Well-Nested Drawings as Models of Syntactic Structure*

Manuel Bodirsky[1], Marco Kuhlmann[2], and Mathias Möhl[2]

[1] Institut für Informatik, Humboldt-Universität zu Berlin, Germany
[2] Programming Systems Lab, Saarland University, Saarbrücken, Germany

Abstract. This paper investigates *drawings* (totally ordered forests) as models of syntactic structure. It offers a new model-based perspective on lexicalised Tree Adjoining Grammar by characterising a class of drawings structurally equivalent to TAG derivations. The drawings in this class are distinguished by a restricted form of non-projectivity (*gap degree at most one*) and the absence of interleaving substructures (*well-nestedness*). We demonstrate that well-nested drawings allow for efficient processing by defining a simple constraint language for them and presenting an algorithm that decides in polynomial time whether a formula in that constraint language is satisfiable on a well-nested drawing.

1 Introduction

There are two major approaches to formal accounts of the syntax of natural language, the proof-theoretic and the model-theoretic approach. Both aim at providing frameworks for answering the question whether a given natural language expression is grammatical. Their methodology, however, is rather different: In a proof-theoretic framework, one tries to set up a system of *derivation rules* (such as the rules in a context-free grammar) so that each well-formed natural language expression stands in correspondence with a derivation in that system. In contrast, in a model-theoretic framework, one attempts to specify a class of *models* for natural language expressions and a set of *constraints* on these models such that an expression is well-formed iff it has a model satisfying all the constraints. The main contribution of this paper is the characterisation of a class of structures that provides a new model-based perspective on Tree Adjoining Grammar (TAG; [2]), a well-known proof-theoretic syntactic framework.

Every syntactic framework needs to account for at least two dimensions of syntactic structure: derivation structure and word order. The derivation structure captures linguistic notions such as dependency and constituency—the idea that a natural language expression can be composed of smaller expressions. Statements about word order are needed to account for the fact that not all

* This paper is the extended version of an article that appears in the proceedings of the 10th Conference on Formal Grammar and the 9th Meeting on Mathematics of Language, Edinburgh, Scotland, UK, 2005 [1].

permutations of the words of a grammatical sentence are neccessarily grammatical themselves. One of the scales along which syntactic frameworks can vary is the flexibility they permit in the relationship between derivation structure and word order. Context-free grammars do not allow any flexibility at all; derivation structure determines word order completely. In mildly context-sensitive grammar formalisms like TAG or Combinatory Categorial Grammar [3], certain forms of discontinuous derivations are permitted ('crossed-serial dependencies'). Other frameworks, such as non-projective dependency grammar [4], allow for even more flexibility to account for languages with free word order.

In this paper we introduce *drawings*, a simple class of structures for which the relaxation of the relationship between derivation structure and word order can be easily measured (§ 2). There is a natural way in which TAG derivations can be understood as drawings (§ 3). We show that the class of drawings induced in this way can be identified by two structural properties: a restriction on the degree of word order flexibility and a global property called *well-nestedness*, which disallows interleaving subderivations. In combination, these two properties capture the 'structural essence' of TAG (§ 4). Finally, we demonstrate that this class of structures allows for efficient processing by defining a simple constraint language for it and presenting a polynomial satisfiability algorithm (§ 5). The paper concludes with a discussion of the relevance of our results and an outlook on future research (§ 6).

2 Drawings

We start by introducing some basic terminology. A *relational structure* is a tuple whose first component is a non-empty, finite set V of *nodes*, and whose remaining components are (in this paper) binary relations on V. The notation Ru stands for the set of all nodes v such that $(u, v) \in R$. We use the standard notations for the transitive (R^+) and reflexive transitive (R^*) closure of binary relations.

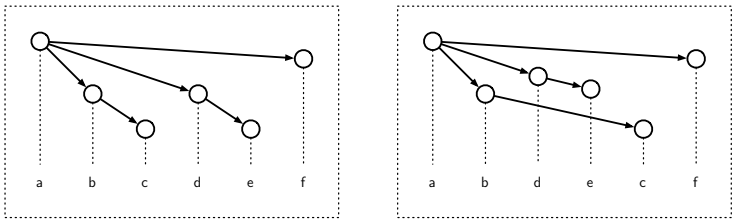
In this paper, we are concerned with two types of relational structures in particular: *forests* and *total orders*. A relational structure $(V; \lhd)$ is called a *forest* iff $\lhd$ is acyclic and every node in V has at most one predecessor with respect to $\lhd$. Nodes in a forest with no $\lhd$-predecessors are called *roots*. A *tree* is a forest that has exactly one root. A *total order* is a relational structure $(V; \prec)$ in which $\prec$ is transitive and for all $v_1, v_2 \in V$, exactly one of the following three conditions holds: $v_1 \prec v_2$, $v_1 = v_2$, or $v_2 \prec v_1$. Given a total order, the *interval* between two nodes v_1 and v_2 is the set of all v such that $v_1 \preceq v \preceq v_2$. The *cover* of a set $V' \subseteq V$, $\mathcal{C}(V')$, is the smallest interval containing V'. A set V' is *convex* iff it is equal to its cover. A *gap* in a set V' is a maximal, non-empty interval in $\mathcal{C}(V') - V'$. We call the number of gaps in a set the *gap degree* of that set and write $G_k(V)$ for the k-th gap in V (counted, say, from left to right).

2.1 Drawings and gaps

Drawings are relational structures with two binary relations: a forest to model derivation structure, and a total order to model word order.

Definition 1. *A* drawing *is a relational structure* $(V; \lhd, \prec)$ *where* $(V; \lhd)$ *forms a forest, and* $(V; \prec)$ *forms a total order. Drawings whose underlying forest is a tree will be called* arborescent*.*

Note that, in contrast to ordered forests (where order is defined on the direct successors of each node), order in drawings is *total*. By identifying each node v in a drawing with the set $(\lhd^*)v$ of nodes in the subtree rooted at v, we can lift the notions of cover and gap as follows: $\mathcal{C}(v) := \mathcal{C}((\lhd^*)v)$, $G_k(v) := G_k((\lhd^*)v)$. The gap degree of a drawing is the maximum among the gap degrees of its nodes.



(a) gap 0 (b) gap 1

Fig. 1: Two drawings

Fig. 1 shows two drawings of the same underlying tree. The circles and solid arcs reflect the forest structure. The dotted lines mark the positions of the nodes with respect to the total order. The labels attached to the dotted lines give names to the nodes. Drawing 1a has gap degree zero, since $\mathcal{C}(v) = (\lhd^*)v$ for all nodes v. In contrast, drawing 1b has gap degree one, since the set

$$\{d, e\} = \{b, d, e, c\} - \{b, c\} = \mathcal{C}(b) - (\lhd^*)b$$

is a gap for node b, and no other node has a gap.

2.2 Related work

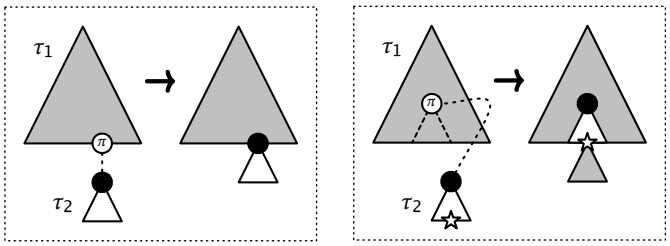
Our terminology can be seen as a model-based reconstruction of the terminology developed for non-projective dependency trees [4], where gaps are defined with respect to tree structures generated by a grammar. The notion of gap degree is closely related to the notion of *fan-out* in work on (string-based) finite copying parallel rewriting systems [5].

3 Drawings for TAG

Tree Adjoining Grammar (TAG) [2] is a proof-theoretic syntactic framework whose derivations manipulate tree structures. This section gives a brief overview of the formalism and shows how drawings model derivations in lexicalised TAGs.

3.1 Tree Adjoining Grammar

The building blocks of a TAG grammar are called *elementary trees;* they are successor-ordered trees in which each node has one of three types: *anchor* (or *terminal node*), *non-terminal node*, or *foot node*. Anchors and foot nodes must be leaves; non-terminal nodes may be either leaves or inner nodes. Each elementary tree can have at most one foot node. Elementary trees without a foot node are called *initial trees*; non-initial trees are called *auxiliary trees*. A TAG grammar is *strictly lexicalised*, if each of its elementary trees contains exactly one anchor.



(a) substitution (b) adjunction

Fig. 2: Combining tree structures in TAG

Trees in TAG can be combined using two operations (Fig. 2): *Substitution* combines a tree structure τ_1 with an initial tree τ_2 by identifying a non-terminal leaf node π of τ_1 with the root node of τ_2 (Fig. 2a). *Adjunction* identifies an inner node π of a structure τ_1 with the root node of an auxiliary tree τ_2; the subtree of τ_1 rooted at π is excised from τ_1 and inserted below the foot node of τ_2 (Fig. 2b; the star marks the foot node). Combing operations are disallowed at root and foot nodes.

TAG *derivation trees* record information about how tree structures were combined during a derivation. Formally, they can be seen as unordered trees whose nodes are labelled with elementary trees, and whose edges are labelled with the nodes at which the combining operations took place. If v is a node in a derivation tree, we write $\ell(v)$ for the label of v. An edge $v_1 - \pi \rightarrow v_2$ signifies that the elementary tree $\ell(v_2)$ was substituted or adjoined into the tree $\ell(v_1)$ at node π.

TAG *derived trees* represent results of derivations; we write $\mathsf{drv}(D)$ for the derived tree corresponding to a derivation tree D. Derived trees are ordered trees made up from the accumulated material of the elementary trees participating in the derivation. In particular, each TAG derivation induces a mapping ρ that maps each node v in D to the root node of $\ell(v)$ in $\mathsf{drv}(D)$. In strictly lexicalised TAGs, a derivation also induces a mapping α that maps each node v in D to the anchor of $\ell(v)$ in $\mathsf{drv}(D)$.

For derivation trees D in strictly lexicalised TAGs, we define

$$\begin{aligned} \mathsf{derived}(v) &:= \{\, \alpha(u) \mid v \lhd^* u \text{ in } D \,\} \quad \text{and} \\ \mathsf{yield}(v) &:= \{\, \pi \mid \pi \text{ is an anchor and } \rho(v) \lhd^* \pi \text{ in } \mathsf{drv}(D) \,\}. \end{aligned}$$

The set $\mathsf{derived}(v)$ contains those anchors in $\mathsf{drv}(D)$ that are contributed by the partial derivation starting at $\ell(v)$; $\mathsf{yield}(v)$ contains those anchors that are dominated by the root node of $\ell(v)$. To give a concrete example: Fig. 3 shows a TAG derivation tree (3a) and its corresponding derived tree (3b). For this derivation, we have

$$\mathsf{derived}(\mathit{like}) = \{\mathit{what}, \mathit{Dan}, \mathit{like}\} \quad \text{and} \quad \mathsf{yield}(\mathit{like}) = \mathsf{derived}(\mathit{like}) \cup \{\mathit{does}\}\,.$$

3.2 TAG drawings

There is a natural relation between strictly lexicalised TAGs and drawings: given a TAG derivation, one obtains a drawing by ordering the nodes in the derivation tree according to the left-to-right order on their corresponding anchors in the derived tree.

Definition 2. *Let D be a derivation tree for a strictly lexicalised* TAG*. A drawing $(V; \lhd, \prec)$ is a* TAG-drawing *iff (a) V is the set of nodes in D; (b) $v_1 \lhd v_2$ iff for some π, there is an edge $v_1 \mathrel{-}\pi\!\rightarrow v_2$ in D; (c) $v_1 \prec v_2$ iff $\alpha(v_1)$ precedes $\alpha(v_2)$ with respect to the leaf order in $\mathsf{drv}(D)$.*

Fig. 3c shows the TAG drawing induced by the derivation in Figs. 3a–b.

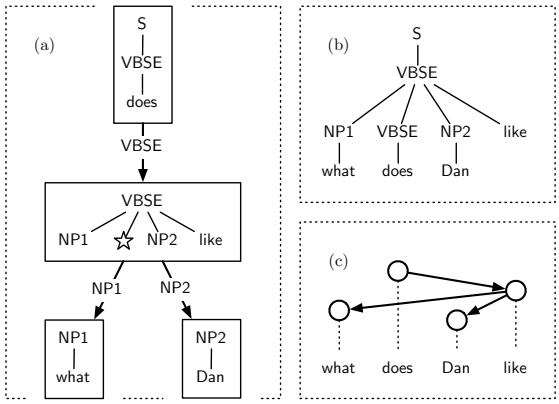


Fig. 3: TAG derivation trees (a), derived trees (b), and drawings (c)

4 The structural essence of TAG

Now that we have defined how TAG derivations induce drawings, we can ask whether *all* drawings (whose underlying forests are trees) are TAG drawings. The answer to this question is 'no': TAG drawings form a proper subclass in

the class of all drawings. As the major technical result of this paper, we will characterise the class of TAG drawings by two structural properties: a restriction on the gap degree and a property we call *well-nestedness* (Definition 3). The relevance of this result is that it provides a characterisation of 'TAG-ness' that does not make reference to any specific grammar, but refers to purely structural properties: well-nested drawings with gap degree at most one are 'just the right' models for TAG in the sense that every TAG derivation induces such a drawing, and for any such drawing we can construct a TAG grammar that allows for a derivation inducing that drawing.

4.1 TAG drawings are gap one

Gaps in TAG drawings correspond to adjunctions in TAG derivations: each adjunction may introduce material into the yield of a node that was not derived from that node. Since auxiliary trees have only one foot node, TAG drawings can have at most one gap.

Lemma 1. *Let D be a* TAG *derivation tree, and let v be a node in D. Then (a)* $\mathsf{derived}(v) \subseteq \mathsf{yield}(v)$*, (b)* $\mathsf{yield}(v) - \mathsf{derived}(v)$ *is convex, and (c)* $\mathsf{derived}(v)$ *contains at most one gap.*

Proof. (a) Each $a \in \mathsf{derived}(v)$ either is the anchor of $\ell(v)$, or has been derived from $\ell(v)$ in one or more steps. In both cases, a is dominated by the root node of $\ell(v)$ in the derived tree (Fig. 2). In particular, after each derivation step $v_1 - \pi \rightarrow v_2$, the root node of $\ell(v_1)$ dominates all anchors derived from the root node of $\ell(v_2)$ (Fig. 2a).

(b) Define $G := \mathsf{yield}(v) - \mathsf{derived}(v)$ and let a_l and a_r be the leftmost and rightmost anchor in G, respectively (assuming that G is non-empty). The only way by which an anchor can have entered G is by an adjunction of $\ell(v)$ into some other elementary tree (Fig. 2b). Now assume that G was not convex, i.e., there is an anchor $a \in \mathsf{derived}(v)$ such that $a_l \prec a \prec a_r$. Since both a_l and a_r are dominated by the foot node of $\ell(v)$, a is dominated by that node as well. This is a contradiction: neither can an anchor be dominated by the foot node of its own elementary tree (the foot node always is a leaf), nor can the foot node be the starting node of a sub-derivation (substitution and adjunction are disallowed at foot nodes). Thus, G is convex.

(c) The third item follows from the preceding two and the observation that $\mathsf{yield}(v)$ is convex.

Corollary 1. TAG *drawings have gap degree at most one.*

4.2 TAG drawings are well-nested

The gap restriction alone is not sufficient to characterise TAG drawings: there are drawings with gap degree one that cannot be induced by a TAG. Fig. 4 shows two examples. To see why these drawings cannot be induced by a TAG notice that in both of them, the cover of two nodes overlap ($\mathcal{C}(b)$ and $\mathcal{C}(c)$ in the left drawing,

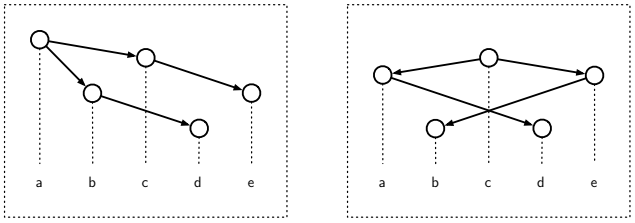


Fig. 4: Two drawings that are not well-nested

$\mathcal{C}(a)$ and $\mathcal{C}(e)$ in the right one). Since each node in a drawing corresponds to a sub-derivation on the TAG side, this would require the overlap of two yields in the derived tree, which is impossible. The present section will make this statement precise.

Definition 3. *Let T_1, T_2 be disjoint trees in a drawing. We say that T_1 and T_2* interleave *iff there are nodes $l_1, r_1 \in T_1$ and $l_2, r_2 \in T_2$ such that $l_1 \prec l_2 \prec r_1 \prec r_2$. A drawing is called* well-nested *iff it does not contain any interleaving trees.*

Well-nestedness is a purely structural property: it does not make reference to any particular grammar at all. In this respect, it is similar to the condition of *planarity* [6]. In fact, one obtains planarity instead of well-nestedness from Definition 3 if the disjointness condition is relaxed such that T_2 may be a subtree of T_1, and l_1, r_1 are chosen from $T_1 - T_2$.

Lemma 2. TAG *drawings are well-nested.*

Proof. Let D be a derivation tree. Imagine the TAG drawing for D, and assume that it contains two interleaving subtrees T_1 and T_2 with witnessing nodes l_1, r_1 and l_2, r_2. We will show that this leads to a contradiction. Let v_1 and v_2 be the root nodes of T_1 and T_2, respectively. The witnessing nodes define two overlapping intervals in the yield of $\mathsf{drv}(D)$: $\alpha(l_1) \cdots \alpha(r_1)$ and $\alpha(l_2) \cdots \alpha(r_2)$. Let π be an anchor present in both of these intervals. Since π is dominated in $\mathsf{drv}(D)$ by both $\rho(v_1)$ and $\rho(v_2)$, assume that $\rho(v_1)$ dominates $\rho(v_2)$ (the other case is symmetric). In this case, $\mathsf{yield}(v_1) \supseteq \mathsf{yield}(v_2)$. It follows that $l_2, r_2 \in H := \mathsf{yield}(v_1) - \mathsf{derived}(v_1)$. Because of $r_1 \in \mathsf{derived}(v_1)$, r_1 is not included in H; therefore, the premise $l_2 \prec r_1 \prec r_2$ implies that H is not convex. This contradicts Lemma 1(b).

4.3 Gap forests

Define a binary relation G on trees in a drawing such that $(T_1, T_2) \in G$ iff nodes from T_1 contribute to a gap in T_2. For well-nested drawings, this relation is acyclic and leads to the notion of the *gap forest*: the gap forest for v represents information about G for v and its successors.

Definition 4. *Let* $(V; \lhd, \prec)$ *be a well-nested drawing and* $v \in V$ *a node with* g *gaps. The* gap forest for v *is defined as the ordered forest* $\mathfrak{gf}(v) = (S; \sqsupset, <)$ *where*

$$\begin{aligned} S &:= \{\{v\}, G_1(v), \ldots, G_g(v)\} \cup \{\, \lhd^* w \mid v \lhd w \,\} \\ \sqsupset &:= \textit{transitive reduction of } \{\, (s_1, s_2) \in S \times S \mid \mathcal{C}(s_1) \supset s_2 \,\} \\ < &:= \{\, (s_1, s_2) \in S \times S \mid \forall v_1 \in s_1 \colon \forall v_2 \in s_2 \colon v_1 \prec v_2 \,\} \end{aligned}$$

The elements of S *are called* spans.

In a gap forest, sibling spans correspond to disjoint sets representing parts of gaps in the yield of their parent span. Sibling spans belonging to the same gap are called *span groups*. Since v has g gaps, each union of sibling spans has a gap degree that is bounded by $g - 1$. Dpans corresponding to the gaps in the yield of v ($G_i(v)$) and the singleton span $\{v\}$ are leaves in $\mathfrak{gf}(v)$.

4.4 Constructing a TAG grammar for a drawing

To complete our characterisation of TAG drawings, we now present an algorithm that takes a well-nested drawing with gap degree at most one and constructs a TAG grammar whose only derivation induces the original drawing. Correctness of the algorithm establishes the following

Lemma 3. *Each well-nested arborescent drawing that has gap degree at most one is a* TAG *drawing.*

The algorithm (see Algorithm 1) works as follows: It is called with the root node of the drawing and the start symbol of the resulting TAG grammar. It then performs a pre-order traversal of the tree structure underlying the drawing and generates one elementary tree for each of its nodes v. This elementary tree must offer adjunction and substitution sites for the elementary trees of the children of v in the order and gap inclusion relation specified by the drawing. These requirements are satisfied by the gap forest of v; therefore, the algorithm first computes $\mathfrak{gf}(v)$ (line 1), adds a new node representing the cover of v, and transforms the resulting tree into a TAG elementary tree (lines 2–3). During this transformation, both tree structure and order remain untouched; only the nodes are renamed: the anchor is named v, the gap is replaced by the foot node $\star$, and all other nodes obtain nonterminal labels that match the labels at the roots of the elementary trees generated in the recursive calls of the algorithm (line 5).

The combination of Lemmata 1, 2 and 3 implies

Theorem 1. *An arborescent drawing is a* TAG *drawing iff it is well-nested and has gap degree at most one.*

5 An efficient constraint solver

The previous section has shown that well-nestedness is an essential structural property of TAG drawings. We now define a simple constraint language for well-

Algorithm 1 ElementaryTree(v, n)

1: compute $\mathsf{gf}(v)$ and build a tree gt_v by adding a root node $\top$
2: $\mathsf{nt} := \{\{v\} \mapsto v, \top \mapsto n, G_1(v) \mapsto \star\} \cup \{\, \lhd^* w \mapsto m \mid v \lhd w, m \text{ fresh non-terminal} \,\}$
3: rename each node u of gt_v into $nt(u)$
4: **add** gt_v to the lexicon
5: **for each** w such that $v \lhd w$ **do** ElementaryTree$(w, \mathsf{nt}(w))$ **done**

nested drawings, and describe an algorithm that decides the satisfiability problem for this language in polynomial time. This algorithm can be used as a constraint propagation algorithm that enforces the 'TAG-ness' of a derivation. We also describe how to use the algorithm to efficiently list the set of all drawings on n vertices in a failure-free search.

5.1 Constraint language

A *constraint set* $\mathcal{C}$ consists of a set of variables $V = \{x_1, \ldots, x_n\}$, a total order $\prec$ on V, and a set C of constraints that are either of the form $x \lhd^+ y$ (*dominance constraint*), or of the form $x \perp y$ (*disjointness constraint*), for $x, y \in V$. A *solution* for a constraint set $\mathcal{C}$ is a well-nested drawing $(V; \lhd, \prec)$ defined on the variables of $\mathcal{C}$, such that $x_i \lhd^+ x_j$ holds in the drawing if the dominance constraint $x_i \lhd^+ x_j$ is in C, and where x_i and x_j lie in disjoint subtrees of the drawing if the disjointness constraint $x_i \perp x_j$ is in C. A constraint set with a solution is called *satisfiable*. The most fundamental computational problem for this constraint language is to determine whether a given constraint set is satisfiable. The main topic in the remainder of this section is an efficient algorithm for this task.

The set of dominance constraints can be seen as a directed graph $(V; \lhd^+)$. It is obvious that if this graph contains a directed cycle, there cannot be a solution for the corresponding constraint set. If the graph is acyclic and does not contain disjointness constraints, we can easily find a linear extension of $\lhd^+$ such that the corresponding drawing satisfies all the constraints, and is well-nested. Consider for example the constraint $a \lhd^+ c$, $d \lhd^+ b$ on the variables a, b, c, d with the linear order $a \prec b \prec c \prec d$. The constraint set has a well-nested solution; e.g. there is a drawing with $a \lhd^+ c \lhd^+ e \lhd^+ b \lhd^+ d$. However, if we add the disjointness constraint $b \perp c$ to C, the resulting constraint set becomes unsatisfiable.

5.2 Listing all drawings

We would like to enumerate all drawings on n vertices. To achieve this, one can use a search procedure that divides the set of such drawings in disjoint parts and *branches*, i.e., enumerates these parts recursively. Ideally, such a search procedure should not produce *failures*, i.e., all branches lead to at least one drawing. An efficient algorithm for checking consistency of a given constraint set can be used to obtain a failure-free search procedure for well-nested drawings.

Assuming that we have such an consistency algorithm (see § 5.3), we now describe this search procedure. First observe, that all different linear orders on the vertices lead to different drawings, and for each linear order there are corresponding drawings. Thus, our search initially branches over all different permutations of the vertices. Permutations on n elements can be listed efficiently. In the following we maintain a constraint set. In the beginning, this constraint set is empty. If the constraint set contains one of the constraints $a \lhd^+ b$, $b \lhd^+ a$, or $a \perp b$ for each pair of vertices $a, b \in V$, then these constraints together with the linear order on the variables fully determine the drawing on n vertices, and we can output it.

Otherwise, we find a pair of variables a, b without such a constraint. In every solution to the constraint, exactly one of the cases $a \lhd^+ b$, $b \lhd^+ a$, or $a \perp b$ holds. Hence, we distinguish between these three cases in our search. However, we want to be sure that we perform a failure-free search, i.e., that all of the cases considered in the search also contain a drawing. Therefore, we first add one of those constraints, and check with the algorithm discussed earlier whether the resulting larger constraint set still has a well-nested solution. If not, we do not consider this case in our search.

5.3 Checking consistency

We present an algorithm that checks for a given constraint set whether there is a well-nested drawing that satisfies all the constraints. For that we first adapt the fruitful notion of *freeness* for tree description constraint languages [7, 8] to drawings. Free variables can be used to decompose the constraint set into smaller parts and to efficiently solve these parts recursively.

Definition 5. *A variable $x \in \mathcal{C}$ is* free *iff $\mathcal{C}$ has a solution that is an arborescent drawing whose root is x.*

We now define the *dominance graph* of a constraint set $\mathcal{C}$, which we use to recursively decompose $\mathcal{C}$. The dominance graph is undirected and defined on the variables of $\mathcal{C}$. It contains an (undirected) edge between two variables x and y if either $x \lhd^+ y$ is in C, or if there are variables u and v such that $x \prec v \prec u \prec y$, $(x \lhd^+ u \in C \vee u \lhd^+ x \in C)$, and $(y \lhd^+ v \in C \vee v \lhd^+ y \in C)$. We use standard graph theoretic terminology for the dominance graph, such as *connectivity* and *connected components*. We freely use these notions also for the constraint set; e.g., we say that a constraint set is connected if its dominance graph is connected.

Lemma 4. *Every solution of a connected constraint set is arborescent.*

Proof. Let u be a root in a solution. All vertices on an undirected path in the dominance graph starting in u have to be below u. Since every vertex is connected to u via an undirected path, u dominates all vertices in the drawing.

This lemma and the definition of freeness imply that a connected constraint set without a free variable does not have a solution. We finally state some obvious necessary conditions for freeness. They can be checked efficiently and turn out to be crucial for the algorithm.

Lemma 5. *Let x be a free variable in a satisfiable constraint set. Then the following two properties hold: (P1) – There is no $y \in V$ such that $y \perp x \in C$. (P2) – There is no $y \in V$ such that $y \lhd^+ x \in C$.*

The algorithm splits into two recursive procedures Solve and SolveCon as follows. The underlying idea is to select a free variable, to decompose the constraint graph without the free variable into components, to recursively solve the parts, and to combine the partial solutions to a global one.

The following graph-theoretic notation will be convenient. If S is a subset of the variables of a constraint set $\mathcal{C}$, we denote with $\mathcal{C}[S]$ the *restriction* of $\mathcal{C}$ to the variables from S, i.e., the subset of constraints that only involves variables from S.

Algorithm 2 Solve($\mathcal{C}$)

1: **compute** the component S of the graph of C
containing the leftmost variable with respect to $\prec$
2: **let** U_1 be the drawing SolveCon($\mathcal{C}[S]$)
3: **let** U_2 be the drawing Solve($\mathcal{C}[V - S]$)
4: **return** the disjoint union of U_1 and U_2

Algorithm 3 SolveCon($\mathcal{C}$)

1: precondition: $\mathcal{C}$ contains a free variable
2: **if** no variable satisfying P1 or P2 exists **then return** 'there is no solution' **else choose** such a variable x
3: **add** the roots of the drawing Solve($\mathcal{C}[V - \{x\}]$) as children below x
4: **return** the drawing rooted at x

Theorem 2. *Procedure* Solve *outputs 'there is no solution' for a constraint set $\mathcal{C}$ if and only if the constraint set has no solution.*

Proof. The procedure Solve first computes the component S in the dominance graph of the given constraint set $\mathcal{C}$ that contains the leftmost variable. Then it invokes SolveCon for the constraint set $\mathcal{C}[S]$. For the constraint set induced by the remaining variables Solve is called. If both recursive calls produced a solution, their disjoint union is a solution for C: The constraints within the components are satisfied by inductive assumption. By definition, there are no dominance edges between different components. The disjointness constraints between different components are clearly also satisfied. Finally, the disjoint union

of U_1 and U_2 is well-nested, since by definition of the dominance graph all interleaving subtrees lie in the same connected component of the dominance graph.

Now we describe the procedure SolveCon. The given constraint set $\mathcal{C}$ has to satisfy the assumption that every solution is an arborescent drawing. This is guaranteed by the calls of SolveCon from the above procedure Solve, see Lemma 4. We can therefore also assume that every satisfiable instance contains a free variable. The procedure first searches for a variable x that satisfies P1 and P2. This can be checked easily in linear time. If there is no such variable, the procedure outputs 'there is no solution' and terminates (and indeed, for such a constraint there is no free node, and therefore no solution). Otherwise, the procedure Solve is called for the constraint set $\mathcal{C}[V - \{x\}]$. If this procedure call produces a solution F, we can return the arborescent drawing that results from introducing an edge from x to all roots in the drawing F. This drawing satisfies all constraints in C: The constraints within $\mathcal{C}[V - \{x\}]$ are satisfied by the inductive assumption. P2 asserts that there are no disjointness constraints incident to x. P1 asserts that there are no dominance constraints $y \lhd^+ x$ in C. Finally, all dominance constraints $x \lhd^+ y$ in C are satisfied by construction.

6 Conclusion

This paper introduced *drawings* as models of syntactic structure and presented a novel perspective on lexicalised TAG by characterising a class of drawings structurally equivalent to TAG derivations. The drawings in this class—we called them TAG drawings—have two properties: they have a *gap degree* of at most one and are *well-nested*. TAG drawings are suitable structures for a model-theoretic treatment of TAG.

We believe that our results can provide a new perspective on the treatment of languages with free word order in TAG. Since TAG's ability to account for word order variations is extremely limited, various attempts have been made to move TAG into a description-based direction.[3] Drawings allow us to analyse these proposals with respect to the question how they extend the class of *models* of TAG, and what new *descriptive means* they offer to talk about these models. We feel that these issues were not clearly separated in previous work on model-theoretic TAG [10, 11].

A model-theoretic approach to natural language processing lends itself to constraint-based processing techniques. We have started to investigate the computational complexity of constraint satisfaction problems on TAG drawings by defining a relevant constraint language and formulating a constraint solver that decides in polynomial time whether a formula in that language can be satisfied on a well-nested drawing. This solver can be used as a propagator in a constraint-based processing framework for TAG descriptions.

Our immediate future work will be concerned with the further development of our processing techniques into a model-based parser for TAGs. The current constraint solver propagates information about structures that are already known;

[3] Kallmeyer's dissertation [9] provides a comprehensive summary.

a full parser would need to construct these structures in the first place. In the longer term, we hope to characterise other proof-theoretic syntactic frameworks in terms of drawings, such as Multi-Component TAG and Combinatory Categorial Grammar.

Acknowledgements We are grateful to Alexander Koller, Guido Tack and an anonymous reviewer for useful comments on earlier versions of this paper. The work of Kuhlmann and Möhl is funded by the Collaborative Research Centre 378 'Resource-Adpative Cognitive Processes' of the Deutsche Forschungsgemeinschaft (DFG).

References

1. Bodirsky, M., Kuhlmann, M., Möhl, M.: Well-nested drawings as models of syntactic structure. In: 10th Conference on Formal Grammar and 9th Meeting on Mathematics of Language, Edinburgh, Scotland, UK (2005)
2. Joshi, A., Schabes, Y.: Tree Adjoining Grammars. In: Handbook of Formal Languages. Volume 3. Springer (1997) 69–123
3. Steedman, M.: The Syntactic Process. MIT Press (2001)
4. Plátek, M., Holan, T., Kuboň, V.: On relax-ability of word-order by d-grammars. In Calude, C., Dinneen, M., Sburlan, S., eds.: Combinatorics, Computability and Logic. Springer (2001) 159–174
5. Rambow, O., Satta, G.: Independent parallelism in finite copying parallel rewriting systems. Th. Computer Science **223** (1999) 87–120
6. Yli-Jyrä, A.: Multiplanarity – a model for dependency structures in treebanks. In: Second Workshop on Treebanks and Linguistic Theories, Växjö, Sweden (2003) 189–200
7. Bodirsky, M., Duchier, D., Niehren, J., Miele, S.: A new algorithm for normal dominance constraints. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA'04), New Orleans (2004) 59–67
8. Bodirsky, M., Kutz, M.: Pure dominance constraints. In: Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS'02). (2002) 287–298
9. Kallmeyer, L.: Tree Description Grammars and Underspecified Representations. PhD thesis, Universität Tübingen (1999)
10. Palm, A.: From constraints to TAGs: A transformation from constraints into formal grammars. In: Second Conference on Formal Grammar, Prague, Czech Republic (1996)
11. Rogers, J.: Syntactic structures as multi-dimensional trees. Research on Language and Computation **1** (2003) 265–305