Approximating Dependency Grammars through Intersection of Regular Languages

Anssi Yli-Jyrä

Department of General Linguistics, University of Helsinki
P.O. Box 9, FIN-00014 University of Helsinki, Finland
anssi.yli-jyra@helsinki.fi

Abstract. The paper formulates projective dependency grammars in terms of constraints on a string based encoding of dependency trees and develops an approach to obtain a *regular approximation* for these grammars. In the approach, dependency analyses are encoded with balanced bracketing that encodes dependency relations among the words of the analyzed sentence. The brackets, thus, indicate dependencies rather than delimit phrases. The encoding allows expressing dependency rules (in the sense of Hays and Gaifman) using a semi-Dyck language and a so-called *context restriction operation*. When the semi-Dyck language in the representation is replaced with a regular restriction of it, we obtain an approximation for the original dependency grammar.

1 Introduction

Simple and efficient approaches to the task of dependency parsing, where dependency analyses are assigned to sentences, has recently attracted considerable attention ([1, 2] etc.). In this paper, we will show that parsing with ambiguous projective dependency grammars is an intersection problem. With a certain restriction, this leads to a linear-time restriction, implementable with finite automata.

A dependency analysis consists of a dependency tree (D-tree) whose nodes are, as assumed in this paper, words in a sentence. A D-tree shows which words are related to which other words and in what way. It shows the structure of the sentence in terms of hierarchical links between its actual elements. D-trees can be visualized using tree diagrams such as the one in Fig. 1.

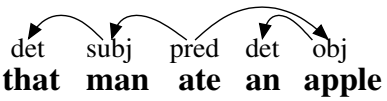


Fig. 1. A D-tree consists of dependency links drawn above a sentence. (This example is borrowed from (Oflazer 2003), but the arrows are drawn here in the opposite direction.)

The links denote syntactic dependencies and are represented by arcs with arrows and category labels. If $X \rightarrowtail Y$ is an arc between two words X and Y, we will say that Y depends immediately on X (or, conversely, X governs Y immediately), and that Y is an immediate syntactic dependent of X (or, X is the immediate syntactic governor of Y). In the D-tree, there is a unique non-governed word called the *root*. The category label of each link indicates *how* Y is dependent on X. Word Y can be *e.g.* a subject (subj), a object (obj) or a determiner (det) of X. (*Cf.* [3], p.14,23)

A D-tree is said to be *planar* (more precisely, *semi-planar*) if the links do not cross each other when drawn above the sentence [4]. Thus, the example in Fig. 1 is planar. A planar D-tree D is *projective* [5, 6] if it remains planar (Fig. 2) even if we add the so-called *left wall* node (//////) that governs the root of D (*cf.* [7], pages 99–100).

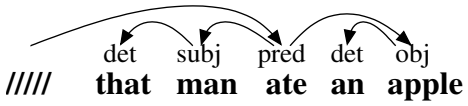


Fig. 2. The D-tree of Fig. 1 maintains planarity when the wall node is added.

In this paper, we will present a regular approximation of grammars that generate only projective dependency trees. The approximation is based on a new formulation of the Hays [8] and Gaifman [9] dependency grammars. The formulation represents dependency trees as bracketed strings. The correspondence between tree diagrams and their bracket-encoding is best understood through an example, such as the one shown in Fig. 3.

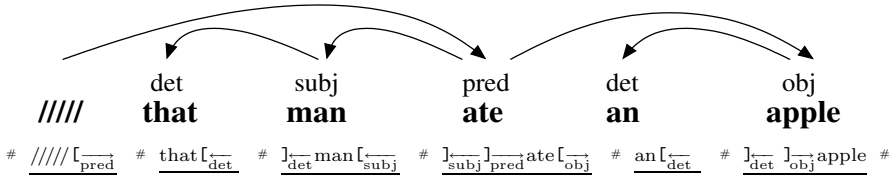


Fig. 3. The correspondence between a tree diagram and its encoding as a string. In order to facilitate reading, brackets belonging to the same tree node have been grouped with an underline that is not part of the encoding.

The string encoding of a D-tree involves the words of a sentence, labeled square brackets, a symbol (#) for word boundaries, and the wall node /////. Each bracket belongs to the word (or to the wall) within the closest surrounding word boundaries. The wall node ///// is not needed to enforce projectivity, but it will make all the word nodes have a governor link, which slightly simplifies the representation.

In the encoding, each arc of the D-tree is split into two parts. An opening bracket $[$ (with an additional label as a subscript) indicates the left end of the arc, and a closing bracket $]$ (with an additional label as a subscript) indicates the right end of the arc. The substring between these brackets must contain a balanced bracketing. A pair of matching brackets always indicate a link between two words in the sentence.

Let Λ be the set of category labels in D-trees. For each category label $a \in \Lambda$, the string encoding has two different labels for brackets. If *e.g.* 'subject' is a label in the D-tree, '$\overrightarrow{\text{subject}}$' and '$\overleftarrow{\text{subject}}$' are possible labels of brackets in the string encoding. Using the bracket label '$\overrightarrow{\text{subject}}$' indicates that the subject is at the closing bracket and using the bracket label '$\overleftarrow{\text{subject}}$' indicates that the subject is at the opening bracket.

The first important contribution of this paper is to point out certain less obvious properties of Robinson's [10] set of axioms for projective D-trees:

- The usual projectivity condition [6] is stronger than her fourth axiom.
- Her axioms do not themselves imply treeness (*cf.* *vs.* [11], p.4).
- Robinson premised the acyclicity property, but this nuance of the axiom set is often disregarded (*cf.* *e.g.* [12, 11, 2]).
- Antisymmetricity is a necessary consequence of non-crossing brackets that represent dependencies and a simple local condition.
- Projectivity follows from the placement and the uniqueness of the root node, irreflexivity, and non-crossing brackets.

The second important contribution of this paper is a reformulation of the generative dependency rules of the Hays and Gaifman dependency grammars (HGDG) as properties of bracketed string languages. The grammar representation makes use of a semi-Dyck language (*cf.* [13]) in order to capture non-local properties of bracketed strings. Because HGDGs generate context-free languages [9], we get from the intersection of these constraints a homomorphic representation for context-free languages. This new characterization resembles the famous Chomsky-Schützenberger theorem [14] that says that every context-free language is a homomorphic image of an intersection of a semi-Dyck language and a regular language.

Finally, the third important contribution of this paper is to present a linear-time parseable non-deterministic dependency grammar with restriction to limited projective dependency trees. This is obtained by replacing the semi-Dyck language with a regular language. A regular approximation for the semi-Dyck language can handle only limited balanced bracketing. Consequently, the intersection of constraints becomes a regular subset approximation for the grammar that does not limit the depth of balanced bracketing. The regular approximation grammar obtained can obviously be applied to sentences in linear time. In contrast to typical linear-time deterministic dependency parsers, our approach leaves the ambiguity unsolved. Moreover, our approach can be used for enumeration of valid D-trees and sentences.

2 Some Related Work

The applicability of finite-state methods to automatic syntactic analysis of natural language has been investigated in different approaches [15]. In pure *finite-state approaches*, finite-state devices are combined so that the whole system could be represented by a single finite-state machine, although it may be of an impractical size. In *extended finite-state approaches*, finite-state devices are used as basic components, but they are combined in such a way that the finite-state nature of the whole system is not necessarily retained.

Oflazer [2] has presented an interesting extended finite-state approach for dependency parsing of natural language. In particular, the string encoding used in our approach can be seen as a notational variant of Oflazer's representation. The representation used by Oflazer encodes the example in Fig. 1 as the string

```
<00(that)0d> <D0(man)0s> <S0(ate)00> <01(an)1d> <Do(apple)00>
```

where the stacked "brackets" d,D, O,o and s,S (corresponding to our $[_{\overleftarrow{\text{det}}}$, $]_{\overleftarrow{\text{det}}}$, $[_{\overrightarrow{\text{obj}}}$, $]_{\overrightarrow{\text{obj}}}$, $[_{\overleftarrow{\text{subj}}}$, $]_{\overleftarrow{\text{subj}}}$) are stored into so-called *channels*.

There exist cubic-time dependency parsers for HGDGs [16]. Most phrase types can be produced with a parser with quadratic time complexity [17]. Deterministic linear time dependency parsers have been studied *e.g.* in [1]. Constraint-based approaches to dependency parsing include Constraint Dependency Grammar [18] and Topological Dependency Grammar [19], which can actually generate non-context-free languages. Other approaches to assigning dependency structures to sentences include many lexicalized grammar formalisms that will not be listed here.

3 Hays and Gaifman Dependency Grammars

Tesnière's work [20] pioneered dependency-syntax grammars (DG). A formulation of more restricted dependency grammars given by Hays [8] and Gaifman [9] is only loosely related to Tesnière's theory, but it is still very influential in practical DG implementations. Their dependency grammar, HGDG, contains three kinds of rules by which the dependency analysis for a particular language is done:

1. Rules of the form $X(\star)$ that state that elements of the *word category* X may govern the sentence. (We adopt a short-hand notation $\star(\{X_1, X_2, \ldots, X_n\})$ for the set of these rules.)
2. Rules giving for every word category X the list of words belonging to it. These are of the form $X : \{w_1, w_2, \ldots, w_n\}$.
3. Rules which give for each word category X those categories which may derive directly from it with their relative positions. For each X there is a finite number of rules of the type $X(V_1 V_2 \ldots V_n \star Y_1 Y_2 \ldots Y_m)$. An application of this rule means that in the D-tree a word of the category X immediately governs words of categories $V_1, V_2, \ldots, V_n$ on the left of X and words of

the categories $Y_1, Y_2, \ldots, Y_m$ on the right. The governed words occur in the order in which their categories are specified in the rule. These rules are called *dependency rules*.

For details of the semantics of HGDGs, the reader is referred to Gaifman [9].

3.1 Axiomatization of Projective Dependency Trees

A famous mathematical axiomatization of the D-graphs generated by HGDGs is given by Robinson [10]. This axiomatization is usually regarded as a set consisting of the following axioms:

1. There is one and only one word that is independent.
2. All other words are immediately governed by some word.
3. No word depends immediately (i.e. directly) on more than one other word.
4. If word A depends immediately on word B and some word C intervenes between them (in the linear order of the words in the sentence), then C depends immediately on A or B or some other intervening element.

It should be noted that Robinson's fourth axiom is not equivalent to the usual condition of *projectivity* in the sense of Harper, Hays, Lecerf and Ihm (*cf.* [6]). The projectivity condition drops the underlined part in the fourth axiom and says instead that C depends transitively on A or B. In contrast to what is claimed in [1], neither of these conditions imply non-existence of cycles longer than one or two edges (Fig. 4). The reader can now easily proof that acyclicity

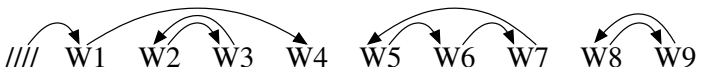


Fig. 4. A D-graph that conforms to Robinson's four axioms but violates acyclicity. The Harper, Hays, Lecerf and Ihm projectivity accepts the cycles at W5-W9, but not at W2-W3.

and connectedness of the dependency trees are not consequences of Robinson's set of axioms, contrary to what is often suggested [12, 11, 2]. It is often forgotten that Robinson included a crucial premise according to which the transitive closure of the immediate dependency relation will be (i) *irreflexive i.e.* without trivial cycles and (ii) *antisymmetric i.e.* without other cycles. When these extra requirements are taken as additional axioms, we are restricted to acyclic structures and the extended axiom set describes exactly the set of projective dependency trees.

4 The Essentials of the New Representation

4.1 The Alphabet

Assume that Λ is the set of category labels. We define four disjoint bracket sets as follows:

$$B_L = \{\, [_{\overrightarrow{a}} \mid a \in \Lambda\}; \qquad B_r = \{]_{\overrightarrow{a}} \mid a \in \Lambda\};$$
$$B_l = \{\, [_{\overleftarrow{a}} \mid a \in \Lambda\}; \qquad B_R = \{]_{\overleftarrow{a}} \mid a \in \Lambda\}.$$

The brackets with capital L and R subscripts attach to the governor and the brackets with small l and r subscripts to the dependent member of the pair. We will assume that Σ is the alphabet for building strings. It is the union of a number of disjoint subsets, namely the set of *word tokens* W, the *labeled left square brackets* $B_L \cup B_l$, the *labeled right square brackets* $B_R \cup B_r$, and the set of other special symbols $\{\#, ///// \}$.

The first string homomorphism $g : \Sigma^* \rightarrow \Sigma'^*$, where $[,] \notin \Sigma$ and $\Sigma' = \Sigma - (B_L \cup B_l \cup B_R \cup B_r) \cup \{\, [,] \,\}$, is defined in such a way that it essentially replaces labeled brackets with the corresponding unlabeled ones. The second string homomorphism $h : \Sigma'^* \rightarrow W^*$ is defined in such a way that it deletes from the strings all other symbols except the words W.

Obviously, the inverse homomorphism h^{-1} can be used to freely inject symbols $\{\#, /////, [,] \}$ into strings of W^*, and g^{-1} to replace [and] with various left and right square brackets. To parse a string $w \in W^+$, we will intersect the inverse homomorphic image $g^{-1}(h^{-1}(w))$ with the grammar G that is an intersection $C_1 \cap C_2 \cap \cdots C_n$ of constraint languages.

4.2 The Context Restriction Operation

A *context restriction of a center* $\mathcal{X}$ *in contexts* $\mathfrak{C}_1, \mathfrak{C}_2, \cdots, \mathfrak{C}_n$ is an operation whose first argument $\mathcal{X}$ is a subset of Σ^* and each context $\mathfrak{C}_i$, $1 \leq i \leq n$, is of the form $\mathcal{V}_i \_\_\_ \mathcal{Y}_i$, where $\mathcal{V}_i, \mathcal{Y}_i \subseteq \Sigma^*$. The operation is expressed using a notation

$$\mathcal{X} \Rightarrow \mathcal{V}_1 \_\_\_ \mathcal{Y}_1, \mathcal{V}_2 \_\_\_ \mathcal{Y}_2, \ldots, \mathcal{V}_n \_\_\_ \mathcal{Y}_n.$$

and it defines the set of all strings $w \in \Sigma^*$ such that, for every possible $v, y \in \Sigma^*$ and $x \in \mathcal{X}$, for which $w = vxy$, there exists some context $\mathcal{V}_i \_\_\_ \mathcal{Y}_i$, $1 \leq i \leq n$, where both $v \in \Sigma^* \mathcal{V}_i$ and $y \in \mathcal{Y}_i \Sigma^*$.

If all the languages involved in a context restriction are regular, the operation defines a regular language. In case $n = 1$, the language expressed by the operation is $\Sigma^* - ((\Sigma^* - \Sigma^* \mathcal{V}_1) \mathcal{X} \Sigma^* \cup \Sigma^* \mathcal{X} (\Sigma^* - \mathcal{Y}_1 \Sigma^*))$. Context restrictions with multiple contexts can also be routinely compiled into finite automata [21].

4.3 The Semi-Dyck Derivative and Its Regular Approximations

The *semi-Dyck language* (*cf.* [13]) over the alphabet $\{\, [,] \,\}$ is the language D_1 generated by the context-free grammar with single nonterminal S, two terminals

$[,]$ and the productions $S \rightarrow \epsilon \mid S\ [\ S\ ]\ S$. The regular language $D_{1,d}$ is an approximation of D_1, where the *depth d of bracketing* is bounded:

$$D_{1,d} = \begin{cases} \epsilon & \text{if } d = 0 \\ (D_{1,d-1} \cup (\ [\ D_{1,d-1}\ ]\ ))^* & \text{if } d > 0 \end{cases}$$

Let $f : \Sigma^* \rightarrow (B_L \cup B_r \cup B_l \cup B_R)^*$ be a string homomorphism that deletes all the other symbols except the square brackets. Obviously, the inverse homomorphism f^{-1} can be used to insert other symbols into the strings of square brackets.

The variable Δ can be given different kinds of values in the following ways:

$$\Delta \quad \leftarrow \quad f^{-1}(g^{-1}(D_1)) \tag{1}$$

$$\Delta \quad \leftarrow \quad f^{-1}(g^{-1}(D_{1,d})) \tag{2}$$

Assignment (1) makes Δ a context-free language and (2) makes it regular. The choice between (1) and (2) will determine whether the grammar in this paper gives exactly the power of HGDGs or whether it admits only a regular approximation for them. Our motivation to use variable Δ is that, in the second case, we actually get a neat finite-state equivalent formalism for a *regular subset* of context-free sets whose dependency structures are naturally described by non-finite-state formalisms.

In both cases, the language Δ is more liberal with respect to bracket labels compared to a semi-Dyck language based on an equivalent number of terminal symbols. Later on in this paper we will, however, employ a technique presented by Wrathall [22] in order to enforce matching bracket labels in aid of variable Δ and the context restriction operation (Wrathall used more elementary operations instead of context restriction).

Note that when Δ is defined to be a context-free derivative of the semi-Dyck language D, the obtained grammar representation will not be regular. Although context-free languages in general are not closed under relative complement that is used in context restrictions, all the constraints and their intersection will be context-free languages because of the "backbone" of balanced bracketing.

5 The Axiomatization of Bracketed Dependency Trees

5.1 The Basic Set of Strings with Balanced Brackets

In our encoding of D-trees, the sentence begins and ends with a word boundary # (3a), and the bracketing must be balanced (3b). These axioms are expressed as constraints:

$$\#\Sigma^*\#; \qquad\qquad \Delta. \tag{3}$$

Moreover, between each two word boundaries there exists at least one word, and two words are always separated by a word boundary. These axioms are expressed

by the following regular constraint languages:

$$\Sigma^* - \Sigma^*\#\,(\Sigma - (W \cup \{/////\}))^*\,\#\,\Sigma^*; \tag{4}$$
$$\Sigma^* - \Sigma^*\,(W \cup \{/////\})\,(\Sigma - \{\#\})^*\,(W \cup \{/////\})\,\Sigma^*\,.$$

In addition, the matching brackets must have equivalent labels. This is done through the following constraints that are inserted for each bracket label (*i.e.* word category) $a \in \Lambda$ (*cf.* [22]):

$$[_{\overrightarrow{a}} \;\Rightarrow\; \_\_\_\; \Delta\;]_{\overrightarrow{a}}; \qquad\qquad [_{\overleftarrow{a}} \;\Rightarrow\; \_\_\_\; \Delta\;]_{\overleftarrow{a}}. \tag{5}$$

5.2 The Properties of Projective Dependency Trees

We will now implement each of the requirements stated by Robinson by means of language properties. Our convention to use the wall node causes some unimportant modifications.

1. There is one and only one node that is independent, *i.e.*

$$\Sigma^*\#(\Sigma - \{\#\} \cup B_l \cup B_r)^*\#\Sigma^*; \tag{6}$$
$$\Sigma^* - \Sigma^*\#(\Sigma - \{\#\} \cup B_l \cup B_r)^*\#(\Sigma^*\#)^*(\Sigma - \{\#\} \cup B_l \cup B_r)^*\#\Sigma^*. \tag{7}$$

2. All word nodes except the wall node (/////) are immediately governed by some node:

$$W \Rightarrow\; (B_r \cup B_l)(\Sigma - \{\#\})^*\;\_\_\_\;,\;\_\_\_\;(\Sigma - \{\#\})^*(B_r \cup B_l). \tag{8}$$

3. No word depends immediately on more than one other word, *i.e.*

$$\Sigma^* \;-\; \Sigma^*(B_r \cup B_l)(\Sigma - \{\#\})^*(B_r \cup B_l)\Sigma^*. \tag{9}$$

4. There are no trivial cycles (irreflexivity), *i.e.*

$$\Sigma^* - \Sigma^*(B_l \cup B_L)(\Sigma - \{\#\})^*(B_r \cup B_R)\Sigma^*. \tag{10}$$

5. If Robinson's fourth axiom is violated between two words A and B, where A is immediately dependent on B, then at least one of the following cases must hold:
   (a) An intervening word C is the root of the sentence. This case can be excluded with the following constraint that requires that the root (in practice, the wall) is not an intervening word:

$$\#\;/////\;\Sigma^*. \tag{11}$$

   (b) An intervening word C is one of the independent words of the sentence ($\longrightarrow$ multiple roots). This case is excluded by Constraint (7).
   (c) An intervening word C is dependent on itself ($\longrightarrow$ violates irreflexivity). This case is excluded by Constraint (10).

(d) An intervening word C is governed by a word that is not A, B, C nor any other intervening word ($\longrightarrow$ a crossing edge). This case is excluded by Constraint (3b).

6. The simplest kind of nontrivial cycle contains two adjacent words. Such a case occurs if the bracketing has either of the following two patterns:

$$\cdots \ \# \ E \ [_{\overleftarrow{X}} \ [_{\overrightarrow{Y}} \ \ \# \ \ ]_{\overrightarrow{Y}} \ ]_{\overleftarrow{X}} \ F \ \ \# \cdots \quad \text{or} \quad \cdots \ \# \ F \ [_{\overrightarrow{Y}} \ [_{\overleftarrow{X}} \ \ \# \ \ ]_{\overleftarrow{X}} \ ]_{\overrightarrow{Y}} \ E \ \ \# \ \cdots$$

Observe that at the word F, the bracket indicating the category of the word itself is not adjacent to it. This means that the link from F to its governor is shorter than a link to one of its dependents that is in the same direction as the governor. In fact, every cycle containing at least two words must have such a word F. There are also other situations where the bracket indicating the category of a word is not adjacent to the word itself. These are exactly those cases in which an intervening element C governs two linked words A or B.

$$\Sigma^* - \Sigma^*(B_L B_l \cup B_r B_R)\Sigma^*. \tag{12}$$

Due to Constraint (10) we can adopt a convention that places closing brackets on the left side of each word and opening brackets on the right side by saying that

$$(W \cup \{/////\}) \Rightarrow \ \# \ (B_R \cup B_r)^* \ \_\_\_ (B_L \cup B_l)^* \ \#. \tag{13}$$

Because the bracketing used in the encoding is balanced, the bracket corresponding to the longest link will be placed closest to the word, and the bracket corresponding to the shortest link will be placed closest to the word boundary #. This conforms the same order that is used by Oflazer [2, page 524] when he allocates so called *channel symbol slots* in his representation.

6 The New Representation for the Grammars

We will now re-express all the rules of HGDGs using context restrictions. The rule listing the categories that can be independent is of the form $\star(\{X_1, X_2, \ldots, X_n\})$. This is expressed through the following regular constraint

$$///// \Rightarrow \ \_\_\_ \ \# \ \{ [_{\overrightarrow{X_1}}, [_{\overrightarrow{X_2}}, \cdots, [_{\overrightarrow{X_n}}\} \ \#. \tag{14}$$

The rules listing words $\{w_1, w_2, \ldots, w_n\}$ in each category X are of the form $X : \{w_1, w_2, \ldots, w_n\}$. In the presence of (13), these rules can be expressed by the following constraints:

$$[_{\overleftarrow{X}} \Rightarrow \{w_1, w_2, \ldots, w_n\} \_\_\_ ; \qquad]_{\overrightarrow{X}} \Rightarrow \_\_\_ \{w_1, w_2, \ldots, w_n\}. \tag{15}$$

Each dependency rule $X(V_1V_2 \ldots V_n \star Y_1Y_2 \ldots Y_m)$ specifying a set of dependents for category X corresponds to the context $\mathfrak{C}_{(V_1V_2 \ldots V_n \star Y_1Y_2 \ldots Y_m)}$:

$$\#\ ]_{\overleftarrow{V_n}}]_{\overleftarrow{V_{n-1}}} \cdots]_{\overleftarrow{V_1}}\ W^*\ \underline{\quad}\ W^*\ [_{\overrightarrow{Y_m}}[_{\overrightarrow{Y_{m-1}}} \cdots]_{\overrightarrow{Y_1}}\ \#.$$

When a word category X has n such contexts $\mathfrak{C}_1, \mathfrak{C}_2, \ldots, \mathfrak{C}_n$, their union corresponds to the following regular context restriction:

$$\{\ ]_{\overrightarrow{X}}, [_{\overleftarrow{X}}\ \} \Rightarrow \mathfrak{C}_1, \mathfrak{C}_2, \ldots, \mathfrak{C}_n. \tag{16}$$

For example, the dependency rule

$$\text{bitransitive(subject} \star \text{object indirect-object)}$$

will be represented using the following context restriction[1]:

$$\{\ ]_{\overrightarrow{\text{bitransitive}}}, [_{\overleftarrow{\text{bitransitive}}}\ \} \Rightarrow\ \#\ ]_{\overleftarrow{\text{subject}}}\ W^*\ \underline{\quad}\ W^*\ [_{\overrightarrow{\text{indirect-object}}}[_{\overrightarrow{\text{object}}}\ \#.$$

7 Discussion on Practical Applicability

As to the regular approximation that is obtained when an approximation $D_{1,d}$ of D_1 is assigned to the variable Δ, the most important practical question is: Can we actually use the obtained finite-state grammar to parse natural language sentences efficiently and accurately?

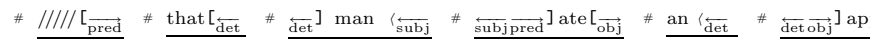
Parsing of the obtained approximation grammars means computing the intersection of the language $g^{-1}(h^{-1}(w))$ and the grammar constraints. Such a system can be seen as a special variant of Finite-State Intersection Grammar (FSIG) (*cf.* [15]), where the most striking problem has been to prevent intermediate results from blowing up in size when the intersection is computed. However, we have some reasons to be more optimistic with the current grammars than with FSIGs in general:

1. There are examples of so-far more successful extended finite-state approaches [2] (*cf.* also [15]), where bracketing at different depths is elaborated incrementally, according to a Bottom-Up or Top-Down parsing strategy. It seems that such strategies could be implemented also in the framework of FSIG by splitting each context restriction into sub-constraints [21].
2. The bracketing employed here represents local context-free trees, which gives rise to new ambiguity packing methods [23] based on parallel decompositions of automata. This may lead to improvements that narrow the distance between techniques for parsing through finite-state intersection and parsing with Chart-like data structures.
3. Most of the constraint languages presented here are locally testable, which entails that their intersection can be done without considerable difficulties. In our initial experiments, we applied them first and enforced the non-local constraints (3b) and (5) in a later stage.

[1] In the expression, the opening bracket for the indirect-object precedes the bracket of the object, because matching brackets obey the LIFO discipline.

At this stage our experiments are still very limited and they merely highlight that the proposed representation is implementable and can be used both in parsing and enumeration of valid sentences. We extracted two kinds of grammars from a portion of the Danish Dependency Treebank [24] (the second, smaller grammar was mainly hand-crafted). As word categories, we used syntactic functions in the first grammar and words themselves in the second one. These grammars represented two (almost) extreme ways to make generalizations from the available data. In both cases, the grammar constraints were given in a script to the XFST program [25]. Compiling a grammar with a few hundred rules into a set of separate automata took only one second. Intersection during parsing of both grammars was also quite fast because the constraint automata were small and only a few of them contributed to parsing of the actual input sentence.

In the bracketing scheme presented of the current paper, the number of dependents per node contributes directly to the depth of nested brackets. It is, however, possible to optimize bracketing depth in such cases by using so-called reduced bracketing. Accordingly, the bracketing of Fig. 3 can be replaced with the following bracketing:



Due to space limitations the intricate details of reduced dependency bracketing cannot be handled here. A scheme for reduced bracketing of dependencies and its extension to non-projective dependency trees appear in [26]. The problems related to grammar induction or extraction and accuracy cannot be discussed here in depth due to space limitations.

8 Acknowledgments

I would like to thank Kimmo Koskenniemi, Lauri Carlson and the three anonymous referees for valuable comments on earlier versions of this article. The work was funded by NorFA under the author's personal Ph.D. scholarship (ref.nr. 010529).

References

1. Nivre, J.: An efficient algorithm for projective dependency parsing. In: 8th Int'l Workshop on Parsing Technologies (IWPT 03), Nancy, France (2003)
2. Oflazer, K.: Dependency parsing with an extended finite-state approach. Computational Linguistics **29** (2003)
3. Mel'čuk, I.A.: Dependency Syntax: Theory and Practice. State University of New York Press, Albany (1988)
4. Sleator, D., Temperley, D.: Parsing English with a link grammar. In: 3rd International Workshop on Parsing Technologies. (1993) 277–291
5. Ihm, P., Lecerf, Y.: Éléments pour une grammaire générale des langues projectives. Technical Report EUR 210.f, Centre de Traitement de l'Information Scientifique — CETIS (1963)

6. Marcus, S.: Algebraic Linguistics; Analytical Models. Academic Press, New York and London (1967)
7. Höfler, S.: Link2tree: A dependency-constituency converter. Lizentiatsarbeit, Institute of Computational Linguistics, University of Zürich (2002)
8. Hays, D.G.: Dependency theory: A formalism and some observations. Language **40** (1964) 511–525
9. Gaifman, H.: Dependency systems and phrase-structure systems. Information and Control **8** (1965) 304–37
10. Robinson, J.J.: Dependency structures and transformational rules. Language **46** (1970) 259–285
11. Debusmann, R.: An introduction to dependency grammar. Hausarbeit für das Hauptseminar *Dependenzgrammatik* SoSe 99. Univeristät des Saarlandes (2000)
12. Lai, T.B.Y., Huang, C.: Functional constraints in dependency grammar. In: GLDV'99. Multilinguale Corpora: Codierung, Structurierung, Analyse. 11. Jahrestagung der Gesellschaft für Linguistische Daten Verarbeitung, 8.-10.7.1999, Frankfurt a/M (1999) 235–244
13. Harrison, M.A.: Introduction to Formal Language Theory. Reading, MA, Addison-Wesley (1978)
14. Chomsky, N., Schützenberger, M.P.: The algebraic theory of context-free languages. In Brafford, P., Hirschberg, D., eds.: Computer Programming and Formal Systems. North-Holland, Amsterdam (1963) 118–161
15. Roche, E., Schabes, Y., eds.: Finite-state language processing. A Bradford Book, MIT Press, Cambridge, MA (1997)
16. Lombardo, V., Lesmo, L.: An Earley-type recognizer for dependency grammar. In: 16th COLING. Volume 2., Copenhagen (1996) 723–728
17. Elworthy, D.: A finite state parser with dependency structure output. In: Proceedings of International Workshop on Parsing Technologies. (2000)
18. Maruyama, H.: Structural disambiguation with constraint propagation. In: Proceedings of the 28th ACL (ACL-90), Pittsburgh, PA (1990) 31–38
19. Duchier, D.: Lexicalized syntax and topology for non-projective dependency grammar. In: Joint Conference on Formal Grammars and Mathematics of Language FGMOL'01, Helsinki (2001)
20. Tesnière, L.: Éléments de Syntaxe Structurale. Editions Klincksieck, Paris (1959)
21. Yli-Jyrä, A., Koskenniemi, K.: Compiling contextual restrictions on strings into finite-state automata. In: The Eindhoven FASTAR Days, Technische Universiteit Eindhoven, Eindhoven, The Netherlands (2004)
22. Wrathall, C.: Characterizations of the Dyck sets. R.A.I.R.O. Informatique théorique/Theoretical Computer Science **11** (1977) 53–62
23. Yli-Jyrä, A.: Simplification of intermediate results during intersection of multiple weighted automata. In Droste, M., Vogler, H., eds.: ¨Weighted Automata — Theory and Applications¨, Dresden, Germany (2004) 46–48
24. Kromann, M.T., Mikkelsen, L., Lynge, S.K.: The Danish Dependency Treebank Website. Dept. of Computational Linguistics, Copenhagen Business School. `http://www.id.cbs.dk/~mtk/treebank` (2003)
25. Beesley, K.R., Karttunen, L.: Finite State Morphology. CSLI Studies in Computational Linguistics. CSLI Publications (2003)
26. Yli-Jyrä, A.: Axiomatization of restricted non-projective dependency trees through finite-state constraints that analyse crossing bracketings. In Kruijff, G.J.M., Duchier, D., eds.: Proceedings of the Workshop of Recent Advances in Dependency Grammar, COLING'04 Workshop. (2004) 33–40