

Generic Axiomatization of Families of Noncrossing Graphs in Dependency Parsing

Anssi Yli-Jyrä

University of Helsinki, Finland
anssi.yli-jyra@helsinki.fi

Carlos Gómez-Rodríguez

Universidade da Coruña, Spain
carlos.gomez@udc.es

Abstract

We present a simple encoding for unlabeled noncrossing graphs and show how its latent counterpart helps us to represent several families of directed and undirected graphs used in syntactic and semantic parsing of natural language as context-free languages. The families are separated purely on the basis of forbidden patterns in latent encoding, eliminating the need to differentiate the families of non-crossing graphs in inference algorithms: one algorithm works for all when the search space can be controlled in parser input.

1 Introduction

Dependency parsing has received wide attention in recent years, as accurate and efficient dependency parsers have appeared that are applicable to many languages. Traditionally, dependency parsers have produced syntactic analyses in tree form, including exact inference algorithms that search for maximum projective trees (Eisner and Satta, 1999) and maximum spanning trees (McDonald et al., 2005) in weighted digraphs, as well as greedy and beam-search approaches that forgo exact search for extra efficiency (Zhang and Nivre, 2011).

Recently, there has been growing interest in providing a richer analysis of natural language by going beyond trees. In semantic dependency parsing (Oepen et al., 2015; Kuhlmann and Oepen, 2016), the desired syntactic representations can have in-degree greater than 1 (re-entrancy), suggesting the search for maximum acyclic subgraphs (Schluter, 2014, 2015). As this inference task is intractable (Guruswami et al., 2011), noncrossing digraphs have been studied instead, e.g. by Kuhlmann and Johnsson (2015) who provide a $O(n^3)$ parser for maximum noncrossing acyclic subgraphs.

Yli-Jyrä (2005) studied how to axiomatize dependency trees as a special case of noncrossing digraphs. This gave rise to a new homomorphic representation of context-free languages that proves the classical Chomsky and Schützenberger theorem using a quite different internal language. In this language, the brackets indicate arcs in a dependency tree in a way that is reminiscent to encoding schemes used earlier by Greibach (1973) and Oflazer (2003). Cubic-time parsing algorithms that are incidentally or intentionally applicable to this kind of homomorphic representations have been considered, e.g., by Nederhof and Satta (2003), Hulden (2011), and Yli-Jyrä (2012).

Extending these insights to arbitrary noncrossing digraphs, or to relevant families of them, is far from obvious. In this paper, we develop (1) a *linear encoding* supporting general noncrossing digraphs, and (2) show that the encoded noncrossing digraphs form a *context-free language*. We then give it (3) two *homomorphic, nonderivative representations* and use the latent local features of the latter to characterize various families of digraphs.

Apart from the obvious relevance to the theory of context-free languages, this contribution has the practical potential to enable (4) *generic context-free parsers* that produce different families of noncrossing graphs with the same set of inference rules while the search space in each case is restricted with lexical features and the grammar.

Outline After some background on graphs and parsing as inference (Section 2), we use an **ontology of digraphs** to illustrate natural families of noncrossing digraphs in Section 3. We then develop, in Section 4, the first **latent context-free representation** for the set of noncrossing digraphs, then extended in Section 5 with additional latent states supporting our **finite-state axiomatization** of digraph properties, and allowing us to

control the search space using the lexicon. The **experiments** in Section 6 cross-validate our axioms and sample the growth of the constrained search spaces. Section 7 outlines the applications for practical parsing, and Section 8 concludes.

2 Background

Graphs and Digraphs A *graph* is a pair (V, E) where V is a finite set of vertices and $E \subseteq \{\{u, v\} \subseteq V\}$ is a set of edges. A sequence of edges of the form $\{v_0, v_1\}, \{v_1, v_2\}, \dots, \{v_{m-1}, v_m\}$, with no repetitions in v_1, \dots, v_m , is a *path* between vertices v_0 and v_m and *empty* if $m = 0$. A graph is a *forest* if no vertex has a non-empty path to itself and *connected* if all pairs of vertices have a path. A *tree* is a connected forest.

A *digraph* is a pair (V, A) where $A \subseteq V \times V$ is a set of arcs $u \rightarrow v$, thus a *directed* graph. Its *underlying graph*, (V, E_A) , has edges $E_A = \{\{u, v\} \mid (u, v) \in A\}$. A sequence of arcs $v_0 \rightarrow v_1, v_1 \rightarrow v_2, \dots, v_{m-1} \rightarrow v_m$, with no repetitions in v_1, \dots, v_m , is a *directed path*, and *empty* if $m = 0$.

A digraph without self-loops $v \rightarrow v$ is *loop-free* (property **DIGRAPH_{LF}**). We will focus on loop-free digraphs unless otherwise specified, and denote them just by **DIGRAPH** for brevity. A digraph is *d-acyclic* (**ACYC_D**), aka a *dag* if no vertex has a non-empty directed path to itself, *u-acyclic* (**ACYC_U**) aka a *m(ixed)-forest* if its underlying graph is a forest, and *weakly connected* (**w.c.**, **CONN_W**) if its underlying graph is connected.

Dependency Parsing The *complete digraph* $G_S(V, A)$ of a sentence $S = x_1 \dots x_n$ consists of vertices $V = \{1, \dots, n\}$ and all possible arcs $A = V \times V - \{(i, i)\}$. The vertex $i \in V$ corresponds to the word x_i and the arc $i \rightarrow j \in A$ corresponds to a possible dependency between the words x_i and x_j .

The task of *dependency parsing* is to find a constrained subgraph $G'_S(V, A')$ of the complete digraph G_S of the sentence. The standard solution is a rooted directed tree called a *dependency tree* or a dag called a *dependency graph*.

Constrained Inference In *arc-factored parsing* (McDonald et al., 2005), each possible arc $i \rightarrow j$ is equipped with a positive weight w_{ij} , usually computed as a weighted sum $w_{ij} = \mathbf{w} \cdot \Phi(S, i \rightarrow j)$ where \mathbf{w} is a weight vector and $\Phi(\mathbf{x}, i \rightarrow j)$ a feature vector extracted from the sentence \mathbf{x} , considering the dependency relation from word x_i to word x_j . Parsing then consists in finding an arc

subset $A' \subseteq A$ that gives us a constrained subgraph $(V, A') \in \text{Constrained}(V, A)$ of the complete digraph (V, A) with maximum sum of arc weights:

$$(V, A') = \underset{(V, A') \in \text{Constrained}(V, A)}{\text{arg max}} \sum_{i \rightarrow j \in A'} w_{i,j}.$$

The complexity of this inference task depends on the constraints imposed on the subgraph. Under no constraints, we simply set $A' = A$. Inference over dags is intractable (Guruswami et al., 2011). Efficient solutions are known for projective trees (Eisner, 1996), various classes of mildly non-projective trees (Gómez-Rodríguez, 2016), unrestricted spanning trees (McDonald et al., 2005), and both unrestricted and weakly connected non-crossing dags (Kuhlmann and Johnsson, 2015).

Parsimony Semantic parsers must be able to produce more than projective trees because the share of projective trees is pretty low (under 3%) in semantic graph banks (Kuhlmann and Johnsson, 2015). However, if we know that the parses have some restrictions, it is better to use them to restrict the search space as much as possible.

There are two strategies for reducing the search space. One is to develop a specialized inference algorithm for a particular natural language or family of dags, such as weakly connected graphs (Kuhlmann and Johnsson, 2015). The other strategy is to control the local complexity of digraphs through lexical categories (Baldrige and Kruijff, 2003) or equivalent mechanisms. This strategy produces a more sensitive model of the language, but requires a principled insight on how the complexity of digraphs can be characterized.

3 Constraints on the Search Space

We will now present a classification of digraphs on the basis of their formal properties.

The Noncrossing Property For convenience, graphs and digraphs may be ordered like in a complete digraph of a sentence. Two edges $\{i, j\}, \{k, l\}$ in an ordered graph or arcs $i \rightarrow j, k \rightarrow l$ in an ordered digraph are said to be *crossing* if $\min\{i, j\} < \min\{k, l\} < \max\{i, j\} < \max\{k, l\}$. A graph or digraph is *noncrossing* if it has no crossing edges or arcs. Noncrossing (di)graphs (**NC-(DI)GRAPH**) are the largest possible (di)graphs that can be drawn on a circle without crossing arcs. In the following, we assume that all digraphs and graphs are noncrossing.

An arc $x \rightarrow y$ is (properly) covered by an arc $z \rightarrow t$ if $(\{x, y\} \neq \{z, t\})$ and $\min\{z, t\} \leq \min\{x, y\} \leq \max\{x, y\} \leq \max\{z, t\}$.

Ontology Fig. 1 presents an ontology of such families of loop-free noncrossing digraphs that can be distinguished by digraphs with 5 vertices.

In the digraph ontology, a *multitree* aka mangrove is a dag with the property of being *strongly unambiguous* ($UNAMB_S$), which asserts that, given two distinct vertices, there is at most one repeat-free path between them (Lange, 1997).¹ A *polytree* (Rebane and Pearl, 1987) is a multitree whose underlying graph is a tree. The *out* property (OUT) of a digraph (V, E) means that no vertex $i \in V$ has two incoming arcs $\{j, k\} \rightarrow i$ s.t. $j \neq k$.

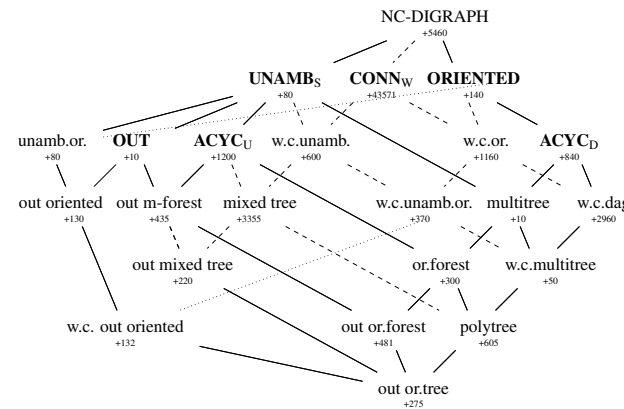


Figure 1: Basic properties split the set of 62464 noncrossing digraphs for 5 vertices into 23 classes

An ordered digraph is *weakly projective* ($PROJ_W$) if for all vertices i, j and k , if $k \rightarrow j \rightarrow i$, then either $\{i, j\} < k$ or $\{i, j\} > k$. In other words, the constraint, aka the *outside-to-inside constraint* (Yli-Jyrä, 2005), states that no outgoing arc of a vertex properly covers an incoming arc. This is implied by a stronger constraint known as *Harper, Hays, Lecerf and Ihm projectivity* (Marcus, 1967).

We can embed the ontology of graphs (unrestricted, connected, forests and trees) into the ontology of digraphs by viewing an undirected graph (V, E) as an inverse digraph $(V, \{(i, j), (j, i) \mid \{i, j\} \in E\})$. This kind of digraph has an *inverse property* (INV). Its opposite is an *oriented (or.)* digraph (V, A) where $i \rightarrow j \in A$ implies $j \rightarrow i \notin A$ (defines the property $ORIENTED$). Out forests and trees are, by convention, oriented digraphs with an underlying forest or tree, respectively.

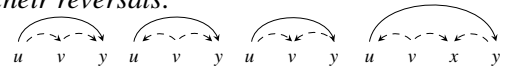
¹A different definition forbids diamonds as minors.

Distinctive Properties A few important properties of digraphs are local and can be verified by inspecting each vertex separately with its incident arcs. These include (i) the out property (OUT), (ii) the nonstandard projectivity property ($PROJ_W$), (iii) the inverse property (INV) and (iv) the orientedness (or.) property.

Properties $UNAMB_S$, $ACYC_D$, $CONN_W$, and $ACYC_U$ are nonlocal properties of digraphs and cannot be generally verified locally, through finite spheres of vertices (Grädel et al., 2005). The following proposition covers the configurations that we have to detect in order to decide the nonlocal properties of noncrossing digraphs.

Proposition 1. Let $G = (V, E)$ be a noncrossing digraph.

- If $G \notin UNAMB_S$, then the digraph contains one of the following four configurations or their reversals:



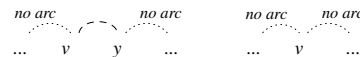
- If $G \notin ACYC_D$, then the graph contains one of the configurations



- If $G \notin ACYC_U$, then the underlying graph contains the following configuration:



- If $G \notin CONN_W$, then the underlying graph contains one of the following configurations:



Proposition 1 gives us a means to implement the property tests in practice. It tells us intuitively that although the paths can be arbitrarily long, any underlying cycle containing more than 2 arcs consists of one covering arc and a linear chain of edges between its end points.

4 The Set of Digraphs as a Language

In this section, we show that the set of noncrossing digraphs is isomorphic to an unambiguous context-free language over a bracket alphabet.

4.1 Basic Encoding

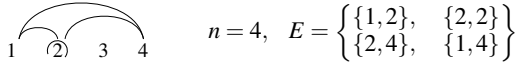
Any noncrossing ordered graph $([1, \dots, n], E)$, even with self-loops, can be encoded as a string of brackets using the algorithm `enc` in Fig. 2. For example, the output for the ordered graph

```

func enc(n,E):
  for i in [1,...,n]:
    for j in [i-1,...,2,1]:
      if {j,i} in E:
        print "]"
    for j in [n,n-1,...,i+1]:
      if {i,j} in E:
        print "["
  if {i,i} in E:
    print "["
  if i<n:
    print "{"
func dec(stdin):
  n = 1; E = {}; s = []
  while c in stdin:
    if c == "[":
      s.push(n)
    if c == "]":
      i = s.pop()
      E.insert((i,n))
    if c == "{":
      n = n + 1
  return (n,E)

```

Figure 2: The encoding and decoding algorithms



is the string $[[\{\}\][\{\}\{\}\]]$. Intuitively, pairs of brackets of the form $\{\}$ can be interpreted as spaces between vertices, and then each set of matching brackets $[\dots]$ encodes an arc that covers the spaces represented inside the brackets.

Any noncrossing ordered digraph $([1, \dots, n], A)$ can be encoded with slight modifications to the algorithm. Instead of printing $[\]$ for an edge $\{i, j\} \in E_A, i \leq j$, the algorithm should now print

```

/ > if (i,j) in A, (j,i) not in A;
< \ if (i,j) not in A, (j,i) in A;
[ ] if (i,j), (j,i) in A.

```

In this way, we can simply encode the digraph $(\{1, 2, 3, 4\}, \{(1, 2), (2, 2), (4, 1), (4, 2)\})$ as the string $</\{\}><[\{\}\{\}\]\backslash$.

Proposition 2. *The encoding respects concatenation where the adjacent nonempty operands have a common vertex.*

Context-Freeness Arbitrary strings with balanced brackets form a context-free language that is known, generically, as a Dyck language. It is easy to see that the graphs **NC-GRAPH** are encoded with strings that belong to the Dyck language D_2 generated by the context-free grammar: $S \rightarrow [S]S \mid \{S\}S \mid \epsilon$. The *encoded graphs*, $L_{\text{NC-GRAPH}}$, are, however, generated exactly by the context-free grammar $S \rightarrow [S']S \mid \{\}S \mid \epsilon, S' \rightarrow [S']T \mid \{\}S, T \rightarrow [S']S \mid \{\}S$. This language is an *unambiguous* context-free language.

Proposition 3. *The encoded graphs, $L_{\text{NC-GRAPH}}$, make an unambiguous context-free language.*

The practical significance of Proposition 3 is that there is a bijection between $L_{\text{NC-GRAPH}}$ and the derivation trees of a context-free grammar.

4.2 Bracketing Beyond the Encoding

Non-Derivational Representation A non-derivational representation for any context-free

language L has been given by Chomsky and Schützenberger (1963). This replaces the stack with a Dyck language D and the grammar rules with co-occurrence patterns specified by a regular language Reg . To hide the internal alphabet from the strings of the represented language, there is a homomorphism that cleans the internal strings of Reg and D from internal markup to get actual strings of the target language:

$$L_{\text{NC-GRAPH}} = h(D \cap Reg).$$

To make this concrete, replace the previous context free grammar by $S'' \rightarrow [S']S \mid \{\}S \mid \epsilon, S \rightarrow [S']S \mid \{\}S \mid \epsilon, S' \rightarrow [S']T \mid \{\}S, T \rightarrow [S']S \mid \{\}S$. The homomorphism h (Fig. 3a) would now relate this language to the original language, mapping the string $[[\{\}\][\{\}\{\}\]]'$ to the string $[[\{\}\][\{\}\{\}\]]$, for example. The Dyck language $D = D_3$ checks that the internal brackets are balanced, and the regular component Reg (Fig. 3b) checks that the new brackets are used correctly. A similar representation for the language $L_{\text{NC-DIGRAPH}}$ of encoded digraphs can be obtained with straightforward extensions.

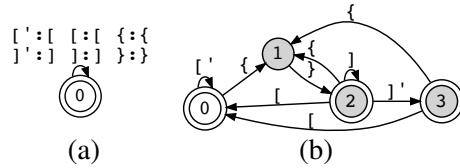


Figure 3: The h and Reg components

The representation $L = h(D \cap Reg)$ is *unambiguous* if, for every word $w \in L$, the preimage $h^{-1}(w) \cap D \cap Reg$ is a single string. This implies that L is an *unambiguous* context-free language.

Proposition 4. *The set of encoded digraphs, $L_{\text{NC-DIGRAPH}}$, has an unambiguous representation.*

Proposition 5. *Let $L_i = h(D \cap R_i), i \in \{0, 1, 2\}$ be unambiguous representations with $R_1, R_2 \subseteq R_0$. Then $L_3 = h(D \cap (R_1 \cap R_2))$ is an unambiguous context-free language and the same as $L_1 \cap L_2$.*

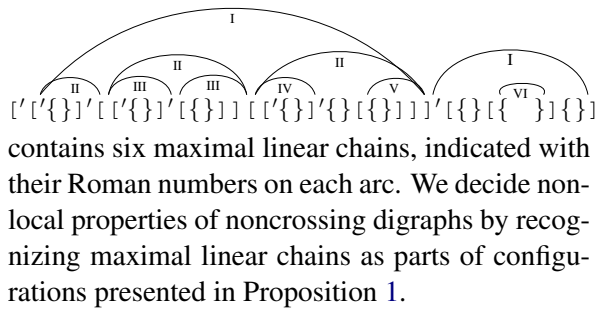
Proof. It is immediate that $L_3 \subseteq L_1 \cap L_2$ and L_3 is an unambiguous context-free language. To show that $L_1 \cap L_2 \subseteq L_3$, take an arbitrary $s \in L_1 \cap L_2$. Since $R_1, R_2 \subseteq R_0$ there is a unique $s' \in h^{-1}(s)$ such that $s' \in D \cap (R_1 \cap R_2)$. Thus $s \in L_3$. \square

5 Latent Bracketing

In this section, we extend the internal strings of the non-derivational representation of $L_{\text{NC-DIGRAPH}}$ in

such a way that the configurations given in Proposition 1 can be detected locally from these.

Classification of Underlying Chains A *maximal linear chain* is a maximally long sequence of one or more edges that correspond to an underlying left-to-right path in the underlying graph in such a way that no edge in this chain is properly covered by an edge that does not properly cover all the edges in the chain. For example, the graph



contains six maximal linear chains, indicated with their Roman numbers on each arc. We decide non-local properties of noncrossing digraphs by recognizing maximal linear chains as parts of configurations presented in Proposition 1. Every *loose* chain (like V and VI) starts with a bracket that is adjacent to a }-bracket. Such a chain can contribute only a covering edge to an underlying cycle. In contrast, a bracket with an apostrophe marks the beginning of a *non-loose* chain that can either start at the first vertex, or share starting point with a covering chain. When a non-loose chain is covered, it can be touched twice by a covering edge. The prefixes of chains are classified incrementally, from left to right, with a finite automaton (Figure 4). All states of the automaton are final and correspond to distinct classes of the chains. These classes are encoded to an extended set of brackets.

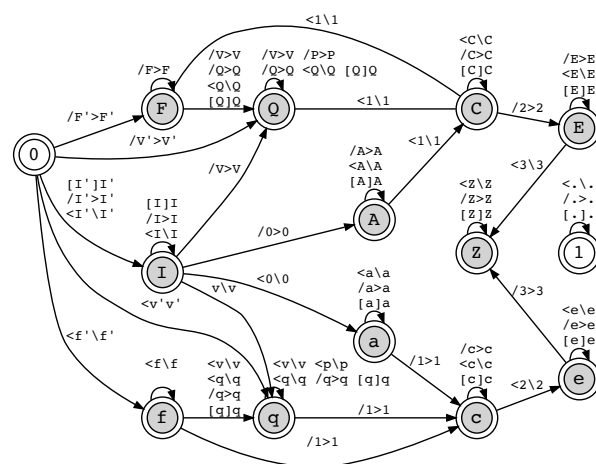


Figure 4: The finite automaton whose state 0 begins non-loose chains and state 1 loose chains

The automaton is symmetric: states with uppercase names are symmetrically related with

corresponding lowercase states. Thus, it suffices to define the initial and uppercase-named states:

- 0 the initial state for a non-loose chain;
- I a bidirectional chain: $u \leftrightarrow (v \leftrightarrow) y$;
- A a primarily bidirectional forward chain: $u \leftrightarrow v \rightarrow y$;
- F a forward chain: $u \rightarrow v \rightarrow y$;
- Q a primarily forward chain: $u \rightarrow v \leftrightarrow (\dots \rightarrow) y$;
- C a primarily forward 1-turn chain: $u \rightarrow v \leftarrow y$;
- E a primarily forward 2-turn chain: $u \rightarrow v \leftarrow x \rightarrow y$;
- Z a 3-turn chain;
- 1 the initial (and only) state for a loose chain;

Recognition of ambiguous paths in configurations $u \xrightarrow{\leftarrow} v \xrightarrow{\leftarrow} y$ and $u \xrightarrow{\leftarrow} v \rightarrow x \leftarrow \leftarrow y$ involves three chain levels. To support the recognition, subtypes of edges are defined according to the chains they cover. The brackets $>I'$, $\setminus I'$, $>I$, $\setminus I$, $\setminus A$, $>a$, $\setminus Q$, $>Q$, $>q$, $\setminus q$, $>C$, $\setminus c$, $\setminus E$, $>e$ indicate edges that constitute a cycle with the chain they cover. The brackets $>V'$, $\setminus V'$, $>V$, $\setminus V$ indicate edges that cover 2-turn chains. Not all states make these distinctions.

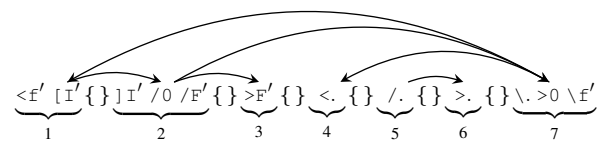
Extended Representation The extended brackets encode the latent structure of digraphs: the orientation and the subtype of the edge and the class of the chain. The total alphabet Σ of the strings now contains the boundary brackets $\{\}$ and 54 pairs of brackets (Figure 4) for edges from which we obtain a new Dyck language, D_{55} , and an extended homomorphism h_{lat} .

The *Reg* component of the language representation is replaced with Reg_{lat} , that is, an intersection of (1) an inverse homomorphic image of *Reg* to strings over the extended alphabet, (2) a local language that constrains adjacent edges according to Figure 4, (3) a local language specifying how the chains start, and (4) a local language that distinguishes pure oriented edges from those that cover a cycle or a 2-turn chain. The new component requires only 24 states as a deterministic automaton.

Proposition 6. $h_{lat}(D_{55} \cap Reg_{lat})$ is an unambiguous representation for $L_{NC-DIGRAPH}$.

The internal language $L_{NC-DIGRAPH_{lat}} = D_{55} \cap Reg_{lat}$ is called the set of *latent encoded digraphs*.

Example Here is a digraph with its latent encoding:



The brackets in the extended representation contain information that helps us recognize, through local patterns, that this graph has a directed cycle

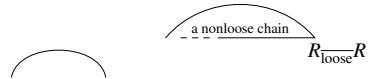
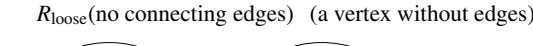
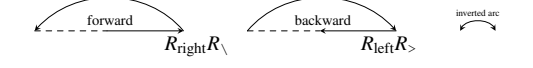
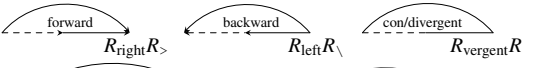
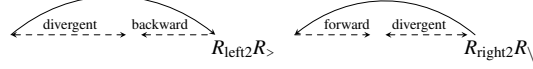

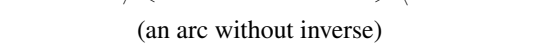
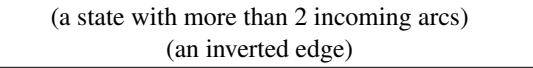
Forbidden patterns in noncrossing digraphs	Property	Constraint language
	ACYC_U	$A_U = \Sigma^* - \Sigma^* R_{\text{loose}} R \Sigma^*$
	CONN_W	$C_W = \Sigma^* - \Sigma^* R_{\text{loose}} (\varepsilon \cup B \Sigma^*) - (B \Sigma^* \cup \Sigma^* B)$
	ACYC_D	$A_D = \Sigma^* - \Sigma^* (R_{\text{right}} R_{\setminus} \cup R_{\text{left}} R_{>} \cup \Sigma_{\text{inv}}) \Sigma^*$
	UNAMB_S	$U_S = \Sigma^* - \Sigma^* (R_{\text{right}} R_{>} \cup R_{\text{left}} R_{\setminus} \cup R_{\text{vergent}} R) \Sigma^* - \Sigma^* (R_{\text{left}2} R_{>} \cup R_{\text{right}2} R_{\setminus}) \Sigma^*$
	PROJ_W	$P_W = \Sigma^* - \Sigma^* (L_{/} L_{<} \cup R_{>} R_{\setminus}) \Sigma^*$
	INV	$I = \Sigma^* - \Sigma^* \Sigma_{\text{or}} \Sigma^*$
	OUT	$Out = \Sigma^* - \Sigma^* \Sigma_{\text{in}} (\Sigma - B) \Sigma_{\text{in}} \Sigma^*$
	ORIENTED	$O = \Sigma^* - \Sigma^* \Sigma_{\text{inv}} \Sigma^*$

Table 1: Properties of encoded noncrossing digraphs as constraint languages

(directed path $1 \rightarrow 2 \rightarrow 7 \rightarrow 1$), is strongly ambiguous (two directed paths $2 \rightarrow 1$ and $2 \rightarrow 7 \rightarrow 1$) and is not weakly connected (vertices 5 and 6 are not connected to the rest of the digraph).

Expressing Properties via Forbidden Patterns

We now demonstrate that all the mentioned non-local properties of graphs have become local in the extended internal representation of the code strings $L_{\text{NC-DIGRAPH}}$ for noncrossing digraphs.

These distinctive properties of graph families reduce to forbidden patterns in bracket strings and then compile into regular constraint languages. These are presented in Table 1. To keep the patterns simple, subsets of brackets are defined:

$L_{/}$	$[-, / \text{-brackets}$	$L_{<}$	$[-, < \text{-brackets}$
$R_{>}$	$]-, > \text{-brackets}$	R_{\setminus}	$]-, \setminus \text{-brackets}$
B	$\{, \}$	R	$R_{>} \cup R_{\setminus}$
R_{loose}	$\}, >., \setminus.,]$	R_{loose}	$R - R_{\text{loose}}$
R_{right}	R reaching F, Q, I, A	R_{left}	R reaching f, q, i, a
$R_{\text{right}2}$	$>P, >2, >E, \setminus E,]E$	$R_{\text{left}2}$	$\setminus p, \setminus 2, \setminus e, >e,]e$
Σ_{in}	$L_{<} \cup R_{>}$	\bar{B}	$\Sigma - B$
R_{vergent}	non- $'$ R reaching I, Q, q, A, a, C, c		
Σ_{or}	all brackets for oriented edges		
Σ_{inv}	all brackets for inverted edges		

6 Validation Experiments

The current experiments were designed (1) to help in developing the components of Reg_{lat} and the constraint languages of axiomatic properties, (2) to validate the representation, the constraint languages and their unambiguity, (3) to learn about the ontology and (4) to sample the integer sequences associated with the cardinality of each

family in the ontology.

Finding the Components Representations of Reg_{lat} were built with scripts written using a finite-state toolkit (Hulden, 2009) that supports rapid exploration with regular languages and transducers.

Validation of Languages Our scripts presented alternative approaches to compute languages of encoded digraphs with n vertices up to $n = 9$. We also implemented a Python script that enumerated elements of families of graphs up to $n = 6$. The solutions were used to cross-validate one another.

The constraint $G_n = \bar{B}^* (\{\} \bar{B}^*)^{n-1}$ ensures n -vertices in encoded digraphs. The finite set of encoded acyclic 5-vertex digraphs was computed with a finite-state approach (Yli-Jyrä et al., 2012) that takes the input projection of the composition

$$\text{Id}(Reg_{\text{lat}} \cap A_D \cap G_5) \circ T_{55} \circ T_{55} \circ T_{55} \circ T_{55} \circ \text{Id}(\varepsilon)$$

where Id defines an identity relation and transducer T_{55} eliminates matching adjacent brackets. This composition differs from the typical use where the purpose is to construct a regular relation (Kaplan and Kay, 1994) or its output projection (Roche, 1996; Oflazer, 2003).

For digraphs with a lot of vertices, we had an option to employ a dynamic programming scheme (Yli-Jyrä, 2012) that uses weighted transducers.

Building the Ontology To build the ontology in Figure 1 we first found out which combinations of digraph properties co-occur to define distinguishable families of digraphs. After the nodes of the

lattice were found, we were able to see the partial order between these.

Integer Sequences We sampled, for important families of digraphs, the prefixes of their related integer sequences. We found out that each family of graphs is pretty much described by its cardinality, see Table 2. In many cases, the number sequence was already well documented (OEIS Foundation Inc., 2017).

7 The Formal Basis of Practical Parsing

While presenting a practical parser implementation is outside of the scope of this paper, which focuses in the theory, we outline in this section the aspects to take into account when applying our representation to build practical natural language parsers.

Positioned Brackets In order to do inference in arc-factored parsing, we incorporate weights to the representation. For each vertex in G_n , the brackets are decorated with the respective position number. Then, we define an input-specific grammar representation where each pair of brackets in D gets an arc-factored weight given the directions and the vertex numbers associated with the brackets.

Grammar Intersection We associate, to each G_n , a quadratic-size context-free grammar that generates all noncrossing digraphs with n vertices. This grammar is obtained by computing (or even precomputing) the intersection $D_{55} \cap Reg_{lat} \cap G_n$ in any order, exploiting the closure of context-free languages under intersection with regular languages (Bar-Hillel et al., 1961). The introduction of the position numbers and weights in the Dyck language gives us, instead, a weighted grammar and its intersection (Lang, 1994). This grammar is a compact representation for a finite set of weighted latent encoded digraphs. Additional constraints during the intersection tailors the grammar to different families of digraphs.

Dynamic Programming The heaviest digraph is found with a dynamic programming algorithm that computes, for each nonterminal in the grammar, the weight of the heaviest subtree. A careful reader may notice some connections to Eisner algorithm (Eisner and Satta, 1999), context-free parsing through intersection (Nederhof and Satta, 2003), and a dynamic programming scheme that

uses contracting transducers and factorized composition (Yli-Jyrä, 2012). Unfortunately, space does not permit discussing the connections here.

Lexicalized Search Space In practical parsing, we want the parser behavior and the dependency structure to be sensitive to the lexical entries or features of each word. We can replace the generic vertex description \bar{B}^* in G_n with subsets that depend on respective lexical entries. Graphical constraints can be applied to some vertices but relaxed for others. This application of current results gives a principled, graphically motivated solution to lexicalized control over the search space.

8 Conclusion

We have investigated the search space of parsers that produce noncrossing digraphs. Parsers that can be adapted to different needs are less dependent on artificial assumptions on the search space. Adaptivity gives us freedom to model how the properties of digraphs are actually distributed in linguistic data. As the adaptive data analysis deserves to be treated in its own right, the current work focuses on the separation of the parsing algorithm from the properties of the search space.

This paper makes four significant contributions.

Contribution 1: Digraph Encoding The paper introduces, for noncrossing digraphs, an encoding that uses brackets to indicate edges.

Bracketed trees are widely used in generative syntax, treebanks and structured document formats. There are established conversions between phrase structure and projective dependency trees, but the currently advocated edge bracketing is expressive and captures more than just projective dependency trees. This capacity is welcome as syntactic and semantic analysis with dependency graphs is a steadily growing field.

The edge bracketing creates new avenues for the study of connections between noncrossing graphs and context-free languages, as well as their recognizable properties. By demonstrating that digraphs can be treated as strings, we suggest that practical parsing to these structures could be implemented with existing methods that restrict context-free grammars to a regular yield language.

Contribution 2: Context-Free Properties Acyclicity and other important properties of noncrossing digraphs are expressible as unambiguous context-free sets of encoded noncrossing

Table 2: Characterizations for some noncrossing families of digraphs and graphs

Name	Sequence prefix for $n = 2, 3, \dots$	Example	Name	Sequence prefix for $n = 2, 3, \dots$	Example
digraph	(KJ): 4,64,1792, 62464 ,2437120,101859328 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat})$		weakly projective digraph	4,36,480, 7744 ,138880,2661376 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap P_W)$	
w.c. digraph	3,54,1539, 53298 ,2051406,84339468 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap C_W)$		w.p. w.c. digraph	3,26,339, 5278 ,90686,1658772 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap P_W \cap C_W)$	
unamb. digr.	4,39,529, 8333 ,142995,2594378 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap U_S)$		w.p. unamb. digr.	4,29,275, 3008 ,35884,453489 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap P_W \cap U_S)$	
m-forest	4,37,469, 6871 ,109369,1837396,32062711 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap A_U)$		w.p. m-forest	4,29,273, 2939 ,34273,421336 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap P_W \cap A_U)$	
out digraph	4,27,207, 1683 ,14229,123840,1102365 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap Out)$		w.p. out digraph	4,21,129, 867 ,6177,45840,350379 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap P_W \cap Out)$	
or. digraph	3,27,405, 7533 ,156735,3492639,77539113 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap O)$		w.p. or. digraph	see w.p. dag $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap P_W \cap O)$	see w.p. dag
dags	(A246756): 3,25,335, 5521 ,101551 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap A_D)$		w.p. dag	3,21,219, 2757 ,38523,574725,8967675 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap P_W \cap A_D)$	
w.c. dag	(KJ): 2,18,242, 3890 ,69074,1306466 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap A_D \cap C_W)$		w.p. w.c. dag	2,14,142, 1706 ,22554,316998,4480592 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap P_W \cap A_D \cap C_W)$	
multitree	3,19,167, 1721 ,19447,233283,2917843 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap A_D \cap U_S)$	see oriented forest or w.c. multitree	w.p. multitree	3,17,129, 1139 ,11005,112797,1203595 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap P_W \cap A_D \cap U_S)$	
or. forest	3,19,165, 1661 ,18191,210407,2528777 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap A_D \cup A_U)$		w.p. or. forest	3,17,127, 1089 ,10127,99329,1010189 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap P_W \cap A_D \cup A_U)$	
w.c. multitree	2,12,98, 930 ,9638,105798,1201062 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap A_D \cap U_S \cap C_W)$		w.p. w.c. multitree	2,10,68, 538 ,4650,42572,404354 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap P_W \cap A_D \cap U_S \cap C_W)$	
out or. forest	3,16,105, 756 ,5738,45088,363221 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap A_D \cap Out)$		w.p. out or. forest	(A003169): 3,14,79, 494 ,3294,22952 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap P_W \cap A_D \cap Out)$	
polytree	(A153231): 2,12,96, 880 ,8736,91392 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap A_D \cap C_W \cap A_U)$		w.p. polytree	(A027307): 2,10,66, 498 ,4066,34970 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap P_W \cap A_D \cap C_W \cap A_U)$	
out or. tree	(A174687): 2,9,48, 275 ,1638,9996 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap A_D \cap C_W \cap Out)$		projective out or tree	(A006013): 2,7,30, 143 ,728,3876,21318 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap P_W \cap A_D \cap C_W \cap Out)$	
graph	(A054726): 2,8,48, 352 ,2880,25216 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap I)$		connected graph	(A007297): 1,4,23, 156 ,1162,9192 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap I \cap C_W)$	
forest	(A054727): 2,7,33, 181 ,1083,6854 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap I \cap A_U)$		tree	(A001764,YJ): 1,3,12, 55 ,273,1428,7752 $h_{lat}(D_{55} \cap G_n \cap Reg_{lat} \cap I \cap A_U \cap C_W)$	

A = (OEIS Foundation Inc., 2017), **KJ** = Kuhlmann (2015) or Kuhlmann and Johnsson (2015), **YJ** = Yli-Jyrä (2012)

digraphs. This facilitates the incorporation of property testing to dynamic programming algorithms that implement exact inference.

Descriptive complexity helps us understand to which degree various graphical properties are local and could be incorporated into efficient dynamic programming during exact inference. It is well known that acyclicity and connectivity are not definable in first-order logic (FO) while they can be defined easily in monadic second order logic (MSO) (Courcelle, 1997). MSO involves set-valued variables whose use in dynamic programming algorithms and tabular parsing is inefficient. MSO queries have a brute force transformation to first-order (FO) logic, but this does not generally help either as it is well known that MSO can express intractable problems.

The interesting observation of the current work is that some MSO definable properties of digraphs become local in our extended encoding. This encoding is linear compared to the size of digraphs: each string over the extended bracket alphabet encodes a fixed assignment of MSO variables. The properties of noncrossing digraphs now reduce to properties of bracketed trees with linear amount of

```

func noncrossing_ACYCU(n,E):
  for {u,y} in E and u < y: # covering edge
    [v,p] = [u,u]
    while p != -1: # chain continues
      [v,p] = [p,-1]
      for vv in [v+1,...,y]: # next vertex
        if {v,vv} in E and {v,vv} != {u,y}:
          if vv == y:
            return False # found cycle uvv
          p = vv # find longest edge
  return True # acyclic

```

Figure 5: Testing $ACYCU$ in logarithmic space

latent information that is fixed for each digraph.

A deeper explanation for our observation comes from the fact that the treewidth of noncrossing and other outerplanar graphs is bounded to 2. When the treewidth is bounded, all MSO definable properties, including the intractable ones, become linear time decidable for individual structures (Courcelle, 1990). They can also be decided in a logarithmic amount of writable space (Elberfeld et al., 2010), e.g. with element indices instead of sets. By combining this insight with Proposition 1, we obtain a logspace solution for testing acyclicity of a noncrossing graph (Figure 5).

Although bounded treewidth is a weaker constraint than so-called bounded treedepth that would immediately guarantee first-order definabil-

ity (Elberfeld et al., 2016), it can sometimes turn intractable search problems to dynamic programming algorithms (Akutsu and Tamura, 2012). In our case, Proposition 1 gave rise to unambiguous context-free subsets of $L_{\text{NC-DIGRAPH}}$. These can be recognized with dynamic programming and used in efficient constrained inference when we add vertex indices to the brackets and weights to the grammar of the corresponding Dyck language.

Contribution 3: Digraph Ontology The context-free properties of encoded digraphs have elegant nonderivative language representations and they generate a semi-lattice under language intersection. Although context-free languages are not generally closed under intersection, all combinations of the properties in this lattice are context-free and define natural families of digraphs. The nonderivative representations for our axiomatic properties share the same Dyck language D_{55} and homomorphism, but differ in terms of forbidden patterns. As a consequence, also any conjunctive combination of these two properties shares these components and thus define a context-free language. The obtained semilattice is an ontology of families of noncrossing digraphs.

Our ontology contains important families of noncrossing digraphs used in syntactic and semantic dependency parsing: out trees, dags, and weakly connected digraphs. It shows the entailment between the properties and proves the existence of less known families of noncrossing digraphs such as strongly unambiguous digraphs and oriented graphs, multitrees, oriented forests and polytrees. These are generalizations of out oriented trees. However, these families can still be weakly projective. Table 2 shows integer sequences obtained by enumerating digraphs in each family. At least twelve of these sequences are previously known, which indicates that the families are natural.

We used a finite-state toolkit to build the components of the nongenerative language representation for latent encoded digraphs and the axioms.²

Contribution 4: Generic Parsing The fourth contribution of this paper is to show that parsing algorithms can be separated from the formal properties of their search space.

²The finite-state toolkit scripts and a Python-based graph enumerator are available at <https://github.com/amikael/ncdigraphs>.

All the presented families of digraphs can be treated by parsers and other algorithms (e.g. enumeration algorithms) in a uniform manner. The parser’s inference rules can stay constant and the choice of the search space is made by altering the regular component of the language representation.

The ontology of the search space can be combined with a constraint relaxation strategy, for example, when an out tree is a preferred analysis, but a dag is also possible as an analysis when no tree is strong enough. The flexibility applies also to dynamic programming algorithms that complement with the encoding and allow inference of best dependency graphs in a family simply by intersection with a weighted CFG grammar for a Dyck language that models position-indexed edges of the complete digraph.

Since the families of digraphs are distinguished by forbidden local patterns, the choice of search space can be made purely on lexical grounds, blending well with lexicalized parsing and allowing possibilities such as choosing, per each word, what kind of structures the word can go with.

Future work We are planning to extend the coverage of the approach by exploring 1-endpoint-crossing and MH_k trees (Pitler et al., 2013; Gómez-Rodríguez, 2016), and related digraphs — see (Yli-Jyrä, 2004; Gómez-Rodríguez et al., 2011). Properties such as *weakly projective*, *out*, and *strongly unambiguous* prompt further study.

An interesting avenue for future work is to explore higher order factorizations for noncrossing digraphs and the related inference. We would also like to have more insight on the transformation of MSO definable properties to the current framework and to logspace algorithms.

Acknowledgements

AYJ has received funding as Research Fellow from the Academy of Finland (dec. No 270354 - A Usable Finite-State Model for Adequate Syntactic Complexity) and Clare Hall Fellow from the University of Helsinki (dec. RP 137/2013). CGR has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 714150 - FASTPARSE) and from the TELEPARES-UDC project (FFI2014-51978-C2-2-R) from MINECO. The comments of Juha Kontinen, Mark-Jan Nederhof and the anonymous reviewers helped to improve the paper.

References

- Tatsuya Akutsu and Takeyuki Tamura. 2012. A polynomial-time algorithm for computing the maximum common subgraph of outerplanar graphs of bounded degree. In Branislav Rován, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012: 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, Springer Berlin Heidelberg, Berlin, Heidelberg, pages 76–87. https://doi.org/10.1007/978-3-642-32589-2_10.
- Jason Baldrige and Geert-Jan M. Kruijff. 2003. Multi-modal combinatory categorial grammar. In *Proceedings of EACL'03: the Tenth Conference on European Chapter of the Association for Computational Linguistics - Volume 1*. Association for Computational Linguistics, Budapest, Hungary, pages 211–218. <https://doi.org/10.3115/1067807.1067836>.
- Yehoshua Bar-Hillel, Micha Perles, and Eliahu Shamir. 1961. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonologie, Sprachwissenschaft und Kommunikationsforschung* 14:113–124.
- Noam Chomsky and Marcel-Paul Schützenberger. 1963. The algebraic theory of context-free languages. *Computer Programming and Formal Systems* pages 118–161.
- Bruno Courcelle. 1990. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Information and Computation* 85(1):12 – 75. [https://doi.org/10.1016/0890-5401\(90\)90043-H](https://doi.org/10.1016/0890-5401(90)90043-H).
- Bruno Courcelle. 1997. The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations*, World Scientific, New-Jersey, London, volume 1, chapter 5, pages 313–400.
- Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*. Copenhagen, Denmark, pages 340–345. <http://aclweb.org/anthology/C/C96/C96-1058.pdf>.
- Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and Head Automaton Grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, College Park, Maryland, USA, pages 457–464. <https://doi.org/10.3115/1034678.1034748>.
- Michael Elberfeld, Martin Grohe, and Till Tantau. 2016. Where first-order and monadic second-order logic coincide. *ACM Trans. Comput. Logic* 17(4):25:1–25:18. <https://doi.org/10.1145/2946799>.
- Michael Elberfeld, Andreas Jakoby, and Till Tantau. 2010. Logspace versions of the theorems of Bodlaender and Courcelle. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Washington, DC, USA, FOCS '10, pages 143–152. <https://doi.org/10.1109/FOCS.2010.21>.
- Carlos Gómez-Rodríguez. 2016. Restricted non-projectivity: Coverage vs. efficiency. *Computational Linguistics* 42(4):809–817. <https://doi.org/10.1162/COLLa.00267>.
- Carlos Gómez-Rodríguez, John A. Carroll, and David J. Weir. 2011. Dependency parsing schemata and mildly non-projective dependency parsing. *Computational Linguistics* 37(3):541–586. <https://doi.org/10.1162/COLLa.00060>.
- Erich Grädel, P. G. Kolaitis, L. Libkin, M. Marx, J. Spencer, Moshe Y. Vardi, Y. Venema, and Scott Weinstein. 2005. *Finite Model Theory and Its Applications (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.
- Sheila Greibach. 1973. The hardest context-free language. *SIAM Journal on Computing* 2(4):304–310. <https://doi.org/10.1137/0202025>.
- Venkatesan Guruswami, Johan Håstad, Rajsekar Manokaran, Prasad Raghavendra, and Moses Charikar. 2011. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM Journal on Computing* 40(3):878914. <https://doi.org/10.1137/090756144>.
- Mans Hulden. 2009. Foma: a finite-state compiler and library. In *Proceedings of the Demonstrations Session at EACL 2009*. Association for Computational Linguistics, Athens, Greece, pages 29–32. <http://www.aclweb.org/anthology/E09-2008>.
- Mans Hulden. 2011. Parsing CFGs and PCFGs with a Chomsky-Schützenberger representation. In Zygmunt Vetulani, editor, *Human Language Technology. Challenges for Computer Science and Linguistics: 4th Language and Technology Conference, LTC 2009, Poznan, Poland, November 6-8, 2009, Revised Selected Papers*, Springer Berlin Heidelberg, Berlin, Heidelberg, pages 151–160. https://doi.org/10.1007/978-3-642-20095-3_14.
- Ronald M. Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics* 20(3):331–378. <http://dl.acm.org/citation.cfm?id=204915.204917>.
- Marco Kuhlmann. 2015. Tabulation of non-crossing acyclic digraphs. arXiv:1504.04993. <https://arxiv.org/abs/1504.04993>.
- Marco Kuhlmann and Peter Johnsson. 2015. Parsing to noncrossing dependency graphs. *Transactions of the Association for Computational Linguistics* 3:559–570. <http://aclweb.org/anthology/Q/Q15/Q15-1040.pdf>.

- Marco Kuhlmann and Stephan Oepen. 2016. Towards a catalogue of linguistic graph banks. *Computational Linguistics* 42(4):819–827. <https://doi.org/10.1162/COLI.a.00268>.
- Bernard Lang. 1994. Recognition can be harder than parsing. *Computational Intelligence* 10(4):486–494. <http://onlinelibrary.wiley.com/doi/10.1111/j.1467-8640.1994.tb00011.x/full>.
- Klaus-Jörn Lange. 1997. An unambiguous class possessing a complete set. In Morvan Reichuk, editor, *STACKS'97 Proceedings*. Springer, volume 1200 of *Lecture Notes in Computer Science*. <http://dl.acm.org/citation.cfm?id=695352>.
- S. Marcus. 1967. *Algebraic Linguistics; Analytical Models*, volume 29 of *Mathematics in Science and Engineering*. Academic Press, New York and London.
- Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Vancouver, British Columbia, Canada, pages 523–530. <http://www.aclweb.org/anthology/H/H05/H05-1066.pdf>.
- Mark-Jan Nederhof and Giorgio Satta. 2003. Probabilistic parsing as intersection. In *8th International Workshop on Parsing Technologies*. LORIA, Nancy, France, pages 137–148.
- OEIS Foundation Inc. 2017. The on-line encyclopedia of integer sequences. <http://oeis.org>, read on 15 January 2017.
- Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinkova, Dan Flickinger, Jan Hajic, and Zdenka Uresova. 2015. SemEval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*. Association for Computational Linguistics, Denver, Colorado, pages 915–926. <http://www.aclweb.org/anthology/S15-2153>.
- Kemal Oflazer. 2003. Dependency parsing with an extended finite-state approach. *Computational Linguistics* 29(4):515–544. <https://doi.org/10.1162/089120103322753338>.
- Emily Pitler, Sampath Kannan, and Mitchell Marcus. 2013. Finding optimal 1-endpoint-crossing trees. *Transactions of the Association for Computational Linguistics* 1:13–24. <http://aclweb.org/anthology/Q13-1002>.
- George Rebane and Judea Pearl. 1987. The recovery of causal poly-trees from statistical data. In *Proceedings of the 3rd Annual Conference on Uncertainty in Artificial Intelligence (UAI 1987)*. Seattle, WA, pages 222–228. <http://dl.acm.org/citation.cfm?id=3023784>.
- Emmanuel Roche. 1996. Transducer parsing of free and frozen sentences. *Natural Language Engineering* 2(4):345–350. <https://doi.org/10.1017/S1351324997001605>.
- Natalie Schluter. 2014. On maximum spanning DAG algorithms for semantic DAG parsing. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*. Association for Computational Linguistics, Baltimore, MD, pages 61–65. <http://www.aclweb.org/anthology/W/W14/W14-2412.pdf>.
- Natalie Schluter. 2015. The complexity of finding the maximum spanning DAG and other restrictions for DAG parsing of natural language. In *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics*. Association for Computational Linguistics, Denver, Colorado, pages 259–268. <http://www.aclweb.org/anthology/S15-1031>.
- Anssi Yli-Jyrä. 2004. Axiomatization of restricted non-projective dependency trees through finite-state constraints that analyse crossing bracketings. In Geert-Jan M. Kruijff and Denys Duchier, editors, *COLING 2004 Recent Advances in Dependency Grammar*. COLING, Geneva, Switzerland, pages 25–32. <https://www.aclweb.org/anthology/W/W04/W04-1504.pdf>.
- Anssi Yli-Jyrä. 2005. Approximating dependency grammars through intersection of star-free regular languages. *Int. J. Found. Comput. Sci.* 16(3):565–579. <https://doi.org/10.1142/S0129054105003169>.
- Anssi Yli-Jyrä. 2012. On dependency analysis via contractions and weighted FSTs. In Diana Santos, Kristin Lindén, and Wanjiku Ng'ang'a, editors, *Shall We Play the Festschrift Game?, Essays on the Occasion of Lauri Carlson's 60th Birthday*. Springer, pages 133–158. https://doi.org/10.1007/978-3-642-30773-7_10.
- Anssi Yli-Jyrä, Jussi Piitulainen, and Aro Voutilainen. 2012. Refining the design of a contracting finite-state dependency parser. In Iñaki Alegria and Mans Hulden, editors, *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*. Association for Computational Linguistics, Donostia-San Sebastián, Spain, pages 108–115. <http://www.aclweb.org/anthology/W12-6218>.
- Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, Oregon, USA, pages 188–193. <http://www.aclweb.org/anthology/P11-2033>.