Parsing Mildly Non-projective Dependency Structures*

Carlos Gómez-Rodríguez†
Departamento de Computación
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

Dependency parsers analyse a sentence in terms of a set of directed links (dependencies) expressing the head-modifier and head-complement relationships which form the basis

*This report is an extended version of the homonymous paper to be published in the proceedings of EACL-2009, including the proofs that could not be included in the paper due to space limitations.

†Partially supported by Ministerio de Educación y Ciencia and FEDER (HUM2007- 66607-C04) and Xunta de Galicia (PGIDIT07SIN005206PR, INCITE08E1R104022ES, INCITE08ENA305025ES, INCITE08PXIB302179PR and Rede Galega de Procesamento da Linguaxe e Recuperación de Información)

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 efficiency, 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. Specifically, 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 efficient 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 define 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 $V = \{w_1, \dots, w_n\}$ and $E \subseteq V \times V$. We write the edge (w_i, w_j) as $w_i \rightarrow w_j$, meaning that the word w_i is a syntactic *dependent* (or a *child*) of w_j or, conversely, that w_j is the *governor* (*parent*) of w_i. We write $w_i \rightarrow^\star w_j$ to denote that there exists a (possibly empty) path from w_i to w_j. The *projection* of a node w_i, denoted $\lfloor w_i \rfloor$, is the set of reflexive-transitive dependents of w_i, that is: $\lfloor w_i \rfloor = \{w_j \in V \mid w_j \rightarrow^\star w_i\}$. In contexts where we refer to different graphs that may share nodes, we will use the notation $\lfloor w_i \rfloor_G$ to denote the projection of a node w_i in the graph G. An *interval* (with endpoints i and j) is a set of the form $[i, j] = \{w_k \mid i \leq k \leq j\}$. We will denote the cardinality of a set S as $\#(S)$.

A dependency graph is said to be a *tree* if it is:

1. acyclic: $w_j \in \lfloor w_i \rfloor$ implies $w_i \rightarrow w_j \notin E$; and
2. each node has exactly one parent, except for one node which we call the *root* or *head*.

A graph verifying these conditions and having a vertex set $V \subseteq \{w_1, \dots, w_n\}$ is a *partial dependency tree*. Given a dependency tree $T = (V, E)$ and a node $u \in V$, the *subtree* induced by the node u is the graph $T_u = (\lfloor u \rfloor, E_u)$ where $E_u = \{w_i \rightarrow w_j \in E \mid w_j \in \lfloor u \rfloor\}$.

2.1 Properties of dependency trees

We now define the concepts of gap degree and well-nestedness (Kuhlmann and Nivre, 2006).

Let T be a (possibly partial) dependency tree for $w_1 \dots w_n$: We say that T is **projective** if $\lfloor w_i \rfloor$ is an interval for every word w_i. Thus every node in the dependency structure must dominate a contiguous substring in the sentence.

The **gap degree** of a particular node w_k in T is the minimum $g \in \mathbb{N}$ such that $\lfloor w_k \rfloor$ can be written as the union of $g + 1$ intervals; that is, the number of discontinuities in $\lfloor w_k \rfloor$. The gap degree of the dependency tree T is the maximum among the gap degrees of its nodes. Note that T has gap degree 0 if and only if T is projective.

The subtrees induced by nodes w_p and w_q are **interleaved** if $\lfloor w_p \rfloor \cap \lfloor w_q \rfloor = \emptyset$ and there are nodes $w_i, w_j \in \lfloor w_p \rfloor$ and $w_k, w_l \in \lfloor w_q \rfloor$ such that $i < k < j < l$. A dependency tree T is **well-nested** if it does not contain two interleaved subtrees. A tree that is not

[1] Parsing unrestricted ill-nested structures, even when the gap degree is bounded, is NP-complete: these structures are equivalent to LCFRS for which the recognition problem is NP-complete (Satta, 1992)

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 defined for constituency-based grammatical formalisms, but Gómez-Rodríguez et al. (2008) define a variant of the framework for dependency-based parsers. We use these *dependency parsing schemata* to define 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* which produce new items from existing ones. Each item contains information about the sentence's structure, and a successful parsing process produces at least one *final item* providing a full dependency analysis for the sentence or guaranteeing its existence.

In a dependency parsing schema, items are defined as sets of partial dependency trees[2]. To define a parser by means of a schema, we must define an item set and provide a set of deduction steps that operate on it. Given an item set $\mathcal{I}$, the set of *final items* for strings of length n is the set of items in $\mathcal{I}$ that contain a full dependency tree for some arbitrary string of length n. A final item containing a dependency tree for a particular string $w_1 \dots w_n$ is said to be a *correct final item* for that string. These concepts can be used to prove the correctness of a parser: for each input string, a parsing schema's deduction steps allow us to infer a set of items, called *valid items* for that string. A schema is said to be *sound* if all valid final items it produces for any arbitrary string are correct for that string. A schema is said to be *complete* if all correct final items are valid. A *correct* parsing schema is one which is both sound and complete.

In constituency-based parsing schemata, deduction steps usually have grammar rules as side conditions. In the case of dependency parsers it is also possible to use grammars (Eisner and Satta, 1999), but many algorithms use a data-driven approach instead, making individual decisions about which dependencies to create by using probabilistic models (Eisner, 1996) or classifiers (Yamada and Matsumoto, 2003). To represent these algorithms as deduction systems, we use the notion of *D-rules* (Covington, 1990). D-rules take the form $a \rightarrow b$, which says that word b can have a as a dependent. Deduction steps in non-grammar-based parsers can be tied to the D-rules associated with the links they create. In this way, we obtain a representation of the underlying logic of the parser while abstracting away from control structures (the particular model used to create the decisions associated with D-rules). Furthermore, the choice points in the parsing process and the information we can use to make decisions are made explicit in the steps linked to D-rules.

[2]The formalism allows items to contain forests, and the dependency structures inside items are defined in a notation with terminal and preterminal nodes, but these are not needed here.

3 The WG_1 parser

3.1 Parsing schema for WG_1

We define 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 different sets of values of i,j,l,r, we restrict these values so that we cannot get two different items representing the same dependency structures. Items ι violating these constraints always have an alternative representation that does not violate them, that we can express with a normalising function $nm(\iota)$ as follows:

$nm([i,j,j,l,r]) = [i,j-1,j,l,r]$ (if $r \leq j-1$ or $r = \diamond$),
or $[i,l-1,j,\diamond,\diamond]$ (if $r = j-1$).
$nm([i,j,l-1,l,r]) = [i,j,l-1,l-1,r]$(if $l > i+1$),
or $[r+1,j,l-1,\diamond,\diamond]$ (if $l = i+1$).
$nm([i,j,i-1,l,r]) = [i-1,j,i-1,l,r]$.
$nm([i,j,r,l,r]) = [i,j,r,l,r-1]$ (if $l < r$),
or $[i,j,r,\diamond,\diamond]$ (if $l = r$).
$nm([i,j,h,l,r]) = [i,j,h,l,r]$ for all other items.

When defining the deduction steps for this and other parsers, we assume that they always produce normalised items. For clarity, we do not explicitly write this in the deduction steps, writing ι instead of $nm(\iota)$ as antecedents and consequents of steps.

The set of initial items is defined as the set

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

where each item $[h,h,h,\diamond,\diamond]$ represents the set containing the trivial partial dependency tree consisting of a single node w_h and no links. This same set of hypotheses can be

[3] In this and subsequent schemata, we use D-rules to express parsing decisions, so partial dependency trees are assumed to be taken from the set of trees licensed by a set of D-rules.

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 $w_{h2} \in [i2, j2] \wedge w_{h1} \notin [i2, j2]$,

Link Gapped:
$$\frac{[h1, h1, h1, \diamond, \diamond] \quad [i2, j2, h2, l2, r2]}{[i2, j2, h1, l2, r2]} \; w_{h2} \rightarrow w_{h1}$$
such that $w_{h2} \in [i2, j2] \setminus [l2, r2] \wedge w_{h1} \notin [i2, j2] \setminus [l2, r2]$,

Combine Ungapped:
$$\frac{[i, j, h, \diamond, \diamond] \quad [j+1, k, h, \diamond, \diamond]}{[i, k, h, \diamond, \diamond]}$$

Combine Opening Gap:
$$\frac{[i, j, h, \diamond, \diamond] \quad [k, l, h, \diamond, \diamond]}{[i, l, h, j+1, k-1]}$$
such that $j < k - 1$,

Combine Keeping Gap Left:
$$\frac{[i, j, h, l, r] \quad [j+1, k, h, \diamond, \diamond]}{[i, k, h, l, r]}$$

Combine Keeping Gap Right:
$$\frac{[i, j, h, \diamond, \diamond] \quad [j+1, k, h, l, r]}{[i, k, h, l, r]}$$

Combine Closing Gap:
$$\frac{[i, j, h, l, r] \quad [l, r, h, \diamond, \diamond]}{[i, j, h, \diamond, \diamond]}$$

Combine Shrinking Gap Centre:
$$\frac{[i, j, h, l, r] \quad [l, r, h, l2, r2]}{[i, j, h, l2, r2]}$$

Combine Shrinking Gap Left:
$$\frac{[i, j, h, l, r] \quad [l, k, h, \diamond, \diamond]}{[i, j, h, k+1, r]}$$

Combine Shrinking Gap Right:
$$\frac{[i, j, h, l, r] \quad [k, r, h, \diamond, \diamond]}{[i, j, h, l, k-1]}$$

The WG_1 parser proceeds bottom-up, by building dependency subtrees and joining them to form larger subtrees, until it finds a complete dependency tree for the input sentence. The logic of the parser can be understood by considering how it infers the item corresponding to the subtree induced by a particular node, given the items for the subtrees induced by the direct dependents of that node. Suppose that, in a complete dependency analysis for a sentence $w_1 \ldots w_n$, the word w_h has $w_{d_1} \ldots w_{d_p}$ as direct dependents (i.e. we have dependency links $w_{d_1} \rightarrow w_h, \ldots, w_{d_p} \rightarrow w_h$). Then, the item corresponding to the subtree induced by w_h is obtained from the ones corresponding to the subtrees induced by $w_{d_1} \ldots w_{d_p}$ by: (1) applying the *Link Ungapped* or *Link Gapped* step to each of the items corresponding to the subtrees induced by the direct dependents, and to the hypothesis $[h, h, h, \diamond, \diamond]$. This allows us to infer p items representing the result

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 parsing process is successful, a final item is found containing a dependency tree for the complete sentence.

3.2 Proof of correctness for WG_1

The parsing schemata formalism can be used to prove the correctness of a parser. To prove that a schema is correct, we need to prove its soundness (all valid final items are correct) and completeness (all correct final items are valid). This is usually done by defining a set of *correct* items for the schema, in such a way that final items in this set are correct final items by the general definition given in Section 2.2; and then proving the stronger claims that all valid items are correct and all correct items are valid. All the correctness proofs in this report follow this general method.

To define the set of correct items for WG_1, we will first provide a definition of the trees that these items must contain: let T be a well-nested partial dependency tree headed at a node w_h. We will call such a tree a *valid tree* for the algorithm WG_1 if it satisfies the following conditions:

- (1) $\lfloor w_h \rfloor$ is either of the form $\{w_h\} \cup [i, j]$ or $\{w_h\} \cup ([i, j] \setminus [l, r])$.
- (2) All the nodes in T have gap degree at most 1 except for w_h, which can have gap degree up to 2.

Given an input string $w_1 \dots w_n$ and a set of D-rules G, we say that an item of the form $[i, j, h, \diamond, \diamond] \in \mathcal{I}_{WG1}$ is *correct* if it contains a valid tree T rooted at w_h, such that $\lfloor w_h \rfloor = \{w_h\} \cup [i, j]$, and all the edges in T are licensed by G.

We say that an item of the form $[i, j, h, l, r] \in \mathcal{I}_{WG1}$ is *correct* if it contains a valid tree T headed at w_h, such that $\lfloor w_h \rfloor = \{w_h\} \cup ([i, j] \setminus [l, r])$, and all the edges in T are licensed by G.

Throughout the proof we will suppose that all items are normalised, that is, $[i, j, h, l, r]$ should always be read as $nm([i, j, h, l, r])$, although we will omit the nm function most of the time for clarity.

Since a final item in WG_1 has the form $[1, n, h, \diamond, \diamond]$, a *correct final item* for this algorithm will contain at least one valid tree rooted at a head w_h and with $\lfloor w_h \rfloor = [1, n]$. This tree must be well-nested because it is valid, and must have gap degree ≤ 1 because the definition of a valid tree implies that every node except for w_h has gap degree ≤ 1, and the fact that $\lfloor w_h \rfloor = [1, n]$ implies that w_h has gap degree 0. Therefore, a correct final item for an input string contains at least one well-nested parse of gap degree ≤ 1 for that string.

Proving the correctness of the WG_1 parser amounts to proving its soundness and completeness.

3.2.1 Soundness

Proving the soundness of the WG_1 parser is showing that all valid final items (that is, items that can be obtained from the hypotheses by applying some sequence of deduction steps) are correct.

We will do this by proving the stronger claim that all valid items are correct. Since all valid items are either hypotheses or obtained by applying a deduction step to other valid items, it suffices to show that (i) hypotheses are correct, and (ii) if the antecedents of a deduction step in WG_1 are correct, then the consequent is also correct.

(i) is trivial, since any hypothesis $[h, h, h, \diamond, \diamond]$ contains the valid dependency tree consisting of a single node (w_h) and no links.

In order to prove (ii), given a set of D-rules G, we must prove that if the antecedents of a deduction step are items containing a valid tree for WG_1 licensed by the D-rules in G, then the consequent must also contain a valid tree for WG_1 licensed by G. In order for a tree to be valid, it must be well-nested, with $\lfloor w_h \rfloor$ of the form $\{w_h\} \cup [i, j]$ or $\{w_h\} \cup ([i, j] \setminus [l, r])$, and with all the nodes having gap degree at most 1 except for the head, which may have gap degree up to 2.

By definition of items in WG_1, all trees contained in items must verify the conditions of a valid tree. Therefore, proving soundness in this case amounts to proving that if the antecedents of a step are nonempty, then the consequent is nonempty.

This can be seen step by step: in the case of a *Link Gapped* step creating a dependency $w_{h2} \rightarrow w_{h1}$, a tree T_c for the consequent item can be obtained from a tree T_a taken from the second antecedent ([i2,j2,h2,l2,r2]) by linking its head (which is w_{h2} by construction of the antecedent) to w_{h1}. Condition (1) of a valid tree is satisfied by construction, since the projection of the head of T_c is the result of adding w_{h1} to the projection of w_{h2} in T_a, and this projection is of the form $[i2, j2] \setminus [l2, r2]$ by construction of the antecedent and by the constraint imposed by the step on w_{h2}. Besides, since by this same constraint we know that T_a must have gap degree 1, the tree T_c that we obtain for the consequent satisfies the condition that all of its nodes have gap degree 1 (since their projections are the same as in the antecedent tree) except for its head, that may have gap degree 2 (since its projection is that of the head node of T_a, plus a new node w_{h1} that can increase the gap degree at most by one). The new link appearing in the consequent item must be licensed by our set of D-rules G, by the side condition of the step. Finally, the well-nestedness constraint is also preserved, since the subtrees induced by nodes in T_c are the same as those in T_a except for the one induced by w_{h1}, which cannot interleave with any other as it contains them all. Therefore, if the antecedents of a *Link Gapped* step are nonempty, we conclude that the consequent is also nonempty, since it contains the valid tree T_c. The same reasoning can be applied to the *Link Ungapped* step.

In the case of *Combiner* steps, a tree T_c for the consequent item can be obtained from the union of two trees T_a and T_b, each taken from one of the antecedent items, and having a common head w_h. In this case, no new links are created, so the consequent tree is obviously permitted by the D-rules G if the antecedent trees are. Condition (1) of a valid tree is satisfied by construction, since the required projection of the head for a valid tree in the consequent of a *Combiner* is the union of those for the antecedents,

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 $a(T, w_k) = a_1 a_2 \ldots a_n$, where $a_k = 0$, $a_i = u$ if $i \in \lfloor w_{d_u} \rfloor$, and $a_i =$“,” (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.*

[4] Note that we use a slightly different notation from (Kuhlmann, 2007): for simplicity in the proofs, we say that each node has a single annotation of the form $\alpha_1, \alpha_2, \ldots, \alpha_n$ instead of saying that it has a list of annotations $\alpha_1, \alpha_2, \ldots \alpha_n$. Of course, the difference is merely notational.

Property 4. *A dependency structure T is ill-nested if, and only if, it contains at least one order annotation of the form $\ldots a \ldots b \ldots a \ldots b \ldots$, 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 at a node w_h, such that $\#(\lfloor w_h \rfloor) = 1$. Since T has only one node, it must be the trivial dependency tree consisting of the single node w_h. In this case, Lemma 1 trivially holds because the initial item $[h, h, h, \diamond, \diamond]$ contains this tree, and initial items are valid by definition.

Induction step Let T be a valid partial dependency tree rooted at a node w_h, such that $\#(\lfloor w_h \rfloor) = N$ (for some $N > 1$).

We will prove that, if Lemma 1 holds for every valid partial dependency tree T' rooted at w'_h such that $\#(\lfloor w'_h \rfloor) < N$, then it also holds for T.

Let $w_{d_1} \ldots w_{d_p}$ be the direct children of w_h in T, ordered by the index of their leftmost transitive dependent, i.e., for every i and j such that $1 \leq i < j \leq p$, then $min\{k \mid w_k \in \lfloor w_{d_i} \rfloor\} < min\{k \mid w_k \in \lfloor w_{d_j} \rfloor\}$.

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 definition), 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 definition), 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 definition), 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 $\{0\} \cup \{1\} \cup \ldots \cup \{p\} \cup \{,\}$; containing a single appearance of the symbol 0. Additionally, by the definition of a valid tree and Property 3 of order annotations, $O(T, w_h)$ must contain either 1 or 2 appearances of each symbol 1 through p (since more than 2 appearances of a symbol q could only occur if w_{d_q} had gap degree ≥ 2). And, from the possible forms of $\lfloor w_h \rfloor$ in a valid tree, we know that $o(T, w_h)$ must have one of the following forms, where α and β are (possibly empty) strings that only contain symbols in $\{1\} \cup \ldots \cup \{p\}$ (not zeros or commas):

- (i) $\alpha 0 \beta$
- (ii) $\alpha, \beta 0 \gamma$
- (iii) $\alpha 0 \beta, \gamma$
- (iv) $0, \alpha, \beta$
- (v) $\alpha, \beta, 0$

[5]Note that the situation where the projection is of this form but with $d_1 \notin [i, j]$ is covered by the third case in this list if $d_1 < i - 1$ or $d_1 > j + 1$; or by this same case if $d_1 = i - 1$ or $d_1 = j + 1$, by rewriting the projection in the equivalent form $\{w_{d_1}\} \cup [i - 1, j]$ or $\{w_{d_1}\} \cup [i, j + 1]$, respectively.

[6]Note that this item is the normalisation of $[i, j, h, h, h]$.

- (vi) $\alpha, 0, \beta$

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 $\alpha 0, \beta, \gamma$, for nonempty α); 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 $w_{d_1} \rightarrow w_h$ 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 $w_{d_2} \rightarrow w_h, \dots, w_{d_p} \rightarrow w_h$ 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 $1\beta_1, \beta_2$, where either β_1 or β_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 $1\beta_1, 0, \beta_2$ or $1\beta_1, \beta_2, 0$. 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 $1\beta_1 1\beta_2$. Due to the well-nestedness constraint, we know that there is no symbol $s \in \{1\} \cup \{2\} \cup \ldots \cup \{p\}$ that appears both in β_1 and in β_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 $w_{d_{b_1}} \ldots w_{d_{b_q}}$, where $b_1 \ldots b_q$ are the non-comma, non-zero symbols appearing in β_1, and adding the node w_h and dependencies $w_{d_{b_1}} \rightarrow w_h, \ldots, w_{d_{b_q}} \rightarrow w_h$ to it.
- T_3: The tree obtained by taking the union of subtrees induced by $w_{d_{c_1}} \ldots w_{d_{c_q}}$, where $c_1 \ldots c_q$ are the non-comma, non-zero symbols appearing in β_2, and adding the node w_h and dependencies $w_{d_{c_1}} \rightarrow w_h, \ldots, w_{d_{c_q}} \rightarrow w_h$ to it.

Note that T_2 or T_3 may be empty trees, since it is possible that the string β_1 or β_2 do not contain any symbol except for zeros and commas. However, both trees cannot be empty at the same time, since in that case we would have $p = 1$.

With this, we divide this case into further cases:

i. If β_1 does not contain any comma: Then, by construction and by the well-nestedness constraint, we know that the projection of w_h in T_2 is of the form $\{w_h\} \cup [l, r]$. Applying the induction hypothesis to T_1, we know that the item $[i, j, h, l, r]$ is valid, and applying it to T_2, we know that $[l, r, h, \diamond, \diamond]$ is also valid. By applying a *Combine Closing Gap* step to these items, we obtain that $\iota = [i, j, h, \diamond, \diamond]$ is valid. Now, we divide into further cases according to the form of β_2:

A. If T_3 is empty (β_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 β_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 ι, we obtain $[i, k, h, \diamond, \diamond]$, the item containing the tree T.

C. If β_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 ι, we get that $[i, k, h, l', r']$ is valid, and this is the item containing the tree T in this case.

ii. If β_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 $\beta_3, 0$, 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 β_1 is of the form "$, \beta_3$", where β_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 $\iota = [i, j, h, l, l'-1]$ is a valid item. Now, if β_2 is empty, we are done: ι is the item containing the tree T. And if β_2 is nonempty, then it must either contain no commas, or be of the form $\beta_4, 0$ (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 ι 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 β_1 is of the form "$\beta_3,$", where β_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 applied to the item $[l, r', h, \diamond, \diamond]$ (for T_2) and the item $[i, j, h, l, r]$ (for T_1), obtaining $\iota = [i, j, h, r'+1, r]$. As in the previous case, if β_2 is empty we do not need to do anything else, and if it is nonempty we apply *Combine Keeping Gap Left* to obtain $[i, k, h, r'+1, r]$, the item containing T.

v. If β_1 is of the form "β_3, β_4", where β_3 and β_4 are not empty and do not

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 $\iota = [i, j, h, l', r']$ is valid. If β_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 ι by the *Combine Keeping Gap Left* step to obtain $[i, k, h, l', r']$, the item containing T.

vi. If β_1 contains two commas: in this case, by construction of the valid tree T, β_1 must be of the form $\beta_3, 0, \beta_4$, where β_3 and β_4 may or may not be empty. So we divide into subcases:

   A. If β_3 and β_4 are both empty, we apply the same reasoning as in case 1-b-ii, except that in this case we know that β_2 cannot contain any commas.

   B. If β_3 is empty and β_4 is nonempty, we apply the same reasoning as in case 1-b-iii, except that in this case we know that β_2 cannot contain any commas.

   C. If β_3 is nonempty and β_4 is empty, we apply the same reasoning as in case 1-b-iv, except that in this case we know that β_2 cannot contain any commas.

   D. If neither β_3 nor β_4 are empty, we apply the same reasoning as in case 1-b-v, except that in this case we know that β_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): $0, \alpha, \beta$. In this case, we can apply symmetric reasoning considering the last symbol of $o(T, w_h)$ instead of the first (note that the case $\alpha, \beta, 0$ 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 $[i, j, C, l, r]$, where C is a category, apart from the existing items $[i, j, h, l, r]$. Steps in the parser are duplicated, to work both with lexicalised and unlexicalised items, except for the *Link* steps, which always work with a lexicalised item and an unlexicalised hypothesis to produce an unlexicalised item, and the *Combine Shrinking Gap* steps, which can work only with unlexicalised items. Steps are added to obtain lexicalised items from their unlexicalised equivalents by binding the head to particular string positions. Finally, we need certain variants of the *Combine Shrinking Gap* steps that take 2 unlexicalised antecedents and produce a lexicalised consequent; an example is the following:

$$\textit{Combine Shrinking Gap Centre L:}\ \frac{[i, j, C, l, r] \qquad [l+1, r, C, l2, r2]}{[i, j, l, l2, r2]}$$

such that $cat(w_l)$=C

Although this version of the algorithm reduces time complexity with respect to the length of the input to $O(n^6)$, it also adds a factor related to the number of categories, as well as constant factors due to using more kinds of items and steps than the original WG_1 algorithm. This, together with the advantages of lexicalised dependency parsing, may mean that the original WG_1 algorithm is more practical than this version.

4 The WG_k parser

The WG_1 parsing schema can be generalised to obtain a parser for all well-nested dependency structures with gap degree bounded by a constant $k(k \geq 1)$, which we call WG_k parser. In order to do this, we extend the item set so that it can contain items with up to k gaps, and modify the deduction steps to work with these multi-gapped items.

4.1 Parsing schema for WG_k

The item set $\mathcal{I}_{WGk}$ is the set of all $[i, j, h, [(l_1, r_1), \ldots, (l_g, r_g)]]$ where $i, j, h, g \in \mathbb{N}$, $0 \leq g \leq k$, $1 \leq h \leq n$, $1 \leq i \leq j \leq n$, $h \neq j$, $h \neq i-1$; and for each $p \in \{1, 2, \ldots, g\}$: $l_p, r_p \in \mathbb{N}$, $i < l_p \leq r_p < j$, $r_p < l_{p+1} - 1$, $h \neq l_p - 1$, $h \neq r_p$.

An item $[i, j, h, [(l_1, r_1), \ldots, (l_g, r_g)]]$ represents the set of all well-nested partial dependency trees rooted at w_h such that $\lfloor w_h \rfloor = \{w_h\} \cup ([i, j] \setminus \bigcup_{p=1}^{g} [l_p, r_p])$, where each interval $[l_p, r_p]$ is called a gap. The constraints $h \neq j, h \neq i+1, h \neq l_p - 1, h \neq r_p$ are added to avoid redundancy, and normalisation is defined as in WG_1. The set of final items is defined as the set $\mathcal{F} = \{[1, n, h, []] \mid h \in \mathbb{N}, 1 \leq h \leq n\}$. Note that this set is the same as in WG_1, as these are the items that we denoted $[1, n, h, \diamond, \diamond]$ in the previous parser.

The parser has the following deduction steps:

Link: $\dfrac{[h1, h1, h1, []] \qquad [i2, j2, h2, [(l_1, r_1), \ldots, (l_g, r_g)]]}{[i2, j2, h1, [(l_1, r_1), \ldots, (l_g, r_g)]]}\ w_{h2} \rightarrow w_{h1}$

such that $w_{h2} \in [i2, j2] \setminus \bigcup_{p=1}^{g} [l_p, r_p]$

$\wedge w_{h1} \notin [i2, j2] \setminus \bigcup_{p=1}^{g} [l_p, r_p]$.

Combine Shrinking Gap Right:

$$\frac{\begin{array}{c}[i, j, h, [(l_1, r_1), \ldots, (l_{q-1}, r_{q-1}), (l_q, r'), (l_s, r_s), \ldots, (l_g, r_g)]] \\ [r_q + 1, r', h, [(l_{q+1}, r_{q+1}), \ldots, (l_{s-1}, r_{s-1})]]\end{array}}{[i, j, h, [(l_1, r_1), \ldots, (l_g, r_g)]]}$$

such that $g \leq k$

Combine Opening Gap:

$$\frac{[i, l_q - 1, h, [(l_1, r_1), \ldots, (l_{q-1}, r_{q-1})]] \qquad [r_q + 1, m, h, [(l_{q+1}, r_{q+1}), \ldots, (l_g, r_g)]]}{[i, m, h, [(l_1, r_1), \ldots, (l_g, r_g)]]}$$

such that $g \leq k$ and $l_q \leq r_q$,

Combine Shrinking Gap Left:

$$\frac{\begin{array}{c}[i, j, h, [(l_1, r_1), \ldots, (l_q, r_q), (l', r_s), (l_{s+1}, r_{s+1}), \ldots, (l_g, r_g)]] \\ [l', l_s - 1, h, [(l_{q+1}, r_{q+1}), \ldots, (l_{s-1}, r_{s-1})]]\end{array}}{[i, j, h, [(l_1, r_1), \ldots, (l_g, r_g)]]}$$

such that $g \leq k$

Combine Keeping Gaps:

$$\frac{[i, j, h, [(l_1, r_1), \ldots, (l_q, r_q)]] \qquad [j + 1, m, h, [(l_{q+1}, r_{q+1}), \ldots, (l_g, r_g)]]}{[i, m, h, [(l_1, r_1), \ldots, (l_g, r_g)]]}$$

such that $g \leq k$,

Combine Shrinking Gap Centre:

$$\frac{\begin{array}{c}[i, j, h, [(l_1, r_1), \ldots, (l_q, r_q), (l', r'), (l_s, r_s), \ldots, (l_g, r_g)]] \\ [l', r', h, [(l_{q+1}, r_{q+1}), \ldots, (l_{s-1}, r_{s-1})]]\end{array}}{[i, j, h, [(l_1, r_1), \ldots, (l_g, r_g)]]}$$

such that $g \leq k$

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 definition of valid trees to a higher gap degree. A valid tree in WG_k can be defined 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 define correct items and correct final items analogously to their definition 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 $k + 1$ commas); then the constraint on the form of $\lfloor w_h \rfloor$ in valid trees implies that the symbol 0 cannot be contiguous to any non-comma symbol in $o(T, w_h)$.

With this, the cases 1a) of the completeness proof for WG_1 can be directly used for WG_k, only taking into account that $o(T, w_h)$ can contain up to $k + 1$ commas. As a consequence of this, instead of *Combine Keeping Gap Right* we use a general *Combine Keeping Gaps* step with more than one gap allowed in its rightmost antecedent item. In the cases 1b), we need to take into account that the symbol 1 can appear up to $k + 1$ times in $o(T, w_h)$. We write $o(T, w_h)$ as $1\beta_1 1\beta_2 \dots 1\beta_g (g \leq k)$ and do with each β_i (for $i < g$) the same case analysis as we do with β_1 in the WG_1 case, and with β_g the same case analysis as with β_2 in the WG_1 case. Each of these cases is proven as in WG_1, with the difference that each string β_i can contain more than one comma, so that instead of the *Combine Shrinking Gap* steps in WG_1 we need to use the general *Combine Shrinking Gap* steps in WG_k, which allow their inner items to have more than one gap. In the

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 defined 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 significant 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 configurations*, 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 parser implicitly works by finding such a binarisation, since *Combine* steps are always applied to two items and no intermediate item generated by them can exceed gap degree k (not counting the position of the head in the projection).

More formally, let $w_1 \dots w_n$ be a string, and T a partial dependency tree headed at a node w_h. A *binarisation* of T is a tree B in which each node has at most two children, and such that:

- (a) Each node in B can be either unlabelled, or labelled with a word w_i. Note that several nodes may have the same label (in the definition of a dependency graph, the set of nodes was the set of words, so a word cannot appear twice in the graph).
- (b) A node labelled w_i is a descendant of w_j in B if and only if $w_i \rightarrow^\star w_j$ in T.

The projection of a node in a binarisation is the set of reflexive-transitive children of that node. With this, we can define the gap degree of a binarisation in the same way as that of a dependency structure. If we denote by $\lfloor n \rfloor_T$ the projection of a node n in a tree T, the condition (b) of a binarisation can be rewritten as follows: $w_i \in \lfloor w_j \rfloor_B \Leftrightarrow w_i \in \lfloor w_j \rfloor_T$.

A dependency structure is **mildly ill-nested** for gap degree k if it has at least one binarisation of gap degree $\leq k$. Otherwise, we say that it is **strongly ill-nested** for gap degree k. It is easy to prove that the set of mildly ill-nested structures for gap degree k includes all well-nested structures with gap degree up to k.

We define MG_1, a parser for mildly ill-nested structures for gap degree 1, as follows:

- the item set is the same as that of WG_1, except that items can now contain any mildly ill-nested structures for gap degree 1, instead of being restricted to well-nested structures; and
- deduction steps are the same as in WG_1, plus the additional steps shown in Figure 1. These extra *Combiner* steps allow the parser to combine interleaved subtrees with simple crossing configurations. The MG_1 parser still runs in $O(n^7)$, as these new steps do not use more than 7 string positions.

$$\textit{Combine Interleaving:}\ \frac{[i,j,h,l,r]}{[l,k,h,r+1,j]}\ \Big/\ [i,k,h,\diamond,\diamond] \qquad \textit{Combine Interleaving Gap C:}\ \frac{[i,j,h,l,r]}{[l,k,h,m,j]}\ \Big/\ [i,k,h,m,r]$$

such that $m < r+1$,

$$\textit{Combine Interleaving Gap L:}\ \frac{[i,j,h,l,r]}{[l,k,h,r+1,u]}\ \Big/\ [i,k,h,j+1,u] \qquad \textit{Combine Interleaving Gap R:}\ \frac{[i,j,h,l,r]}{[k,m,h,r+1,j]}\ \Big/\ [i,m,h,l,k-1]$$

such that $u > j$, such that $k > l$.

Figure 1: Additional steps to turn WG_1 into MG_1.

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

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: General form of the MG_k Combiner step.

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 to gaps in the joined structure. The legal combinations of structures for gap degree k will correspond to strings where symbols a and b each appear at most $k+1$ times, g appears at most k times and is not the first or last symbol, and there is no more than one consecutive appearance of any symbol. Given a string of this form, the corresponding *Combiner* step is given by the expression in Figure 2. As a particular example, the *Combine Interleaving Gap C* step in Figure 1 can be obtained from the string *abgab*.

Therefore, we define the parsing schema for MG_k, a parser for mildly ill-nested structures for gap degree k, as the schema where

- the item set is the same as that of WG_k, except that items can now contain any mildly ill-nested structures for gap degree k, instead of being restricted to well-nested structures; and
- the set of deduction steps consists of a *Link* step as the one in WG_k, plus a set of *Combiner* steps obtained as expressed in Figure 2.

5.2 Complexity

As the string used to generate a *Combiner* step can have length at most $3k+2$, and the resulting step contains an index for each symbol of the string plus two extra indexes, it is easy to see that the MG_k parser has complexity $O(n^{3k+4})$. Note that the item and deduction step sets of an MG_k parser are always supersets of those of WG_k. In particular, the steps for WG_k are those obtained from strings that do not contain *abab* or *baba* as a scattered substring.

5.3 Proof of correctness for MG_k

In order to prove the correctness of the MG_k parser, we will first introduce some properties of binarisations that arise as corollaries of their definition in Section 5.1. If a tree B is a binarisation of a (partial) dependency tree T headed at w_h, then we have that:

- (i) A node w_i appears in T if and only if a node labelled w_i appears in B,
- (ii) $\lfloor w_i \rfloor_B = \lfloor w_i \rfloor_T$,
- (iii) If the root of B is labelled, then its label is w_h.

Properties (i) and (ii) are direct consequences of condition (b) of the definition of a binarisation. Property (iii) is obtained from (b) and property (i): the label of the root node of B cannot be a $w_d \neq w_h$ because this would require w_h to be a transitive

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 defining 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 satisfies 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 definition 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 final 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, final items in MG_k only contain mildly ill-nested trees for gap degree k, as expected.

To prove correctness of the MG_k parser, we need to prove its soundness and completeness.

5.3.1 Soundness

As in the proofs for the previous algorithms, we prove soundness of the MG_k parser by showing that (i) hypotheses are correct, and (ii) if the antecedents of a deduction step in WG_1 are correct, then the consequent is also correct. (i) is trivial, since each hypothesis in the MG_k parser contains a tree consisting of a single node w_h, which is trivially a valid tree.

To show (ii), given a set of D-rules G, we must prove that if the antecedents of a deduction step are items containing a valid tree for MG_k licensed by the D-rules in G, then the consequent must also contain a valid tree for MG_k licensed by G. In order to do this, we obtain a valid tree for the consequent item of each step from a valid tree for each of its antecedents exactly in the same way as in WG_k: by adding a new head node

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 $w_d \rightarrow w_h$, 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 binarisation whose head does not have gap degree greater than k, and linking it to a new head adds at most one gap); and the rest of the nodes have gap degree at most k because their projections do not change with respect to the binarisation of the antecedent tree. This binarisation trivially verifies (3), because its root node is labelled w_h and has the head of the B_a as its only child, and (4) because it can only contain one node labelled w_h, which is the root, as w_h cannot appear in B_a.

In the case of *Combiner* steps, if B_1 and B_2 are the binarisations corresponding to the antecedent items, we can construct a binarisation for the consequent B_c from B_1 and B_2 as follows:

- If the consequent item is of the form $[i, j, h, [(l_1, r_1) \dots (l_g, r_g)]]$ such that $w_h \notin ([i, j] \setminus \bigcup_{p=1}^{g} [l_p, r_p])$, then we take the binarisations B_1 and B_2, we remove their head nodes labelled w_h from them, we link the direct children of that head in each of the two binarisations (which must be two, d_1 and d_2, since B_1 and B_2 verify condition (3)) to a fresh unlabelled node, and finally we link this unlabelled node to w_h. This tree B_c is a binarisation for the tree in the consequent item obtained by performing the union of two trees in the antecedent items. It can be shown that the projection of w_h in B_c satisfies condition (1) by construction, following the same reasoning as in the proof for WG_1. And we can see that B_c also meets the constraints of (2) because:
  - The projection of w_h in B_c is the union of the projections of w_h in B_1 and B_2, which by construction of the consequent of *Combiner* steps, and property (ii) of binarisations, must be of the form $\lfloor w_h \rfloor_{B_c} = \{w_h\} \cup ([i, j] \setminus \bigcup_{p=1}^{g} [l_p, r_p])$ with $g \leq k$. Since the gap degree of $\bigcup_{p=1}^{g} [l_p, r_p])$ cannot exceed k, the gap degree of $\lfloor w_h \rfloor_{B_c}$ cannot exceed $k + 1$.

- The fresh unlabelled node that we have added does not dominate any node labelled w_h. We know this because no antecedent item cannot be of the form described in (4), since if one of the antecedent items were of that form, then the consequent item would be of that form too, by construction of consequent items. Therefore, for the binarisations corresponding to antecedent items, we know that they contain a single node labelled w_h, and thus our unlabelled node does not dominate any node labelled w_h. Therefore, the projection of this node must be $\lfloor w_h \rfloor_{B_c} \setminus \{w_h\}$, which in this case equals $([i,j] \setminus \bigcup_{p=1}^{g} [l_p, r_p])$ with $g \leq k$, and therefore the node has gap degree $\leq k$.
- The rest of the nodes in B_c have the same projection as they had in B_1 or B_2, so they have gap degree $\leq k$.

It can be seen that this binarisation also satisfies (3) and (4) because, by construction, it has a single node labelled w_h which is its root, and this node has a single child.

- If the consequent item is of the form $[i, j, h, [(l_1, r_1) \dots (l_g, r_g)]]$ such that $w_h \in ([i,j] \setminus \bigcup_{p=1}^{g} [l_p, r_p])$, then we take the binarisations B_1 and B_2, we remove their head nodes labelled w_h from them, we link the direct children of that head in each of the two binarisations (which must be two nodes, d_1 and d_2, as B_1 and B_2 satisfy (3)) to a fresh node labelled w_h, and finally we link this node to another node also labelled w_h. The obtained tree B_c is a binarisation for the valid tree in the consequent item obtained by performing the union of two trees in the antecedent items. It satisfies condition (1) by construction, as in the previous case, and meets the constraints of (2) because:
  - By construction, the projection of both fresh nodes labelled w_h in this case is $\lfloor w_h \rfloor_{B_1} \cup \lfloor w_h \rfloor_{B_2}$, and by the hypothesis of this case we know that that projection is of the form $\lfloor w_h \rfloor_{B_c} = ([i,j] \setminus \bigcup_{p=1}^{g} [l_p, r_p])$, and therefore has gap degree at most k.
  - The rest of the nodes in B_c have the same projection as they had in B_1 or B_2, so they have gap degree $\leq k$.

  This binarisation trivially verifies (3), and it also meets (4) because the item associated to the consequent is of the form that allows several nodes to be labelled w_h.

With this, we have proven that if an MG_k step is applied to correct antecedents, it produces correct consequents, and we conclude the soundness proof for MG_k.

5.3.2 Completeness

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

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 $\iota = [i, j, d, (l_1, r_1), \ldots, (l_g, r_g)]$ (which contains T' by construction) is valid. The item $[i, j, h, (l_1, r_1), \ldots, (l_g, r_g)]$ containing T can be obtained from ι 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' 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' 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' are well-formed binarisations of the same tree, if Lemma 2 holds for B', it also must hold for B. As we know that $\#B' < N$ (since it contains one less node than B), Lemma 2 holds for B' 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' induced by this single child node is a binarisation of the same tree as B. We know that B' 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' are binarisations of the same tree and we know that Lemma 2 holds for B' 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'_1 and B'_2:

  - B'_1 is the tree obtained by taking the subtree induced by c_1 and linking its head c_1 to w_h,
  - B'_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'_1 and B'_2, except for the head, must have gap degree $\leq k$ because their projection in B'_1 and B'_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'_1 and B'_2 because, by construction, $\lfloor w_h \rfloor_{B'_1} = \lfloor c_1 \rfloor_B \cup \{w_h\}$, and $\lfloor c_1 \rfloor_B$ has gap degree $\leq k$; and a similar reasoning can be made in B'_2. Thus, we have that B'_1 and B'_2 are well-formed binarisations.

  By applying the induction hypothesis to B'_1 and B'_2, we obtain that the items containing their associated dependency trees T'_1 and T'_2 are valid. By construction, since c_1 has gap degree $\leq k$ in B'_1 and c_2 has gap degree $\leq k$ in B'_2, the projection of w_h in the trees T'_1 and T'_2 obtained by unbinarising B'_1 and B'_2 by removing the unlabelled and redundant nodes will be the union of g_1 and g_2 intervals respectively,

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 classified by gap degree, and mild and strong ill-nestedness (for their gap degree); appearing in treebanks for Arabic (Hajič et al., 2004), Czech (Hajič et al., 2006), Danish (Kromann, 2003), Dutch (van der Beek et al., 2002), Latin (Bamman and Crane, 2006), Portuguese (Afonso et al., 2002), Slovene (Džeroski et al., 2006), Swedish (Nilsson et al., 2005) and Turkish (Oflazer 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' and T_2' 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 correspond to all the ways in which two unions of up to $k+1$ intervals each can be combined into another by performing their union, we know that the item that contains T can be obtained from the items containing T_1' and T_2' by a *Combiner* step, and thus this item is valid, concluding the completeness proof.

5.4 Mildly ill-nested dependency structures

The MG_k algorithm defined in the previous section allows us to parse any mildly ill-nested structure for a given gap degree k in polynomial time. We have characterised the set of mildly ill-nested structures for gap degree k as those that have a binarisation of gap degree $\leq k$. Since a binarisation of a dependency structure cannot have lower gap degree than the original structure, the set of mildly ill-nested structures for gap degree k only contains structures with gap degree at most k. Furthermore, the set of mildly ill-nested structures for gap degree k contains all the well-nested structures with gap degree up to k.

Figure 3 shows an example of a structure that has gap degree 1, but is strongly ill-nested for gap degree 1. This is one of the smallest possible such structures: by generating all the possible trees up to 10 nodes (without counting a dummy root node located at position 0), it can be shown that all the structures of any gap degree k with length smaller than 10 are well-nested or only mildly ill-nested for that gap degree k.

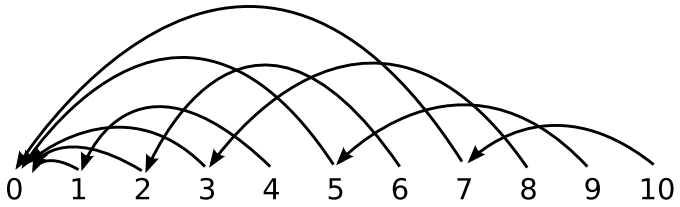

Figure 3: One of the smallest strongly ill-nested structures. This dependency structure has gap degree 1, but is only mildly ill-nested for gap degree ≥ 2.

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 be binarised, and binarisations have finite gap degree). For example, the structure in Figure 3 is mildly ill-nested for gap degree 2. Therefore, MG_k parsers have the property of being able to parse any possible dependency structure as long as we make k large enough.

In practice, structures like the one in Figure 3 do not seem to appear in dependency treebanks. We have analysed treebanks for nine different languages, obtaining the data presented in Table 1. None of these treebanks contain structures that are strongly ill-nested for their gap degree. Therefore, in any of these treebanks, the MG_k parser can parse every sentence with gap degree at most k.

6 Conclusions and future work

We have defined a parsing algorithm for well-nested dependency structures with bounded gap degree. In terms of computational complexity, this algorithm is comparable to the best parsers for related constituency-based formalisms: when the gap degree is at most 1, it runs in $O(n^7)$, like the fastest known parsers for LTAG, and can be made $O(n^6)$ if we use unlexicalised dependencies. When the gap degree is greater than 1, the time complexity goes up by a factor of n^2 for each extra unit of gap degree, as in parsers for coupled context-free grammars. Most of the non-projective sentences appearing in treebanks are well-nested and have a small gap degree, so this algorithm directly parses the vast majority of the non-projective constructions present in natural languages, without requiring the construction of a constituency grammar as an intermediate step.

Additionally, we have defined a set of structures for any gap degree k which we call mildly ill-nested. This set includes ill-nested structures verifying certain conditions, and can be parsed in $O(n^{3k+4})$ with a variant of the parser for well-nested structures. The practical interest of mildly ill-nested structures can be seen in the data obtained from several dependency treebanks, showing that all of the ill-nested structures in them are mildly ill-nested for their corresponding gap degree. Therefore, our $O(n^{3k+4})$ parser can analyse all the gap degree k structures in these treebanks.

The set of mildly ill-nested structures for gap degree k are defined as the set of structures that have a binarisation of gap degree at most k. This definition is directly related

to the way the MG_k parser works, since it implicitly finds such a binarisation. An interesting line of future work would be to find an equivalent characterisation of the set of mildly ill-nested structures which is more grammar-oriented and would provide a more linguistic insight into these structures.

References

Susana Afonso, Eckhard Bick, Renato Haber, and Diana Santos. "floresta sintá(c)tica": a treebank for Portuguese. In *Proceedings of the 3rd Intern. Conf. on Language Resources and Evaluation (LREC)*, pages 1968–1703, 2002.

Nart B. Atalay, Kemal Oflazer, and Bilge Say. The annotation process in the turkish treebank. In *In Proc. of EACL Workshop on Linguistically Interpreted Corpora (LINC)*, pages 243–246, 2003.

David Bamman and Gregory Crane. The design and use of a Latin dependency treebank. In *Proceedings of the Fifth Workshop on Treebanks and Linguistic Theories (TLT2006)*, pages 67–78, 2006. URL `http://ufal.mff.cuni.cz/tlt2006/pdf/110.pdf`.

Manuel Bodirsky, Marco Kuhlmann, and Mathias Möhl. Well-nested drawings as models of syntactic structure (extended version). Technical report, Saarland University, 2005.

Michael A. Covington. A dependency parser for variable-word-order languages. Technical Report AI-1990-01, Athens, GA, 1990. URL `citeseer.ist.psu.edu/article/covington90dependency.html`.

Sašo Džeroski, Tomaž Erjavec, Nina Ledinek, Petr Pajas, Zdeněk Žabokrtský, and Andreja Žele. Towards a slovene dependency treebank. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006)*, pages 1388–1391, 2006.

Jason Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340–345, Copenhagen, August 1996. URL `citeseer.ist.psu.edu/eisner97three.html`.

Jason Eisner and Giorgio Satta. A faster parsing algorithm for lexicalized tree-adjoining grammars. In *Proceedings of the 5th Workshop on Tree-Adjoining Grammars and Related Formalisms (TAG+5)*, pages 14–19, Paris, May 2000.

Jason Eisner and Giorgio Satta. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 457–464, Morristown, NJ, USA, 1999. Association for Computational Linguistics. ISBN 1-55860-609-3. doi: http://dx.doi.org/10.3115/1034678.1034748.

Carlos Gómez-Rodríguez, 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 Hajič, Otakar Smrž, Petr Zemánek, Jan Šnaidauf, and Emanuel Beška. 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 Hajič, Jarmila Panevová, Eva Hajičová, Jarmila Panevová, Petr Sgall, Petr Pajas, Jan Štěpánek, Jiří Havelka, and Marie Mikulová. Prague dependency treebank 2.0 (ldc2006t01). CDROM CAT: LDC2006T01., ISBN 1-58563-370-4, 2006.

Jiří 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.

Günter 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, Saarbrücken, Germany, 2007.

Marco Kuhlmann and Mathias Möhl. 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, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530, Morristown, NJ, USA, 2005. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1220575.1220641.

Jens Nilsson, Johan Hall, and Joakim Nivre. MAMBA meets TIGER: Reconstructing a Swedish treebank from antiquity. In *Proceedings of NODALIDA 2005 Special Session on Treebanks*, pages 119–132, May 2005.

Joakim Nivre and Jens Nilsson. Pseudo-projective dependency parsing. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 99–106, Morristown, NJ, USA, 2005. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1219840.1219853.

Kemal Oflazer, Bilge Say, Dilek Zeynep Hakkani-Tür, and Gökhan Tür. Building a Turkish treebank. in a. abeille, ed., *Building and Exploiting Syntactically-annotated Corpora*. kluwer, dordrecht., 2003.

Giorgio Satta. Recognition of linear context-free rewriting systems. In *Proceedings of the 30th annual meeting on Association for Computational Linguistics*, pages 89–95, Morristown, NJ, USA, 1992. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/981967.981979.

Klaas Sikkel. *Parsing Schemata — A Framework for Specification and Analysis of Parsing Algorithms*. Texts in Theoretical Computer Science — An EATCS Series. Springer-Verlag, Berlin/Heidelberg/New York, 1997. ISBN 3-540-61650-0.

L. van der Beek, G. Bouma, R. Malouf, and G. van Noord. The alpino dependency treebank. In *Computational Linguistics in the Netherlands (CLIN)*, Twente University, 2002.

K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th annual meeting on Association for Computational Linguistics*, pages 104–111, Morristown, NJ, USA, 1987. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/981175.981190.

Hiroyasu Yamada and Yuji Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of 8th International Workshop on Parsing Technologies*, pages 195–206, 2003. URL `http://www.jaist.jp/ h-yamada/pdf/iwpt2003.pdf`.