FG 2007:
The 12th conference on Formal Grammar
Dublin, Ireland
August 4-5, 2007

Organizing Committee:
Laura Kallmeyer **Paola Monachesi**
Gerald Penn **Giorgio Satta**

CENTER FOR THE STUDY
OF LANGUAGE
AND INFORMATION

Contents

1

The string-generative capacity of regular dependency languages

Marco Kuhlmann and Mathias Möhl

Abstract

This paper contributes to the formal theory of dependency grammar. We apply the classical concept of *algebraic recognizability* to characterize regular sets of dependency structures, and show how in this framework, two empirically relevant structural restrictions on dependency analyses yield infinite hierarchies of ever more expressive string languages.

Keywords dependency grammar, generative capacity

1.1 Introduction

Syntactic representations based on word-to-word dependencies have a long and venerable tradition in descriptive linguistics. Lately, they have also been used in many computational tasks; one of them is parsing. (Nivre, 2006, provides a good overview of the field.) In dependency parsing, there is a specific interest in *non-projective* dependency analyses, in which a word and its dependents may be spread out over a discontinuous region of the sentence; such structures naturally arise in the syntactic analysis of languages with flexible word order. Unfortunately, most formal results on non-projectivity are rather discouraging: for example, while grammar-driven dependency parsers that are restricted to projective structures can be as efficient as parsers for lexicalized context-free grammar (Eisner and Satta, 1999), parsing is prohibitively expensive when unrestricted forms of non-projectivity are permitted (Neuhaus and Bröker, 1997). A similar problem arises in data-driven dependency parsing (McDonald and Pereira, 2006).

FG-2007.
Organizing Committee:, Laura Kallmeyer, Paola Monachesi, Gerald Penn, Giorgio Satta.

In search of a balance between the need for more expressivity and the disadvantage of increased processing complexity, several authors have proposed constraints to identify classes of 'mildly' non-projective dependency structures that are computationally well-behaved. In this paper, we focus on two of these proposals: the *gap-degree restriction*, which puts a bound on the number of discontinuities in the region of a sentence covered by a word and its dependents, and the *well-nestedness condition*, which constrains the arrangement of dependency subtrees (Bodirsky et al., 2005). Both constraints have been shown to be in very good fit with data from dependency treebanks (Kuhlmann and Nivre, 2006). However, very little is known about the formal properties of *sets* of restricted non-projective dependency structures.

Contents of the paper

In this paper, we contribute to the formal theory of dependency grammar. This theory was pioneered by Gaifman (1965), who presented a formalism that generates sets of projective dependency structures and is weakly equivalent to context-free grammar. We generalize and extend Gaifman's work by exploring languages based on restricted classes of non-projective dependency structures: We apply the classical concept of *algebraic recognizability* (Mezei and Wright, 1967) to dependency structures; this yields a canonical notion of *regular dependency languages*. The main contribution of the paper is the result that, in this framework, both the gap-degree parameter and the well-nestedness condition have an immediate impact on string-generative capacity.

Regular dependency languages are weakly equivalent to the languages generated by lexicalized Linear Context-Free Rewriting Systems (LCFRS) (Weir, 1988, Kuhlmann and Möhl, 2007). We show that the string-language hierarchy known for LCFRS can be recovered by controlling the gap-degree parameter. Adding the well-nestedness condition leads to a proper decrease in generative capacity on nearly all levels of this hierarchy; it induces the language hierarchy known for Coupled Context-Free Grammars (Hotz and Pitsch, 1996). The hierarchies coincide only in the projective case (gap-degree 0), where we find the languages generated by Gaifman's formalism.

Structure of the paper

The remainder of the paper is structured as follows. Section 1.2 contains some basic notions related to strings and terms. In Section 1.3, we develop the notion of regular dependency languages. Then, in Section 1.4 we present our main results in the form of a hierarchy theorem. Section 1.5 concludes the paper.

1.2 Preliminaries

We expect the reader to be familiar with the basic concepts of universal algebra (see Denecke and Wismath, 2001, for an introduction), and only introduce our particular notation. Throughout the paper, we write $[n]$ to refer to the set of positive integers up to and including n.

1.2.1 Strings

An *alphabet* is a non-empty, finite set of *symbols*. The set of strings over the alphabet A is denoted by A^*; the set of non-empty strings is denoted by A^+. The empty string is denoted by ε. In the following, let $w \in A^*$ be a string over A. We write $|w|$ for the length of w, and define the set of *positions* in w as the set $pos(w) := \{\, i \in \mathbb{N} \mid 1 \leq i \leq |w| \,\}$.

1.2.2 Terms

Let S be a non-empty set of *sorts*. An *S-sorted alphabet* consists of an alphabet Σ and a *type assignment* $type_\Sigma : \Sigma \to S^+$. In the following, let $\sigma \in \Sigma$ be a symbol. We usually write $\sigma : s_1 \times \cdots \times s_n \to s$ instead of $type_\Sigma(\sigma) = s_1 \cdots s_n s$. The integer n is called the *rank* of σ. A *ranked alphabet* is a sorted alphabet with a single sort; in this case, the type of each symbol is uniquely determined by its rank. If Σ is a sorted alphabet and A is an alphabet, $\langle \Sigma, A \rangle$ denotes the sorted alphabet $\Sigma \times A$ in which

$$type_{\langle \Sigma, A \rangle}(\langle \sigma, a \rangle) \;=\; type_\Sigma(\sigma)\,, \qquad \text{for all } \langle \sigma, a \rangle \in \Sigma \times A.$$

In the following, let Σ be a sorted alphabet. The set of *terms* over Σ of sort s is defined by the recursive equation

$$T^s_\Sigma \;:=\; \{\, \sigma(t_1, \ldots, t_n) \mid \sigma : s_1 \times \cdots \times s_n \to s \,\wedge\, \forall i \in [n].\, t_i \in T^{s_i}_\Sigma \,\}\,.$$

The set of all terms over Σ is defined as $T_\Sigma := \{\, t \mid s \in S \,\wedge\, t \in T^s_\Sigma \,\}$. For a given term t, the set of *nodes of* t is a subset of $\mathbb{N}^*$, defined recursively as $nod(\sigma(t_1, \ldots, t_n)) := \{\varepsilon\} \cup \{\, iu \mid i \in [n] \,\wedge\, u \in nod(t_i) \,\}$, where juxtaposition denotes concatenation. We also use the notations t/u for the subterm of t rooted at u, and $t(u)$ for the symbol at the node u of t. A *context* is a term $c \in T_\Sigma$ in which some subterm $c/u \in T^s_\Sigma$ has been replaced by the special symbol $\Box^s_c$, the *hole* of c. We write C_Σ for the set of all contexts obtained from terms in T_Σ. Given a context $c \in C_\Sigma$ with hole $\Box^s_c$ and a term $t \in T^s_\Sigma$, we write $c \cdot t$ for the term that results from replacing the hole of c by t. We also define the following notation for iterated replacement into contexts:

$$c^0 \cdot t := t\,, \qquad c^n \cdot t := c \cdot c^{n-1} \cdot t\,, \;\; n \geq 1\,.$$

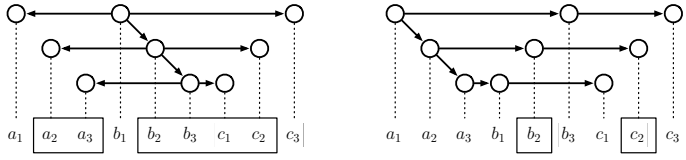

FIGURE 1 Two labelled dependency structures

1.3 Regular dependency languages

In this section, we introduce dependency structures, and motivate the notion of regular dependency languages.

1.3.1 Dependency structures

For the purposes of this paper, dependency structures are defined relative to a ranked alphabet Σ of term constructors and an (unranked) alphabet A of labels. More precisely, a *labelled dependency structure* over Σ and A is a pair $d = (t, \vec{u})$, where $t \in T_{\langle \Sigma, A \rangle}$ is a term, and $\vec{u}$ is a list of the nodes in t. Given two nodes $u_1, u_2 \in \mathit{nod}(t)$, we say that u_1 *governs* u_2, and write $u_1 \trianglelefteq u_2$, if $u_2 = u_1 w$, for some sequence $w \in \mathbb{N}^*$; we say that u_1 *precedes* u_2, $u_1 \preceq u_2$, if the position of u_1 in $\vec{u}$ is equal to or properly precedes the position of u_2; we say that u_1 is *labelled* with a, $\lambda(u_1) = a$, if $t(u_1) = \langle \sigma, a \rangle$, for some $\sigma \in \Sigma$. The *surface string* of a dependency structure $d = (t, u_1 \cdots u_n)$ is defined as $s(d) := \lambda(u_1) \cdots \lambda(u_n)$.

Example 1 Figure 1 shows how we visualize dependency structures: nodes are represented by circles, governance is represented by arrows, precedence by the left-to-right order of the nodes, and labelling by the dotted lines. (The boxes are irrelevant for now.)

We write $D_\Sigma(A)$ to refer to the set of all labelled dependency structures over Σ and A. A *dependency language* over Σ and A is a subset of $D_\Sigma(A)$. Given such a dependency language, the *term language* and the *string language* corresponding to L are defined as follows:

$$\begin{aligned} tL &:= \{\, t \in T_{\langle \Sigma, A \rangle} \mid (t, \vec{u}) \in L \,\}, \\ sL &:= \{\, w \in A^+ \mid d \in L \wedge s(d) = w \,\}. \end{aligned}$$

1.3.2 Structural constraints

We now define the classes of 'mildly' non-projective dependency structures that we want to investigate in this paper. Let $d = (t, \vec{u})$ be a dependency structure, and let $u, u_1, u_2 \in \mathit{nod}(t)$ be nodes. The *yield* of u is the set of all nodes governed by u; it is denoted by $\lfloor u \rfloor$. The *interval* between u_1 and u_2 is the set of all nodes preceded by $\min(u_1, u_2)$

and succeeded by $\max(u_1, u_2)$; it is denoted by $[u_1, u_2]$. Every yield $\lfloor u \rfloor$ is partitioned into *blocks*: u_1, u_2 are in the same block of $\lfloor u \rfloor$, if $u \trianglelefteq w$, for all nodes $w \in [u_1, u_2]$. The number of blocks that constitute the yield $\lfloor u \rfloor$ of u is called the *block-degree* of u. The *gap-degree* of u is the block-degree, minus 1. The class of all dependency structures in which every node has block-degree at most k is denoted by D_k. Two yields $\lfloor u_1 \rfloor, \lfloor u_2 \rfloor$ *interleave*, if there are nodes $v_1, w_1 \in \lfloor u_1 \rfloor$ and $v_2, w_2 \in \lfloor u_2 \rfloor$ such that $v_1 \prec v_2$, $v_2 \prec w_1$, and $w_1 \prec w_2$. A dependency structure is called *well-nested*, if for every pair of interleaving yields $\lfloor u_1 \rfloor, \lfloor u_2 \rfloor$, either $u_1 \trianglelefteq u_2$ or $u_2 \trianglelefteq u_1$ holds. The class of all well-nested dependency structures is denoted by D_{wn}.

Example 2 Both structures in Figure 1 have block-degree 2. To witness, consider the node b_2: the yield of this node falls into two blocks (marked by the boxes), and this is the highest number of blocks per node. The left structure is well-nested. The right structure is not well-nested; for example, the yields $\lfloor b_1 \rfloor$ and $\lfloor b_2 \rfloor$ interleave, but neither does b_1 govern b_2, nor vice versa.

1.3.3 Local order annotations

Kuhlmann and Möhl (2007) show how to encode dependency structures into terms over a sorted set Ω of *local order annotations*. This result is based on the observation that the blocks of a node have a recursive structure that closely follows the term structure: the blocks of a node u can be decomposed into the singleton interval containing u, and the blocks of the children of u. The precedence relation of a dependency structure can therefore be represented, in a unique way, by annotating each node u with an order on its sub-blocks, and information on how to merge adjacent sub-blocks to obtain the blocks of u.

Example 3 Consider the node b_2 in the left structure in Figure 1. This node has two blocks: the first block consists of the first (and only) block of a_2, followed by the first block of b_3; the second block consists of b_2 itself, followed by the second block of b_3 and the first (and only) block of c_2. If we number the direct dependents of b_2 from left-to-right, then the local order among the constituents of the blocks of b_2 can be annotated as $\langle 12, 023 \rangle$, where the ith occurrence of the symbol j refers to the ith block of the direct dependent j, the symbol 0 refers to b_2, and the two components of the tuple represent the two blocks of b_2.

An order annotation ω of a node u with n children fixes the block-degree of u as the number $|\omega|$ of components of ω and the block-degree of the ith child of u as the number $\#_i(\omega)$ of occurrences of the symbol i in ω. To reflect this, ω receives the type $\#_1(\omega) \times \cdots \times \#_n(\omega) \to |\omega|$.

1.3.4 Regular sets of dependency structures

Based on the local order annotations, we can give an algebraic structure to dependency structures as follows: Let $k \in \mathbb{N}$. The set D^k_Ω of *expanded dependency structures* over Ω of sort k is the set of all pairs $(t, \langle \vec{u}_1, \ldots, \vec{u}_k \rangle)$ with $\vec{u}_i \neq \varepsilon$ for all $i \in [k]$, and $(t, \vec{u}_1 \cdots \vec{u}_k) \in D_\Omega$.[1] To every order annotation $\omega \in \Omega$ of a node with n children, we associate an n-ary algebraic operation f_ω on expanded dependency structures. This operation takes the disjoint union of its arguments, introduces a new root node, puts the blocks corresponding to the components of the argument structures and the block for the root node into the order defined by ω, and groups these ordered constituents into blocks according to the component structure of ω. An order annotation $\omega : k_1 \times \cdots \times k_n \to k$ thus is a compact description of a function $f_\omega : D^{k_1}_\Omega \times \cdots \times D^{k_n}_\Omega \to D^k_\Omega$.

Example 4 The local order annotation $\langle 12, 023 \rangle$ gives rise to a composition operation $f_{\langle 12,023 \rangle} : D^1_\Omega \times D^2_\Omega \times D^1_\Omega \to D^2_\Omega$. In the composition of the left structure in Figure 1, this operation takes the sub-structure with yield $\{a_2\}$ (1 block), the sub-structure with yield $\{a_3, b_3, c_1\}$ (2 blocks), and the sub-structure with yield $\{c_2\}$ (1 block) and produces the substructure with yield $\{a_2, a_3, b_2, b_3, c_1, c_2\}$ (2 blocks).

Once we are able to view dependency structures as the result of operations in an algebra, we can investigate *recognizable sets* of dependency structures (Mezei and Wright, 1967). The concept of algebraic recognizability provides a canonical notion of 'regularity' for sets of structures, be they strings, trees, graphs, or general relational structures (see Courcelle, 1996, for an overview). It is a fundamental concept with many different characterizations: it can be understood in terms of homomorphisms into finite algebras, automata, and definability in monadic second-order logic. In this paper, we use its characterization through *regular term grammars* (see Denecke and Wismath, 2001):

Definition 1 Let Σ be an X-sorted alphabet, and let $x \in X$ be a sort. A *regular term grammar* is a construct $G = (N, \Sigma, S, P)$, where N is an X-indexed family of alphabets of *non-terminal symbols*, $S \in N_x$ is a distinguished *start symbol*, and P is a finite set of *productions* of the form $A \to t$, where $A \in N_y$ and $t \in T^y_\Sigma(N)$, for some sort $y \in X$.

Given that the conversion between dependency structures and terms over the signature Ω of local order annotations is one-to-one (Kuhlmann and Möhl, 2007), recognizable dependency languages are isomorphic to their corresponding term languages. This motivates the following definition:

[1] We ignore the aspect of labelling for the sake of simplicity.

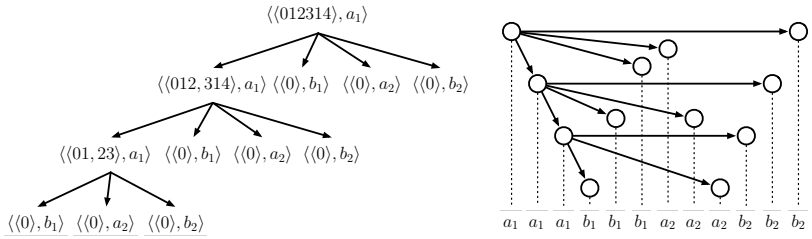

FIGURE 2 Regular dependency grammars

Definition 2 A dependency language over Σ and A is called *regular*, if its encoding as a set of terms over $\langle \Omega, A \rangle$ is generated by some regular term grammar.

In slight abuse of terminology, we refer to regular term grammars over (subsets of) $\langle \Omega, A \rangle$ as *regular dependency grammars*. The class of all regular dependency languages is denoted by REGD. Note that, while the set Ω of all order annotations is infinite, regular dependency grammars require us to get by with a finite subset of these annotations per language. In particular, every regular dependency language is built on a finite set of sorts, corresponding to the numbers of blocks per child in the order annotations and composition operations.

Example 5 To illustrate the definition of regular dependency grammars, we present a grammar that generates a language L with

$$sL = \textsc{count}(2)\,, \quad \text{where} \quad \textsc{count}(k) := \{\, a_1^n b_1^n \cdots a_k^n b_k^n \mid n \geq 1 \,\}\,.$$

Note that, for $k = 1$, $\textsc{count}(k)$ is homomorphic to the familiar context-free language $a^n b^n$, and that for every $k > 1$, $\textsc{count}(k)$ is not context-free. We present the productions of a grammar with start symbol S.

$$S \to \langle\langle 012314\rangle, a_1\rangle(R, B_1, A_2, B_2) \qquad S \to \langle\langle 0123\rangle, a_1\rangle(B_1, A_2, B_2)$$
$$R \to \langle\langle 012, 314\rangle, a_1\rangle(R, B_1, A_2, B_2) \qquad R \to \langle\langle 01, 23\rangle, a_1\rangle(B_1, A_2, B_2)$$
$$A_2 \to \langle\langle 0\rangle, a_2\rangle \qquad B_i \to \langle\langle 0\rangle, b_i\rangle\,, \quad \text{for } i \in [2].$$

Figure 2 shows a term generated by this grammar, and its corresponding dependency structure. Note that the structure is well-nested.

The construction in the example can be extended to obtain a regular dependency grammar for every language $\textsc{count}(k)$, $k \in \mathbb{N}$. In this construction, we need exactly two sorts, 1 and k. The usage of the sort k implies, that the resulting dependency language contains structures with block-degree k or, equivalently, gap-degree $k-1$. In the next section, we show that $\textsc{count}(k)$ actually *enforces* these structures.

1.4 Main results

In this section, we present the main technical results of this paper: that both the gap-degree parameter and the well-nestedness condition have immediate consequences for the string-generative capacity of regular dependency languages. The proof is based on two technical lemmata.

1.4.1 Technical lemmata

The first lemma is a pumping lemma for regular term languages; it can be seen as the correspondent to Ogden's lemma for context-free string languages. Let $L \subseteq T_\Sigma$ be a term language, and let $t \in L$. A non-empty context $p \in C_\Sigma$ is called *pumpable*, if there is a context $c \in C_\Sigma$ and a term $t' \in T_\Sigma$ such that $t = c \cdot p \cdot t'$ and $c \cdot p^n \cdot t' \in L$, for all $n \geq 0$.

Lemma 1 *For every regular term language $L \subseteq T_\Sigma$, there is a constant $n_L \geq 1$ such that, if $t \in L$ and at least n_L nodes in t have been marked as distinguished, then there exists a pumpable context $p \in C_\Sigma$ in t such that at least one node of p is marked as distinguished.*

Proof. The proof is based on the observation that, for all terms t with rank at most n, if the number of distinguished nodes in t is greater than n^i, then there is at least one root-to-leaf path in t that visits at least $i + 1$ nodes that qualify as roots of a context containing distinguished nodes. By choosing i to be the number of non-terminals in a (normal form) grammar for L, we thus obtain a pumpable context. □

The second lemma formulates two elementary results about intervals of positions in strings. Let $w \in A^*$ be a string. A *mask* for w is a non-empty sequence $M = [i_1, j_1] \cdots [i_n, j_n]$ of intervals of positions in w such that $j_k < i_{k+1}$, for all $k \in [n-1]$. We call the intervals $[i, j]$ the *blocks* of the mask M, and use $|M|$ to denote their number. We write $B \in M$, if B is a block of M. Given a string w and a mask M for w, the set of *positions corresponding to* M is defined as

$$pos([i_1, j_1] \cdots [i_n, j_n]) \;:=\; \{\, i \in pos(w) \mid \exists k \in [n].\, i \in [i_k, j_k] \,\}\,.$$

Given a set P of positions in w, we write $\bar{P}$ for the set of remaining positions, and $[P]$ for the minimal mask for w such that $pos(M) = P$. We say that P *contributes* to a block B of some mask, if $P \cap B \neq \emptyset$. For masks M with an even number of blocks, we define the *fusion* of M as

$$F([i_1, j_1][i'_1, j'_1] \cdots [i_n, j_n][i'_n, j'_n]) \;:=\; [i_1, j'_1] \cdots [i_n, j'_n]\,.$$

Lemma 2 *Let $w \in A^*$ be a string, let M be a mask for w with an even number of blocks, and let P be a set of positions in w such that both P and $\bar{P}$ contribute to every block of M. Then $|[P]| \geq |M|/2$. Furthermore, if $|[P]| \leq |M|/2$, then $P \subseteq pos(F(M))$.*

Proof. For every block $B \in [P]$, let $n(B)$ be the number of blocks in M that B contributes to. We make two observations:

- Since P contributes to each block of M, $|M| \leq \sum_{B \in [P]} n(B)$.
- Since $\bar{P}$ contributes to each block of M, no block $B \in [P]$ can fully contain a block of M; therefore, $n(B) \leq 2$, for all blocks $B \in [P]$.

Putting these two observations together, we see that

$$|M| \;\leq\; \textstyle\sum_{B \in [P]} n(B) \;\leq\; \sum_{B \in [P]} 2 \;=\; 2 \cdot |[P]|\,.$$

For the second part of the lemma, let $M = [i_1, j_1][i'_1, j'_1] \cdots [i_n, j_n][i'_n, j'_n]$ and $[P] = [k_1, l_1] \cdots [k_n, l_n]$. Then, each block of $[P]$ contributes to exactly two blocks of M. More precisely, for each $h \in [n]$, the block $[k_h, l_h]$ of $[P]$ contributes to the blocks $[i_h, j_h]$ and $[i'_h, j'_h]$ of M. Because $\bar{P}$ also contributes to $[i_h, j_h]$ and $[i'_h, j'_h]$, $[k_h, l_h]$ is a proper subset of $[i_h, j'_h]$, which is a block of $F(M)$. Hence, $P \subseteq pos(F(M))$. □

1.4.2 String languages that enforce structural properties

We now show how certain families of string languages can be used to enforce structural properties in regular dependency languages. The first family that we consider is the family COUNT from the example above. For what follows, put $\text{REGD}(D) := \{\, L \in \text{REGD} \mid L \subseteq D \,\}$.

Lemma 3 *Let $k \in \mathbb{N}$. Every language $L \in \text{REGD}$ with $sL = \text{COUNT}(k)$ contains structures with block-degree at least k.*

Proof. Let $L \subseteq D_\Sigma$, and let $d_1 = (t_1, \vec{u}_1)$ be a dependency structure contained in L such that $|s(d_1)| \geq n_L$, where n_L is the constant from Lemma 1. Because of the one-to-one correspondence between the positions in the string $s(d_1)$ and the nodes in the term t_1, we have $|t_1| \geq n_L$. If we now mark all nodes in t_1 as distinguished, Lemma 1 tells us that there exist contexts $p, c \in C_\Sigma$ and a term $t' \in T_\Sigma$ with $t_1 = c \cdot p \cdot t'$ such that the 'pumped' term $t_2 := c \cdot p^2 \cdot t'$ is contained in tL. Hence, there must be a linearization $\vec{u}_2$ of the nodes in t_2 such that $d_2 := (t_2, \vec{u}_2) \in L$. Now, let P be the set of positions in $s(d_2)$ that correspond to the subterm $p \cdot t'$ of t_2, and hence to a yield $\lfloor u \rfloor$, and let M be the uniquely determined mask for $s(d_2)$ with $2k$ blocks in which each of the blocks covers $|s(d_2)|/2k$ positions. Since $c \cdot p \cdot t' \in tL$ and $c \cdot p^2 \cdot t' \in tL$, the context p contributes to $s(d_2)$ at least one occurrence of every symbol a_i, b_i, for $i \in [k]$. Hence, both P and $\bar{P}$ contribute to each block of M. With the first part of Lemma 2, we then deduce that $|[P]| \geq k$. By the one-to-one correspondence between the positions in $s(d_2)$ and the nodes in d_2, this means that $\lfloor u \rfloor$ is distributed over at least k blocks; hence, d_2 has block-degree at least k (gap-degree at least $k - 1$). □

To enforce ill-nested dependency structure, we choose the family

$$\textsc{resp}(k) \;:=\; \{\, a_1^m b_1^m c_1^n d_1^n \cdots a_k^m b_k^m c_k^n d_k^n \mid m, n \in \mathbb{N} \,\},$$

where $\textsc{resp}(2)$ is from Weir (1988). Similar to $\textsc{count}(k)$, $\textsc{resp}(k)$ can be generated by a regular dependency grammar of degree k.

Lemma 4 *Let $k \in \mathbb{N}$, $k > 1$. Every $L \in \textsc{regd}(D_k)$ with $sL = \textsc{resp}(k)$ contains structures that are not well-nested.*

Proof. Consider a dependency structure $d_1 = (t_1, \vec{u}_1) \in L$ with

$$s(d_1) \;=\; a_1^m b_1^m c_1^n d_1^n \cdots a_k^m b_k^m c_k^n d_k^n \,,$$

where $m = n = n_L$ is the constant from Lemma 1, and mark all nodes in t_1 that are labelled with symbols from $X := \{\, x_i \mid x \in \{a, b\}, i \in [k] \,\}$ as distinguished. Because of the one-to-one correspondence between the positions in $s(d_1)$ and the nodes in t_1, we thus have $n_L \cdot |X| \geq n_L$ marked nodes. Lemma 1 then tells us that there exist contexts $p, c \in C_\Sigma$ and a term $t' \in T_\Sigma$ such that $t_2 := c \cdot p^2 \cdot t'$ belongs to tL. Thus, there is a linearization $\vec{u}_2$ of the nodes in t_2 such that the dependency structure $d_2 := (t_2, \vec{u}_2)$ belongs to L. Let v be the root node of the context p in t_1 and let w be the root node of the instance of p in t_2 that is farther away from the root. As another consequence of Lemma 1, at least one node in p is labelled with a symbol $x \in X$.

We now show that the set of labels that occur in the subterm $p \cdot t'$ of t_1 equals X. Since both $s(d_1)$ and $s(d_2)$ are elements of $\textsc{resp}(k)$, all symbols $x' \in X$ must occur equally often as labels in p, and since at least one node of p is labelled with $x \in X$, each element of X occurs at least once as a label in p and hence also in $p \cdot t'$. To show the inclusion in the other direction, consider the set P of positions in $s(d_2)$ that are contributed by the yield $\lfloor w \rfloor$ that equals the set of nodes in $p \cdot t'$. Furthermore, let M be the (uniquely determined) mask for $s(d_2)$ with $2k$ blocks in which each block contains all the occurrences of one symbol in X. We now apply Lemma 2 to deduce that $|[P]| \geq k$; since $d_2 \in D_k$, we even have $|[P]| = k$, and $P \subseteq pos(F(M))$. Given that all positions in $pos(F(M))$ are labelled with symbols from X, all nodes in $\lfloor w \rfloor$ and consequently all nodes in $p \cdot t'$ are labelled with symbols in X.

We now apply Lemma 1 a second time, marking all elements of the set $Y := \{\, y_i \mid y \in \{c, d\}, i \in [k] \,\}$. Analogously to the first application, we deduce the existence of a pumpable context with root node v' such that the set of labels occurring in $\lfloor v' \rfloor$ equals Y. Since X and Y are set-wise disjoint, neither $v \trianglelefteq v'$ nor $v' \trianglelefteq v$; however, the yields $\lfloor v \rfloor$ and $\lfloor v' \rfloor$ interleave. Hence, the structure d_2 is not well-nested. □

1.4.3 A hierarchy theorem

We are now ready to present our main result, a hierarchy theorem for the string languages corresponding to regular dependency languages. We define $s\text{REGD}(D) := \{ sL \mid L \in \text{REGD}(D) \}$. From Example 5 and Lemmata 3 and 4, we get the following result:

Theorem 5 (Hierarchy theorem) *For every $k \in \mathbb{N}$,*

- $s\text{REGD}(D_k) \subsetneq s\text{REGD}(D_{k+1})$,
- $s\text{REGD}(D_k \cap D_{wn}) \subsetneq s\text{REGD}(D_{k+1} \cap D_{wn})$,
- $s\text{REGD}(D_k \cap D_{wn}) \subsetneq s\text{REGD}(D_k)$ *if* $k > 1$,
- $s\text{REGD}(D_{k+1} \cap D_{wn}) - s\text{REGD}(D_k) \neq \emptyset$.

This theorem recovers and relates the string-language hierarchies for Linear Context-Free Rewriting Systems (Weir, 1988) and Coupled Context-Free Grammars (Hotz and Pitsch, 1996).

1.5 Conclusion

In this paper, we have motivated the notion of regular dependency languages (Kuhlmann and Möhl, 2007) by the notion of algebraic recognizability and shown that, in this framework, both the gap-degree restriction and the well-nestedness condition have an immediate impact on the string-generative capacity: they induce two infinite hierarchies of ever more expressive dependency languages. These hierarchies are intimately related to the hierarchies obtained by several mildly context-sensitive grammar formalisms.

The close link between 'mild' forms of non-projectivity and notions of formal power provides a promising starting point for future work. Structural properties such as gap-degree and well-nestedness are immediately observable in dependency analyses (for example, in dependency treebanks), and may therefore offer an interesting alternative to a comparison of grammar formalisms based on the relatively weak notion of string-generative capacity, or the very formalism-specific notion of strong generative capacity (cf. Kallmeyer, 2006).

Acknowledgements The research reported in this paper is funded by the German Research Foundation. We thank the anonymous reviewers of the paper for their detailed comments.

References

Bodirsky, Manuel, Marco Kuhlmann, and Mathias Möhl. 2005. Well-nested drawings as models of syntactic structure. In *Tenth Conference on Formal Grammar and Ninth Meeting on Mathematics of Language*. Edinburgh, Scotland, UK.

Courcelle, Bruno. 1996. Basic notions of universal algebra for language theory and graph grammars. *Theoretical Computer Science* 163:1–54.

Denecke, Klaus and Shelly L. Wismath. 2001. *Universal algebra and applications in theoretical computer science*. Chapman and Hall/CRC.

Eisner, Jason and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 457–464. College Park, Maryland, USA.

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

Hotz, Günter and Gisela Pitsch. 1996. On parsing coupled-context-free languages. *Theoretical Computer Science* 161:205–233.

Kallmeyer, Laura. 2006. Comparing lexicalized grammar formalisms in an empirically adequate way: The notion of generative attachment capacity. In *International Conference on Linguistic Evidence*, pages 154–156. Tübingen, Germany.

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

Kuhlmann, Marco and Joakim Nivre. 2006. Mildly non-projective dependency structures. In *21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING-ACL), Main Conference Poster Sessions*, pages 507–514. Sydney, Australia.

McDonald, Ryan and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Eleventh Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 81–88. Trento, Italy.

Mezei, Jorge E. and Jesse B. Wright. 1967. Algebraic automata and context-free sets. *Information and Control* 11:3–29.

Neuhaus, Peter and Norbert Bröker. 1997. The complexity of recognition of linguistically adequate dependency grammars. In *35th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 337–343. Madrid, Spain.

Nivre, Joakim. 2006. *Inductive Dependency Parsing*, vol. 34 of *Text, Speech and Language Technology*. Dordrecht, The Netherlands: Springer-Verlag.

Weir, David J. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph.D. thesis, University of Pennsylvania, Philadelphia, Pennsylvania, USA.