathbb{N}, 1 \leq h \leq n, 1 \leq i \leq j \leq n, h \neq j, h \neq i-1\},$$

where each item of the form $[i, j, h, \diamond, \diamond]$ represents the set of all well-nested partial dependency trees[3] with gap degree at most 1, rooted at w_h, and such that $\lfloor w_h \rfloor = \{w_h\} \cup [i, j]$, and

$$\mathcal{I}_2 = \{[i, j, h, l, r] \mid i, j, h, l, r \in \mathbb{N}, 1 \leq h \leq n, 1 \leq i < l \leq r < j \leq n, \\ h \neq j, h \neq i-1, h \neq l-1, h \neq r\}$$

where each item of the form $[i, j, h, l, r]$ represents the set of all well-nested partial dependency trees rooted at w_h such that $\lfloor w_h \rfloor = \{w_h\} \cup ([i, j] \setminus [l, r])$, and all the nodes (except possibly h) have gap degree at most 1. We call items of this form *gapped items*, and the interval $[l, r]$ the *gap* of the item. Note that the constraints $h \neq j, h \neq i+1, h \neq l-1, h \neq r$ are added to items to avoid redundancy in the item set. Since the result of the expression $\{w_h\} \cup ([i, j] \setminus [l, r])$ for a given head can be the same for di erent sets of values of i, j, l, r, we restrict these values so that we cannot get two di erent items representing the same dependency structures. Items violating these constraints always have an alternative representation that does not violate them, that we can express with a normalising function $nm($

used for all the parsers, so we do not make it explicit for subsequent schemata. Note that initial items are separate from the item set $\mathcal{I}_{WG1}$ and not subject to its constraints, so they do not require normalisation.

The set of final items for strings of length n in WG_1 is defined as the set

$$\mathcal{F} = \{[1; n; h; \diamond; \diamond] \mid h \in \mathbb{N}; 1 \leq h \leq n\};$$

which is the set of the items in $\mathcal{I}_{WG1}$ containing dependency trees for the complete input string (from position 1 to n), with their head at any word w_h.

Finally, the deduction steps of the WG_1 parser are the following:

Link Ungapped:

$$\frac{[h1; h1; h1; \diamond; \diamond] \quad [i2; j2; h2; \diamond; \diamond]}{[i2; j2; h1; \diamond; \diamond]} \; w_{h2} \rightarrow w_{h1}$$

such that w_{h2}

of linking each of the dependent subtrees to the new head w_h; (2) applying the various *Combine* steps to join all of the items obtained in the previous step into a single item. The *Combine* steps perform a union operation between subtrees. Therefore, the result is a dependency tree containing all the dependent subtrees, and with all of them linked to h: this is the subtree induced by w_h. This process is applied repeatedly to build larger subtrees, until, if the pars.9701 by

3.2.1 Soundness

Proving the soundness of the WG_1

and by checking the steps one by one we can see that their constraints guarantee that this union satisfies the condition. The gap degree of the head in T_c is guaranteed to be at most 2 by this condition (1), and the gap degree of the rest of the nodes in T_c is guaranteed to be ≤ 1 because their induced subtrees are the same as in the antecedent tree T_a or T_b in which they appeared (note that, by construction of the antecedents of *Combiner* steps, the only node that appears both in T_a and T_b is w_h, so the rest of the nodes in T_c can only come from one of the antecedent trees). Therefore, (2) also holds. Regarding well-nestedness, we note that the subtree induced by the head of the consequent tree cannot interleave with any other, and the rest of the subtrees are the same as in the antecedent trees. Thus, since the subtrees in each antecedent tree did not interleave among themselves (T_a and T_b are well-nested), the only way in which the consequent tree could be ill-nested would be having a subtree of one antecedent tree interleaving with a subtree of the other antecedent tree. This can be checked step by step, and in every single *Combiner* step we can see that two subtrees coming from each of the antecedent trees cannot interleave. As an example, in a *Combine Closing Gap* step:

$$\frac{[i, j, h, l, r] \qquad [l, r, h, \diamond, \diamond]}{[i, j, h, \diamond, \diamond]}$$

In order for a subtree in the second antecedent to be able to interleave with a subtree in the first antecedent, it would need to have nodes in the interval $[l, r]$ and nodes in the set $[1, i-1] \cup [j+1, n]$, but this is impossible by construction, since the projection of a tree in the second antecedent is of the form $\{w_h\} \cup [l, r]$.

Analogous reasoning can be applied for the rest of the *Combiner* steps, concluding that all of them preserve well-nestedness. With this we have proven (ii), and therefore the soundness of WG_1.

3.2.2 Order annotations

In the completeness proof for WG_1, we will use the concept of *order annotations* (Kuhlmann, 2007; Kuhlmann and Möhl, 2007). Here we will outline the concept and some properties relevant to the proof, a more detailed discussion can be found in (Kuhlmann, 2007).

Order annotations are strings that encode the precedence relation between the nodes of a dependency tree: if we take a dependency tree with its words unordered and decorate each node with an order annotation, we will obtain a particular ordering for the words. Order annotations are related to projectivity, gap degree and well-nestedness: there exists a set of order annotations that, when applied to nodes in any structure, will result in an ordering of the nodes that satisfies projectivity, and the same can be said about the properties of well-nestedness and having gap degree bounded by a constant k. In addition to this, order annotations are closely related to the way in which the parsers defined in this report construct subtrees with their *Combine* steps, and this will make them useful for proving their correctness.

Let T be a dependency structure for a string $w_1 \dots w_n$, and w_k a node in T. Let $w_{d_1} \dots w_{d_p}$ be the direct dependents of w_k in T, ordered by the position of the leftmost element in their projection, i.e. $min\{i \in \mathbb{N} \mid w_i \in \lfloor w_{d_u} \rfloor\} < min\{j \in \mathbb{N} \mid w_j \in \lfloor w_{d_v} \rfloor\}$ if and only if $u < v$.

The order annotation for a node w_k is a string over the alphabet $\{0;1;\ldots;p\} \cup \{\text{\;"}\}$ obtained from the following process:

- Build a string $a(T;w_k) = a_1a_2\ldots a_n$, where $a_k = 0$, $a_i = u$ if $i \in \lfloor w_{d_u} \rfloor$, and $a_i = \text{\,"}$ (comma) otherwise (i.e. if $i \notin \lfloor w_k \rfloor$).
- The order annotation for w_k, $o(T;w_k)$, is the string obtained by collapsing all adjacent occurences of the same symbol in $a(T;w_k)$ into a single occurence, and removing all leading and trailing commas.[4]

By construction, order annotations have the following property:

Property 1. *If the order annotation for a node w_k is a string $o(T;w_k) = o_1\ldots o_q$, then there exist unique natural numbers $i_1 < i_2;\ldots < i_{q+1}$ such that:*

- *If the symbol 0 appears in position v in $o(T;w_k)$, then $i_v = k$ and $i_{v+1} = k+1$.*
- *If a symbol $s \in (\mathbb{N} \setminus \{0\})$ appears in positions $v_1;\ldots;v_r$ in $o(T;w_k)$, then the projection of the sth dependent of w_k in T is $\{[i_{v_1}; i_{v_1+1} - 1]\} \cup \{[i_{v_2}; i_{v_2+1} - 1]\} \cup \ldots \cup \{[i_{v_r}; i_{v_r+1} - 1]\}$.*

In particular, it can be checked that i_1 is always the index associated to the leftmost node in $\lfloor w_k \rfloor$, i_{q+1} the index associated to the rightmost node in $\lfloor w_k \rfloor$ plus 1, and for each i_v such that $1 < v \leq q$, the differences $d_v = (i_v - i_1)$ correspond to the positions in the intermediate string $a(T;w_k)$ such that the d_vth symbol in $a(T;w_k)$ differs from the $(d_v + 1)$th.

By using this property to reason about the projections of a dependency tree's nodes, we can show the following, more particular properties:

Property 2.

A node w_k has gap degree g in a dependency structure T if, and only if, the comma symbol (;) appears g times in $o(T;w_k)$.

(Corollary 1) The gap degree of a dependency structure T is the maximum value among the number of commas in the order annotations of each of its nodes.

(Corollary 2) A dependency structure is projective if, and only if, none of the order annotations associated to its nodes contain a comma.

Property 3. *If a number $s \in (\mathbb{N} \setminus \{0\})$ appears $g+1$ times in an order annotation $o(T;w_k)$, then the sth direct child of w_k (in the ordering mentioned earlier) has gap degree g, and therefore the dependency structure T has gap degree at least g.*

Property 4. *A dependency structure T is ill-nested if, and only if, it contains at least one order annotation of the form $:::a:::b:::a:::b:::$, for some $a;b \in (\mathbb{N} \setminus \{0\})$. Otherwise, T is well-nested.*

These properties allow us to define the sets of structures verifying well-nestedness and/or bounded gap degree only in terms of their order annotations. Sets that can be characterized in this way are said to be *algebraically transparent* (Kuhlmann, 2007).

3.2.3 Completeness

Proving completeness of the WG_1 parser is proving that all correct final items are valid. We will show this by proving the following, stronger claim:

Lemma 1. *Let T be a valid partial dependency tree headed at a node w_h. Then:*

- *(a) If $\lfloor w_h \rfloor = \{w_h\} \cup [i;j]$, then the item $[i;j;h;\diamond;\diamond]$ containing T is valid under this parser.*
- *(b) If $\lfloor w_h \rfloor = \{w_h\} \cup ([i;j] \setminus [l;r])$, then the item $[i;j;h;l;r]$ containing T is valid under this parser.*

It is clear that this lemma implies the completeness of the parser: a final item $[1;n;h;\diamond;\diamond]$ is correct only if it contains a tree rooted at w_h with gap degree ≤ 1 and projection $[1;n]$. Such a tree is in case (a) of Lemma 1, implying that the correct final item $[1;n;h;\diamond;\diamond]$ is valid. Therefore, this lemma implies that all correct final items are valid, and therefore that that WG_1 is complete.

3.2.4 Proof of Lemma 1

We will prove Lemma 1 by induction on $\#(\lfloor w_h \rfloor)$. In order to do this, we will show that Lemma 1 holds for valid trees T rooted at w_h such that $\#(\lfloor w_h \rfloor) = 1$, and then we will prove that if Lemma 1 holds for every valid tree T' such that $\#(\lfloor w_h \rfloor) < N$, then it also holds for all trees T such that $\#(\lfloor w_h \rfloor) = N$.

Base case Let T be a valid tree rooted also

c) =

Base case Let)< N, then it W item B049lds for Lf 18s8C 0 Td [(T)]TJ/F61

We know that $p \geq 1$ because if $\#(\lfloor w_h \rfloor) > 1$, then w_h must have at least one dependent. We now consider two cases: $p = 1$ and $p > 1$. In the case where $p = 1$, consider the subtree of T induced by w_{d_1}. Since $\#(\lfloor w_{d_1} \rfloor) = N - 1$, we know by induction hypothesis that the item corresponding to this tree is valid. This item is:

- $[i; j; d_1; \diamond; \diamond]$, if $\lfloor w_{d_1} \rfloor$ is of the form $\{w_{d_1}\} \cup [i; j]$, with $d_1 \in [i; j]$[5]. In this case, applying a *Link* step to this item and the initial item $[h; h; h; \diamond; \diamond]$ (which is valid by de nition), with the D-rule $w_{d_1} \rightarrow w_h$ (which must exist in order for T to be valid); we obtain $[i; j; h; \diamond; \diamond]$, which is the item corresponding to w_h by Lemma 1.

- $[i; j; d_1; h; h]$, if $\lfloor w_{d_1} \rfloor$ is of the form $\{w_{d_1}\} \cup ([i; j] \setminus \{w_h\})$. In this case, applying a *Link* step to this item and the initial item $[h; h; h; \diamond; \diamond]$ (which is valid by de nition), with the D-rule $w_{d_1} \rightarrow w_h$ (which must exist, as in the previous case); we obtain $[i; j; h; \diamond; \diamond]$[6], which is the item corresponding to w_h by Lemma 1.

- $[i; j; d_1; l; r]$, if $\lfloor w_{d_1} \rfloor$ is of the form $\{w_{d_1}\} \cup ([i; j] \setminus [l; r])$. In this case, applying a *Link* step to this item and the initial item $[h; h; h; \diamond; \diamond]$ (which is valid by de nition), with the D-rule $w_{d_1} \rightarrow w_h$; we obtain $[i; j; h; l; r]$; which is the item corresponding to w_h by Lemma 1.

With this, we have proven the induction step for the case where $p = 1$ (the head node of our partial dependency tree has a single direct child). It now remains to prove it for $p \geq 1$ (the head node has more than one direct dependent).

In order to show this, let $o(T; w_h)$ be the order annotation associated to the head node w_h in tree T. By construction, $O(T; w_h)$ must be a string of symbols in the alphabet $\{0\} \cup \{1\} \cup ::: \cup \{p\} \cup \{;\}$; containing a single appearance of the symbol 0. Additionally, by the de nition of of ofh

- (vi) $;0;$

Note that, by Property 2 of order annotations, the first case corresponds to a tree where the head has gap degree 0, in the next two cases the head has gap degree 1, and the last three are the cases where the gap degree of the head is 2: in these three latter cases, the constraint that $\lfloor w_h \rfloor$ must be of the form $\{w_h\} \cup ([i;j] \setminus [l;r])$ for the tree T to be valid implies that the symbol 0 representing the head in the annotation must be surrounded by commas: if we have a gap degree 2 annotation of any other form (for example $0;\ ;$, for nonempty); the projection of w_h does not meet this constraint. This can be seen by using Property 1 of order annotations to obtain this projection.

Taking these considerations into account, we will now divide the proof in different cases and subcases based on $o(T;w_h)$, starting with its first symbol:

1. If $o(T;w_h)$ begins with the symbol 1:
   a) If there are no more appearances of the symbol 1 in $o(T;w_h)$:

      Then we consider the following trees:

      - T_1: The tree obtained by taking the subtree induced by w_{d_1} (which by Property 1 must have a yield of the form $[i;j]$, as the symbol 1 appears only once in $o(T;w_h)$), and adding the node w_h and dependency $w_{d_1} \to w_h$ to it.
      - T_2: The tree obtained by taking the union of subtrees induced by $w_{d_2} \dots w_{d_p}$, and adding the node w_h and dependencies $w_{d_2} \to w_h; \dots; w_{d_p} \to w_h$ to it.

      And we divide this case into three further cases:

      i. If $o(T;w_h)$ does not contain any comma: Then, by Property 1[7], the projection of w_h in T_2 will be of the form $[j+1;k] \cup \{w_h\}$. By applying the induction hypothesis to T_1 and T_2, we know that the items $[i;j;h;\diamond;\diamond]$ and $[j+1;k;h;\diamond;\diamond]$ are valid. Therefore, the item $[i;k;h;\diamond;\diamond]$ is also valid because it can be obtained from these two items by applying a *Combine Ungapped* step. As in this case the projection of w_h in T is $[i;k] \cup [h]$, this item $[i;k;h;\diamond;\diamond]$ is the item containing the tree T, and its validity proves Lemma 1 in this particular subcase.

      ii. If $o(T;w_h)$ contains at least one comma, and the second symbol in $o(T;w_h)$ is a comma: Then $o(T;w_h)$ must be of the form (ii), (v) or (vi); and the projection of w_h in T_2 will be of the form $[i_2;k] \cup \{w_h\}$, for $i_2 > j+1$. Therefore, we know by the induction hypothesis that the items $[i;j;h;\diamond;\diamond]$ (for T_1) and $[i_2;k;h;\diamond;\diamond]$ (for T_2) are valid, and by applying *Combine Opening Gap* to these items, we obtain $[i;k;h;j+1;i_2-1]$, which is the item containing the tree T.

[7] In the remainder of the proof, we will always use Property 1 of order annotations to relate them to projections; so we will not mention it explicitly in subsequent cases.

iii. If $o(T; w_h)$ contains at least one comma, but the second symbol in $o(T; w_h)$ is not a comma:

A. First, in the case that $o(T; w_h)$ contains exactly one comma, then it is of the form $1\ {}_1;\ {}_2$, where either ${}_1$ or ${}_2$ contains the symbol 0 and neither of them is empty. In this case, we can see that the projection of w_h in T_2 is of the form $\{w_h\} \cup [j + 1; l - 1] \cup [r + 1; k]$, so by induction hypothesis the item $[j + 1; k; h; l; r]$ is valid. We apply *Combine Keeping Gap Right* to $[i; j; h; \diamond; \diamond]$ (which is valid by T_1 as in the previous cases) and $[j + 1; k; h; l; r]$ to obtain $[i; k; h; l; r]$, which is the item containing T.

B. Second, in the case where $o(T; w_h)$ contains two commas, then it is of the form $1\ {}_1; 0;\ {}_2$ or $1\ {}_1;\ {}_2; 0$. Then the projection of w_h in T_2 will again be of the form $\{w_h\} \cup [j + 1; l - 1] \cup [r + 1; k]$, so we can follow the same reasoning as in the previous case to show that the item $[i; k; h; l; r]$ containing T is valid.

b) If there is a second appearance of symbol 1 in $o(T; w_h)$: Then $o(T; w_h)$ is of the form $1\ {}_1 1\ {}_2$. Due to the well-nestedness constraint, we know that there is no symbol $s \in \{1\} \cup \{2\} \cup : : : \cup \{p\}$ that appears both in ${}_1$ and in ${}_2$. This allows us to consider the following trees:

- T_1: The tree obtained by taking the subtree induced by w_{d_1} (which must have a yield of the form $[i; l - 1] \cup [r + 1; j]$, as the symbol 1 appears twice in $o(T; w_h)$), and adding the node w_h and dependency $w_{d_1} \rightarrow w_h$ to it.
- T_2: The tree obtained by taking the union of subtrees induced by $w_{d_{b_1}} : : : w_{d_{b_q}}$, where $b_1 : : : b_q$ are the non-comma, non-zero symbols appearing in ${}_1$, and adding the node w_h and dependencies $w_{d_{b_1}} \rightarrow w_h; : : : ; w_{d_{b_q}} \rightarrow w_h$ to it.
- T_3: The tree obtained by taking the union of subtrees induced by $w_{d_{c_1}} : : : w_{d_{c_q}}$, where $c_1 : : : c_q$ are the non-comma, non-zero symbols appearing in ${}_2$, and adding the node w_h and dependencies $w_{d_{c_1}} \rightarrow w_h; : : : ; w_{d_{c_q}} \rightarrow w_h$ to it.

Note that T_2 or T

0912 Tf 65.422 0 Td [(f)]TJ/F11 10.9091 T8e6 2.795

A. If T_3 is empty ($_2$ is empty except for a possible 0 symbol), then we are done, as $[i; j; h; \diamond; \diamond]$ is already the item containing the tree T.

B. If $_2$ does not contain a comma, then the projection of w_h in T_3 is of the form $\{w_h\} \cup [j+1; k]$, so by induction hypothesis the item $[j+1; k; h; \diamond; \diamond]$ is valid. By applying *Combine Ungapped* to this item and , we obtain $[i; k; h; \diamond; \diamond]$, the item containing the tree T.

C. If $_2$ contains one or two commas, then the projection of w_h in T_3 is of the form $\{w_h\} \cup [j+1; l' - 1] \cup [r' + 1; m]$, and by induction hypothesis, $[j+1; k; h; l'; r']$ is valid. By applying *Combine Keeping Gap Right* to this item and , we get that $[i; k; h; l'; r']$ is valid, and this is the item containing the tree T in this case.

ii. If $_1$ contains a single symbol, and it is a comma: In this case, T_2 is empty, but we know that T_3 must be nonempty (since $p > 1$) and it must either have no commas, or be of the form $_3; 0$, corresponding to the expression (v). In any of these cases, we know that the projection of w_h in T_3 will be of the form $\{w_h\} \cup [j+1; k]$. Therefore, applying the induction hypothesis to T_1 we know that the item $[i; j; h; l; r]$ is valid, and with T_3 we know that $[j+1; k; h; \diamond; \diamond]$ is also valid. By applying the *Combine Keeping Gap Left* step to these two items, we obtain $[i; k; h; l; r]$, the item containing the tree T.

iii. If $_1$ is of the form \; $_3$", where $_3$ is not empty and does not contain commas: then, by construction and by the well-nestedness constraint, we know that the projection of w_h in T_2 is of the form $\{w_h\} \cup [l'; r]$, with $l < l' \leq r$; so the items $[i; j; h; l; r]$ (for T_1) and $[l'; r; h; \diamond; \diamond]$ (for T_2) are valid. By applying *Combine Shrinking Gap Right* to these two items, we obtain that $= [i; j; h; l; l' - 1]$ is a valid item. Now, if $_2$ is empty, we are done: is the item containing the tree T. And if $_2$ is nonempty, then it must either contain no commas, or be of the form $_4; 0$ (corresponding to the expression (v)). In any of these cases, we know that the projection of w_h in T_3 will be of the form $\{w_h\} \cup [j+1; k]$. So, by induction hypothesis, the item $[j+1; k; h; \diamond; \diamond]$ is valid; and by applying *Combine Keeping Gap Left* to and this item we obtain that $[i; k; h; l; l' - 1]$ is valid: this is the item containing the tree T in this case.

iv. If $_1$ is of the form \ $_3$;", where $_3$ is not empty and does not contain commas, this case is symmetric with respect to the last one: in this case, the projection of w_h in T_2 is of the form $\{w_h\} \cup [l; r']$, with $l \leq r' < r$; and the step *Combine Shrinking Gap Left* is step contain commas: in this case, by construction and by the well-nestedness constraint, we know that the projection of w_h in T_2 is of the form $\{w_h\} \cup [l; l'-1] \cup [r'+1; r]$, with $l < l' \leq r' < r$. With this, this case is analogous to the previous two cases: from T_1 we know that the item $[i; j; h; l; r]$ is valid, and we combine it with the item $[l; r; h; l'; r']$ (from T_2), in this case using *Combine Shrinking Gap Centre*. With this, we obtain that the item $= [i; j; h; l'; r']$ is valid. If ${}_2$ is empty, this is the item containing the tree T. If not, we make the same reasoning as in the two previous cases to conclude that the item $[j+1; k; h; \diamond; \diamond]$ is valid, and we combine it with by the *Combine Keeping Gap Left* step to obtain $[i; k; h; l'; r']$, the item containing T.

vi. If ${}_1$ contains two commas: in this case, by construction of the valid tree T, ${}_1$ must be of the form ${}_3; 0; {}_4$, where ${}_3$ and ${}_4$ may or may not be empty. So we divide into subcases:

   A. If ${}_3$ and ${}_4$ are both empty, we apply the same reasoning as in case 1-b-ii, except that in this case we know that ${}_2$ cannot contain any commas.

   B. If ${}_3$ is empty and ${}_4$ is nonempty, we apply the same reasoning as in case 1-b-iii, except that in this case we know that ${}_2$ cannot contain any commas.

   C. If ${}_3$ is nonempty and ${}_4$ is empty, we apply the same reasoning as in case 1-b-iv, except that in this case we know that ${}_2$ cannot contain any commas.

   D. If neither ${}_3$ nor ${}_4$ are empty, we apply the same reasoning as in case 1-b-v, except that in this case we know that ${}_2$ cannot contain any commas.

2. If $o(T; w_h)$ begins with the symbol 0:

   a) If $o(T; w_h)$ begins with 01, we can apply the same reasonings as in case 1, because the expressions for the projections do not change.

   b) If $o(T; w_h)$ begins with 0 followed immediately by a comma, then we have an annotation of the form (iv): $0; \; ; \;$. In this case, we can apply symmetric reasoning considering the last symbol of $o(T; w_h)$ instead of the rst (note that the case $\; ; \; ; 0$ has already been proven as part of case 1, and all the steps in the schema are symmetric).

As this covers all the possible cases of the order annotation $o(T; w_h)$, we have completed the proof of the induction step for Lemma 1, and this concludes the proof of completeness for the WG_1 parsing schema.

3.3 Computational complexity

The time complexity of WG_1 is $O(n^7)$, as the step *Combine Shrinking Gap Centre* works with 7 free string positions. This complexity with respect to the length of the input is as expected for this set of structures, since Kuhlmann (2007) shows that they are equivalent to LTAG, and the best existing parsers for this formalism also perform in $O(n^7)$ (Eisner and Satta, 2000). Note that the *Combine* step which is the bottleneck only uses the 7 indexes, and not any other entities like D-rules, so its $O(n^7)$ complexity does not have any additional factors due to grammar size or other variables. The space complexity of the parser is $O(n^5)$, due to the 5 indexes in items.

It is possible to build a variant of this parser with time complexity $O(n^6)$, as with parsers for unlexicalised TAG, if we work with unlexicalised D-rules specifying the possibility of dependencies between pairs of categories instead of pairs of words. In order to do this, we expand the item set with unlexicalised items of the form [

An item $[i; j; h; [(l_1; r_1); \dots; (l_g; r_g)]]$ represents the set of all well-nested partial dependency trees rooted at w_h such that $\lfloor w_h \rfloor = \{w_h\} \cup ([i; j] \setminus \bigcup_{p=1}^{g} [l_p; r$

As expected, the WG_1 parser corresponds to WG_k when we make $k = 1$. WG_k works in the same way as WG_1, except for the fact that *Combine* steps can create items with more than one gap.

4.2 Proof of correctness for WG_k

The proof of correctness for WG_k is analogous to that of WG_1, but generalising the de nition of valid trees to a higher gap degree. A valid tree in WG_k can be de ned as a partial dependency tree T, headed at w_h, such that

- (1) $\lfloor w_h \rfloor$ is of the form $\{w_h\} \cup ([i; j] \setminus \bigcup_{p=1}^{g} [l_p; r_p])$, with $0 \leq g \leq k$,
- (2) All the nodes in T have gap degree at most k except for w_h, which can have gap degree up to $k + 1$.

With this, we can de ne correct items and correct nal items analogously to their de nition in WG_1.

Soundness is proven as in WG_1: changing the constraints for nodes so that any node can have gap degree up to k and the head of a correct tree can have gap degree $k + 1$, the same reasonings can be applied to this case.

Completeness is proven by induction on $\#(\lfloor w_h \rfloor)$, just as in WG_1. The base case is the same as in WG_1, and for the induction step, we also consider the direct children $w_{d_1} \dots w_{d_p}$ in w_h. The case where $p = 1$ is proven by using *Linker* steps just as in WG_1. In the case for $p \geq 1$, we also base our proof in the order annotation $o(T; w_h)$, but we have to take into account that the set of possible annotations is larger when we allow the gap degree to be greater than 1, so we must take into account more cases in this part of the proof.

In particular, an order annotation $o(T; w_h)$ for a valid tree for WG_k can contain up to $k + 1$ commas and up to $k + 1$ appearances of each symbol in $\{1\} \cup \dots \cup \{p\}$; since the head of such a tree can have gap degree at most $k + 1$ and the rest of its nodes are limited to gap degree k. If the head has gap degree exactly $k + 1$ (i.e., if $o(T; w_h)$ contains k

same way, the cases in which we used *Combine Keeping Gap* steps in the proof for WG_1 are solved by using the general *Combine Keeping Gap* step in WG_k.

4.3 Computational complexity

The WG_k parser runs in time $O(n^{5+2k})$: as in the case of WG_1, the deduction step with most free variables is *Combine Shrinking Gap Centre*, and in this case it has $5 + 2k$ free indexes. Again, this complexity result is in line with what could be expected from previous research in constituency parsing: Kuhlmann (2007) shows that the set of well-nested dependency structures with gap degree at most k is closely related to coupled context-free grammars in which the maximal rank of a nonterminal is $k + 1$; and the constituency parser de ned by Hotz and Pitsch (1996) for these grammars also adds an n^2 factor for each unit increment of k. Note that a small value of k should be enough to cover the vast majority of the non-projective sentences found in natural language treebanks. For example, the Prague Dependency Treebank contains no structures with gap degree greater than 4. Therefore, a WG_4 parser would be able to analyse all the well-nested structures in this treebank, which represent 99.89% of the total. Increasing k beyond 4 would not produce further improvements in coverage.

5 Parsing ill-nested structures

The WG_k parser analyses dependency structures with bounded gap degree as long as they are well-nested. This covers the vast majority of the structures that occur in natural-language treebanks (Kuhlmann and Nivre, 2006), but there is still a signi cant minority of sentences that contain ill-nested structures. Unfortunately, the general problem of parsing ill-nested structures is NP-complete, even when the gap degree is bounded: this set of structures is closely related to LCFRS with bounded fan-out and unbounded production length, and parsing in this formalism has been proven to be NP-complete (Satta, 1992). The reason for this high complexity is the problem of *unrestricted crossing con gurations*, appearing when dependency subtrees are allowed to interleave in every possible way. However, just as it has been noted that most non-projective structures appearing in practice are only \slightly" non-projective (Nivre and Nilsson, 2005), we characterise a sense in which the structures appearing in treebanks can be viewed as being only \slightly" ill-nested. In this section, we generalise the algorithms WG_1 and WG_k to parse a proper superset of the set of well-nested structures in polynomial time; and give a characterisation of this new set of structures, which includes all the structures in several dependency treebanks.

5.1 The MG_1 and MG_k parsers

The WG_k parser for well-nested structures presented previously is based on a bottom-up process, where *Link* steps are used to link completed subtrees to a head, and *Combine* steps are used to join subtrees governed by a common head to obtain a larger structure. As WG_k is a parser for well-nested structures of gap degree up to k, its *Combiner* steps correspond to all the ways in which we can join two sets of sibling subtrees meeting these constraints, and having a common head, into another. Therefore, this parser does not use *Combiner* steps that produce interleaved subtrees, since these would generate items corresponding to ill-nested structures.

We obtain a polynomial parser for a wider set of structures of gap degree at most k, including some ill-nested ones, by having *Combiner* steps representing every way in which two sets of sibling subtrees of gap degree at most k with a common head can be joined into another, including those producing interleaved subtrees, like the steps for gap degree 1 shown in Figure 1. Note that this does not mean that we can build every possible ill-nested structure: some structures with complex crossed configurations have gap degree k, but cannot be built by combining two structures of that gap degree. More specifically, our algorithm will be able to parse a dependency structure (well-nested or not) if there exists a *binarisation* of that structure that has gap degree at most k. The

Combine Interleaving: $\dfrac{[i; j; h; l; r] \quad [l; k; h; r+1; j]}{[i; k; h; \diamond; \diamond]}$

Combine Interleaving Gap C: $\dfrac{[i; j; h; l; r] \quad [l; k; h; m; j]}{[i; k; h; m; r]}$ such that $m < r + 1$,

Combine Interleaving Gap L: $\dfrac{[i; j; h; l; r] \quad [l; k; h; r+1; u]}{[i; k; h; j+1; u]}$ such that $u > j$,

Combine Interleaving Gap R: $\dfrac{[i; j; h; l; r] \quad [k; m; h; r+1; j]}{[i; m; h; l; k-1]}$ such that $k > l$.

Figure 1: Additional steps to turn WG_1 into MG_1.

$$\frac{[i_{a_1}; i_{a_p+1} - 1; h; [(i_{a_1+1}; i_{a_2} - 1); \ldots; (i_{a_{p-1}+1}; i_{a_p} - 1)]] \quad [i_{b_1}; i_{b_q+1} - 1; h; [(i_{b_1+1}; i_{b_2} - 1); \ldots; (i_{b_{q-1}+1}; i_{b_q} - 1)]]}{[i_{min(a_1; b_1)}; i_{max(a_p+1; b_q+1)} - 1; h; [(i_{g_1}; i_{g_1+1} - 1); \ldots; (i_{g_r}; i_{g_r+1} - 1)]]}$$

for each string of length n with a's located at positions $a_1 \ldots a_p (1 \leq a_1 < \ldots < a_p \leq n)$, b's at positions $b_1 \ldots b_q (1 \leq b_1 < \ldots < b_q \leq n)$, and g's at positions $g_1 \ldots g_r (2 \leq g_1 < \ldots < g_r \leq n-1)$, such that $1 \leq p \leq k$, $1 \leq q \leq k$, $0 \leq r \leq k-1$, $p + q + r = n$, and the string does not contain more than one consecutive appearance of the same symbol.

Figure 2: Gen0k

In order to generalise this algorithm to mildly ill-nested structures for gap degree k, we need to add a *Combine* step for every possible way of joining two structures of gap degree at most k into another. This can be done in a systematic way by considering a set of strings over an alphabet of three symbols: a and b to represent intervals of words in the projection of each of the structures, and g to represent intervals that are not in the projection of either of the structures, and will correspond tocoln dependent of w_d in T. These properties of binarisations will be used throughout the proof.

As for the previous algorithms, we will start the proof by de ning the sets of valid trees and correct items for this algorithm, which we will use to prove soundness and completeness.

Let T be a partial dependency tree headed at a node w_h. We will call such a tree a *valid tree* for the algorithm WG_k if it satis es the following:

- (1) $\lfloor w_h \rfloor$ is of the form $\{w_h\} \cup ([i; j] \setminus \bigcup_{p=1}^{g} [l_p; r_p])$, with $0 \leq g \leq k$,
- (2) There exists a binarisation of T such that all the nodes in it have gap degree at most k except for its root node, which can have gap degree up to $k + 1$.

Note that, since by property (ii) a binarisation cannot decrease the gap degree of a tree, condition (2) implies that all the nodes in T must have gap degree at most k except for w_h, which can have gap degree at most $k + 1$.

That is, the de nition of a valid tree in this case is as in WG_k, but changing the well-nestedness constraint to the weaker requirement of having a binarisation of gap degree k (except for the particular case of the root node, which can have gap degree $k + 1$). As in WG_1 and WG_k, we will say that an item is *correct* if it contains some valid tree T licensed by a set of D-rules G, and throughout the proof we will suppose that all items are normalised.

Given an input string $w_1 \dots w_n$, a correct nal item for MG_k will have the form $[1; n; h; []]$, and contain at least one valid tree T rooted at a head w_h and with $\lfloor w_h \rfloor = [1; n]$, which is a complete parse for the input. Since in a tree contained in an item of this form the projection of the head cannot have any gaps and thus the head has gap degree 0, we have that there exists a binarisation of T such that every one of its nodes, including the head, has gap degree at most k. Therefore, T is mildly ill-nested for gap degree k and, more generally, nal items in MG_k only contain mildly ill-nested trees for gap degree k, as expected.

and linking the head of the antecedent tree to it, for *Link* steps, and by considering the union of the trees corresponding to the antecedents, for *Combine* steps.

We can show that the resulting tree is licensed by G and that it satisfies the condition (1) of a valid tree in the same way as we did in WG_1 and WG_k. So, to prove soundness, it only remains to show that the resulting tree has a binarisation verifying the gap degree constraint (2).

To prove this, we show that a binarisation satisfying (2) of the tree corresponding to the consequent item can be constructed from the corresponding binarisations of the antecedent items. We will prove the stronger claim that such a binarisation can be constructed, with the additional constraints that: (3) its root node must be labelled (therefore, by one of the properties of binarisations, its label corresponds to the head node of the original tree) and can have at most one direct child, and that (4) the binarisation can only contain more than one node labelled w_h if the item is of the form $[i, j, h, [(l_1, r_1) \dots (l_g, r_g)]]$ such that $w_h \in ([i, j] \setminus \bigcup_{p=1}^{g} [l_p, r_p])$.

In the case of each *Link* step adding a link $w_d \to w_h$, such a binarisation can be constructed by taking the binarisation B_a corresponding to the non-initial antecedent item, and linking its head to a new node labelled w_h. The resulting tree is a binarisation of the consequent tree, and it satisfies (2) because the head can have gap degree at most $k+1$ (by construction of the antecedents of *Link* steps, the antecedent item must have a

Proposition 1. *Let T be a partial dependency tree headed at node w_h, and valid for MG_k. Then, if $\lfloor w_h \rfloor = \{w_h\} \cup ([i, j] \setminus \bigcup_{p=1}^{g} [l_p, r_p])$, for $p \leq k$, the item $[i, j, h, (l_1, r_1), \ldots, (l_g, r_g)]$ containing T is valid under this parser.*

It is clear that this proposition implies the completeness of the parser: a final item $[1, n, h, []]$ is correct only if it contains a tree rooted at w_h, valid for MG_k and with projection $\lfloor w_h \rfloor = [1, n]$. By Proposition 1, having such a tree implies that the correct final item $[1, n, h, []]$ is valid. Therefore, this lemma implies that all correct final items are valid, and thus that MG_k is complete.

Since valid trees for the MG_k parser must be mildly ill-nested for gap degree k, every valid tree must have at least one binarisation where every node has gap degree $\leq k$ except possibly the head, that can have gap degree $k + 1$. We will call a binarisation satisfying this property a well-formed binarisation for MG_k.

Using this, we can prove Proposition 1 if we prove the following lemma:

Lemma 2. *Let B be a well-formed binarisation of a partial dependency tree T, headed at node w_h and valid for MG_k. If the projection of w_h in T is $\lfloor w_h \rfloor_T = \lfloor w_h \rfloor_B = \{w_h\} \cup ([i, j] \setminus \bigcup_{p=1}^{g} [l_p, r_p])$, for $p \leq k$, the item $[i, j, h, (l_1, r_1), \ldots, (l_g, r_g)]$ containing T is valid under this parser.*

5.3.3 Proof of Lemma 2

We will prove this lemma by induction on the number of nodes of B (denoted $\#B$). In order to do this, we will show that Lemma 2 holds for well-formed binarisations B of trees T rooted at w_h such that $\#B = 1$, and then we will prove that if Lemma 2 holds for every well-formed binarisation B' such that $\#B' < N$, then it also holds for binarisations B such that $\#B = N$.

Base case Let B be a well-formed binarisation of a partial dependency tree T, rooted at a node w_h and valid for MG_k, and such that $\#B = 1$. In this case, since B has only one node, it must be a binarisation of the trivial dependency tree consisting of the single node w_h. Thus, Lemma 2 trivially holds because the initial item $[h, h, h, []]$ contains this tree, and initial items are valid by definition.

Induction step Let B be a well-formed binarisation of some partial dependency tree T, headed at node w_h and valid for MG_k, such that $\lfloor w_h \rfloor_T = \{w_h\} \cup ([i, j] \setminus \bigcup_{p=1}^{g} [l_p, r_p])$, and $\#B = N$; and suppose that Lemma 2 holds for every well-formed binarisation B' of a tree T' such that $\#B' < N$. We will prove that Lemma 2 holds for B.

In order to do this, we consider different cases depending on the number and type of children of the head node labelled w_h in B:

- If w_h has a single child in B, and it is a node labelled w_d ($w_d \neq w_h$): then, the subtree B' induced by w_d in B is a binarisation of some tree T', such that $\lfloor w_d \rfloor_{T'} = \lfloor w_h \rfloor_T \setminus \{w_h\}$ (note that no nodes labelled w_h can appear in B', since w_h cannot be a dependent of w_d). As $\#B' < N$ and B' is well-formed because all its nodes are non-head nodes of B; by applying the induction hypothesis, we obtain that the item $\quad = [i; j; d; (l_1; r_1); : : : ; (l_g; r_g)]$ (which contains T^0 by construction) is valid. The item $[i; j; h; (l_1; r_1); : : : ; (l_g; r_g)]$ containing T can be obtained from and the initial item $[h; h; h; ()]$ by a *Link* step, and therefore it is valid, so we have proven Lemma 2 in this case.

- If w_h has a single child in B, and it is an unlabelled node: call this unlabelled node n. Then, the subtree B^0 obtained from removing n from B and linking its children directly to w_h is a binarisation of the same tree as B. We know that B^0 is well-formed because its non-head nodes have the same projections as in B and therefore must have gap degree $\leq k$ and, as B is well-formed, n has gap degree $\leq k$, so the subtree created by linking the children of n to w_h can have gap degree at most $k + 1$, and it only will have degree $k + 1$ if $\lfloor w_h \rfloor_{B'} \setminus \{w_h\}$ has k gaps. As B and B^0 are well-formed binarisations of the same tree, if Lemma 2 holds for B^0, it also must hold for B. As we know that $\# B^0 < N$ (since it contains one less node than B), Lemma 2 holds for B^0 by the induction hypothesis, so this case is proven.

- If w_h has a single child in B, and it is a node labelled w_h: then, the subtree B^0 induced by this single child node is a binarisation of the same tree as B. We know that B^0 is well-formed because its nodes have the same projections as they had in B, and therefore they must all have gap degree $\leq k$ by the well-formedness of B. Reasoning as in the previous case, since B and B^0 are binarisations of the same tree and we know that Lemma 2 holds for B^0 for the induction hypothesis, this implies that it holds for B as well.

- If w_h has two children in B: in this case, regardless of whether the direct children of w_h are labelled or unlabelled nodes, we call them c_1 and c_2 and consider two partial dependency trees B^0_1 and B^0_2:

  { B^0_1 is the tree obtained by taking the subtree induced by c_1 and linking its head c_1 to w_h,

  { B^0_2 is the tree obtained by taking the subtree induced by c_2 and linking its head c_2 to w_h.

  We know that all the nodes in B^0_1 and B^0_2, except for the head, must have gap degree $\leq k$ because their projection in B^0_1 and B^0_2 is the same as their projection in B, which is a well-formed binarisation. We know that w_h must have degree $\leq k + 1$ in B^0_1 and B^0_2 because, by construction, $\lfloor w_h \rfloor_{B'_1}$

Language	Structures								
	Total	Nonprojective							
		Total	By gap degree				By nestedness		
			Gap deg. 1	Gap deg. 2	Gap deg. 3	Gap d. > 3	Well-Nested	Mildly Ill-Nest.	Strongly Ill-Nest.
Arabic	2995	205	189	13	2	1	204	1	0
Czech	87889	20353	19989	359	4	1	20257	96	0
Danish	5430	864	854	10	0	0	856	8	0
Dutch	13349	4865	4425	427	13	0	4850	15	0
Latin	3473	1743	1543	188	10	2	1552	191	0
Portuguese	9071	1718	1302	351	51	14	1711	7	0
Slovene	1998	555	443	81	21	10	550	5	0
Swedish	11042	1079	1048	19	7	5	1008	71	0
Turkish	5583	685	656	29	0	0	665	20	0

Table 1: Counts of dependency trees classi ed by gap degree, and mild and strong ill-nestedness (for their gap degree); appearing in treebanks for Arabic (Hajic et al., 2004), Czech (Hajic et al., 2006), Danish (Kromann, 2003), Dutch (van der Beek et al., 2002), Latin (Bamman and Crane, 2006), Portuguese (Afonso et al., 2002), Slovene (Dzeroski et al., 2006), Swedish (Nilsson et al., 2005) and Turkish (O azer et al., 2003; Atalay et al., 2003).

for $g_1; g_2 \leq k + 1$. We also know that the union of the projections of w_h in T_1^{0} and T_2^{0} is the union of $g_c \leq k + 1$ intervals, and is the same as the projection of w_h in T. Therefore, as the indexes of the *Combiner* steps in MG_k

0 1 2 3 4 5 6 7 8 9

Figure 3: One of the smallest strongly ill-nested structures. This dependency structure has gap degree 1, but is only mildly ill-nested for gap degree ≥ 2.

Even if a structure T is strongly ill-nested for a given gap degree, there is always some $m \in \mathbb{N}$ such that T is mildly ill-nested for m (since every dependency structure can to the way the MG_k parser works, since it implicitly nds such a binarisation. An inter-

Carlos Gomez-Rodr guez, John Carroll, and David Weir. A deductive approach to dependency parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL'08:HLT)*, pages 968{976. Association for Computational Linguistics, 2008.

Jan Hajic, Otakar Smrz, Petr Zemanek, Jan Snaidauf, and Emanuel Beska. Prague Arabic dependency treebank: Development in data and tools. In *Proceedings of the NEMLAR International Conference on Arabic Language Resources and Tools*, pages 110{117, 2004.

Jan Hajic, Jarmila Panevova, Eva Hajicova, Jarmila Panevova, Petr Sgall, Petr Pajas, Jan Stepanek, Jir Havelka, and Marie Mikulova. Prague dependency treebank 2.0 (ldc2006t01). CDROM CAT: LDC2006T01., ISBN 1-58563-370-4, 2006.

Jir Havelka. Beyond projectivity: Multilingual evaluation of constraints and measures on non-projective structures. In *ACL 2007: Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, 2007.

Gunter Hotz and Gisela Pitsch. On parsing coupled-context-free languages. *Theor. Comput. Sci.*, 161(1-2):205{233, 1996. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/0304-3975(95)00114-X.

Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars, 1997.

Matthias T. Kromann. The danish dependency treebank and the underlying linguistic theory. In *Proceedings of the 2nd Workshop on Treebanks and Linguistic Theories (TLT)*, 2003.

Marco Kuhlmann. *Dependency Structures and Lexicalized Grammars*. Doctoral dissertation, Saarland University, Saarbrucken, Germany, 2007.

Marco Kuhlmann and Mathias Mohl. Mildly context-sensitive dependency languages. In *45th Annual Meeting of the Association for Computational Linguistics (ACL)*, Prague, Czech Republic, 2007.

Marco Kuhlmann and Joakim Nivre. Mildly non-projective dependency structures. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 507{514, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

Ryan McDonald and Giorgio Satta. On the complexity of nonprojective data-driven dependency parsing. In *IWPT 2007: Proceedings of the 10th Conference on Parsing Technologies*. Association for Computational Linguistics, 2007.

Ryan McDonald, Fernando Pereira, Kiri295(r-422(Ki1(c)51(R8)28(ereira,)-422(Kd4fs0y)28(an)-44rkd [(Cz