Synchronous Rewriting in Treebanks

Laura Kallmeyer
University of Tübingen
Tübingen, Germany
lk@sfs.uni-tuebingen.de

Wolfgang Maier
University of Tübingen
Tübingen, Germany
wo.maier@uni-tuebingen.de

Giorgio Satta
University of Padua
Padova, Italy
satta@dei.unipd.it

Abstract

Several formalisms have been proposed for modeling trees with discontinuous phrases. Some of these formalisms allow for synchronous rewriting. However, it is unclear whether synchronous rewriting is a necessary feature. This is an important question, since synchronous rewriting greatly increases parsing complexity. We present a characterization of recursive synchronous rewriting in constituent treebanks with discontinuous annotation. An empirical investigation reveals that synchronous rewriting is actually a necessary feature. Furthermore, we transfer this property to grammars extracted from treebanks.

1 Introduction

Discontinuous phrases are frequent in natural language, particularly in languages with a relatively free word order. Several formalisms have been proposed in the literature for modeling trees containing such phrases. These include non-projective dependency grammar (Nivre, 2006), discontinuous phrase structure grammar (DPSG) (Bunt et al., 1987), as well as linear context-free rewriting systems (LCFRS) (Vijay-Shanker et al., 1987) and the equivalent formalism of simple range concatenation grammar (sRCG) (Boullier, 2000). Kuhlmann (2007) uses LCFRS for non-projective dependency trees. DPSG have been used in Plaehn (2004) for data-driven parsing of treebanks with discontinuous constituent annotation. Maier and Søgaard (2008) extract sRCGs from treebanks with discontinuous constituent structures.

Both LCFRS and sRCG can model discontinuities and allow for synchronous rewriting as well. We speak of synchronous rewriting when two or more context-free derivation processes are instantiated in a synchronous way. DPSG, which has also been proposed for modeling discontinuities, does not allow for synchronous rewriting because the different discontinuous parts of the yield of a non-terminal are treated locally, i.e., their derivations are independent from each other. So far, synchronous rewriting has not been empirically motivated by linguistic data from treebanks. In this paper, we fill this gap by investigating the existence of structures indicating synchronous rewriting in treebanks with discontinuous annotations. The question of whether we can find evidence for synchronous rewriting has consequences for the complexity of parsing. In fact, parsing with synchronous formalisms can be carried out in time polynomial in the length of the input string, with a polynomial degree depending on the maximum number of synchronous branches one can find in derivations (Seki et al., 1991).

In this paper, we characterize synchronous rewriting as a property of trees with crossing branches and in an empirical evaluation, we confirm that treebanks do contain recursive synchronous rewriting which can be linguistically motivated. Furthermore, we show how this characterization transfers to the simple RCGs describing these trees.

2 Synchronous Rewriting Trees in German treebanks

By **synchronous rewriting** we indicate the synchronous instantiation of two or more context-free derivation processes. As an example, consider the language $L = \{a^n b^n c^n d^n \mid n \geq 1\}$. Each of the two halves of some $w \in L$ can be obtained through a stand-alone context-free derivation, but for w to be in L the two derivations must be synchronized somehow. For certain tasks, synchronous rewriting is a desired property for a formalism. In machine translation, e.g., synchronous

rewriting is extensively used to model the synchronous dependence between the source and target languages (Chiang, 2007). The question we are concerned with in this paper is whether we can find instances of recursive synchronous rewriting in treebanks that show discontinuous phrases.

We make the assumption that, if the annotation of a treebank allows to express synchronous rewriting, then all cases of synchronous rewriting are present in the annotation. This means that, on the one hand, there are no cases of synchronous rewriting that the annotator "forgot" to encode. Therefore unrelated cases of parallel iterations in different parts of a tree are taken to be truly unrelated. On the other hand, if synchronous rewriting is annotated explicitly, then we take it to be a case of true synchronous rewriting, even if, based on the string, it would be possible to find an analysis that does not require synchronous rewriting. This assumption allows us to concentrate only on explicit cases of synchronous rewriting .

We concentrate on German treebanks annotated with trees with crossing branches. In such trees, synchronous rewriting amounts to cases where different components of a non-terminal category develop in parallel. In particular, we search for cases where the parallelism can be iterated. An example is the relative clause in (1), found in TIGER. Fig. 1 gives the annotation. As can be seen in the annotation, we have two VP nodes, each of which has a discontinuous span consisting of two parts. The two parts are separated by lexical material not belonging to the VPs. The two components of the second VP (*Pop-Idol* and *werden*) are included in the two components of the first, higher, VP (*genausogut auch Pop-Idol* and *werden können*). In other words, the two VP components are rewritten in parallel containing again two smaller VP components.

(1) ...der genausogut auch Pop-Idol hätte werden können
    ...who as well   also pop-star AUX become could
    "who could as well also become a pop-star"

Let us assume the following definitions: We map the elements of a string to their positions. We then say that the yield Υ of a node n in a tree is the set of all indices i such that n dominates the leaf labeled with the ith terminal. A yield Υ has a gap if there are $i_1 < i_2 < i_3$ such that $i_1, i_3 \in \Upsilon$ and $i_2 \notin \Upsilon$. For all $i, j \in \Upsilon$ with $i < j$, the set $\Upsilon_{\langle i,j \rangle} = \{k \mid i \leq k \leq j\}$ is a component of Υ if $\Upsilon_{\langle i,j \rangle} \subseteq \Upsilon$ and $i-1 \notin \Upsilon$ and $j+1 \notin \Upsilon$. We order

the components of Υ such that $\Upsilon_{\langle i_1,j_1 \rangle} < \Upsilon_{\langle i_2,j_2 \rangle}$ if $i_1 < i_2$.

Trees showing **recursive synchronous rewriting** can be characterized as follows: We have a non-terminal node n_1 with label A whose yield has a gap. n_1 dominates another node n_2 with label A such that for some $i \neq j$, the ith component of the yield of n_2 is contained in the ith component of the yield of n_1 and similar for the jth component. We call the path from n_1 to n_2 a **recursive synchronous rewriting segment (RSRS)**.

Table 1 shows the results obtained from searching for recursive synchronous rewriting in the German TIGER and NeGra treebanks. In a preprocessing step, punctuation has been removed, since it is directly attached to the root node and therefore not included in the annotation.

	TIGER	NeGra
number of trees	40,013	20,597
total num. of RSRS in all trees	1476	600
av. RSRS length in all trees	2.13	2.12
max. RSRS length in all trees	5	4

Table 1: Synchronous rewriting in treebanks

Example (1) shows that we find instances of recursive synchronous rewriting where each of the rewriting steps adds something to both of the parallel components. (1) was not an isolated case.

The annotation of (1) in Fig. 1 could be turned into a context-free structure if the lowest node dominating the material in the gap while not dominating the synchronous rewriting nodes (here VAFIN) is attached lower, namely below the lower VP node. (Note however that there is good linguistic motivation for attaching it high.) Besides such cases, we even encountered cases where the discontinuity cannot be removed this way. An example is (2) (resp. Fig. 2) where we have a gap containing an NP such that the lowest node dominating this NP while not dominating the synchronous rewriting nodes has a daughter to the right of the yields of the synchronous rewriting nodes, namely the extraposed relative clause. This structure is of the type $a^n c b^n d$, where a and b depend on each other in a left-to-right order and can be nested, and c and d also depend on each other and must be generated together. This is a structure that requires synchronous rewriting, even on the basis of the string language. Note that the nesting of VPs can be iterated, as can be seen in (3).

(2) ...ob   auf deren Gelände der Typ von
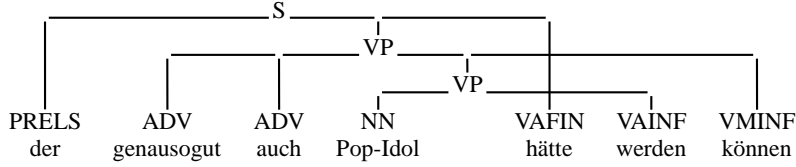   ...whether on their premises the type of

Figure 1: Example for recursive synchronous rewriting

Abstellanlage gebaut werden könne, der ...
parking facility built be could, which ...
"whether on their premises precisely the type of parking
facility could be built, which ..."

(3) ...ob auf deren Gelände der Typ von
...whether on their premises the type of
Abstellanlage eigentlich hätte schon gebaut werden
parking facility actually had already built be
sollen, der ...
should, which ...
"whether on their premises precisely the type of parking
facility should actually already have been built, which
..."

As a conclusion from these empirical results, we state that to account for the data we can find in treebanks with discontinuities, i.e., with crossing branches, we need a formalism that can express synchronous rewriting.

3 Synchronous Rewriting in Grammars Extracted from Treebanks

In the following, we will use simple RCG (which are equivalent to LCFRS) to model our treebank annotations. We extract simple RCG rewriting rules from NeGra and TIGER and check them for the possibility to generate recursive synchronous rewriting.

A **simple RCG** (Boullier, 2000) is a tuple $G = (N, T, V, P, S)$ where a) N is a finite set of predicate names with an arity function $dim\colon N \to \mathbb{N}$, b) T and V are disjoint finite sets of terminals and variables, c) P is a finite set of clauses of the form

$$A(\alpha_1, \ldots, \alpha_{dim(A)}) \to A_1(X_1^{(1)}, \ldots, X_{dim(A_1)}^{(1)})$$
$$\cdots A_m(X_1^{(m)}, \ldots, X_{dim(A_m)}^{(m)})$$

for $m \geq 0$ where $A, A_1, \ldots, A_m \in N$, $X_j^{(i)} \in V$ for $1 \leq i \leq m, 1 \leq j \leq dim(A_i)$ and $\alpha_i \in (T \cup V)^*$ for $1 \leq i \leq dim(A)$, and e) $S \in N$ is the start predicate name with $dim(S) = 1$. For all $c \in P$, it holds that every variable X occurring in c occurs exactly once in the left-hand side (LHS) and exactly once in the RHS. A simple RCG $G = (N, T, V, P, S)$ is a simple k-**RCG** if for all $A \in N, dim(A) \leq k$.

For the definition of the language of a simple RCG, we borrow the LCFRS definitions here: Let $G = \langle N, T, V, P, S \rangle$ be a simple RCG. For every $A \in N$, we define the yield of A, $yield(A)$ as follows:

a) For every $A(\vec{\alpha}) \to \varepsilon$, $\vec{\alpha} \in yield(A)$;

b) For every clause

$$A(\alpha_1, \ldots, \alpha_{dim(A)}) \to A_1(X_1^{(1)}, \ldots, X_{dim(A_1)}^{(1)})$$
$$\cdots A_m(X_1^{(m)}, \ldots, X_{dim(A_m)}^{(m)})$$

and all $\vec{\tau}_i \in yield(A_i)$ for $1 \leq i \leq m$, $\langle f(\alpha_1), \ldots, f(\alpha_{dim(A)}) \rangle \in yield(A)$ where f is defined as follows:

(i) $f(t) = t$ for all $t \in T$,

(ii) $f(X_j^{(i)}) = \vec{\tau}_i(j)$ for all $1 \leq i \leq m, 1 \leq j \leq dim(A_i)$ and

(iii) $f(xy) = f(x)f(y)$ for all $x, y \in (T \cup V)^+$.

c) Nothing else is in $yield(A)$.

The language is then $\{w \mid \langle w \rangle \in yield(S)\}$.

We are using the algorithm from Maier and Søgaard (2008) to extract simple RCGs from NeGra and TIGER. For the tree in Fig. 1, the algorithm produces for instance the following clauses:

```
PRELS(der) → ε
ADV(genausogut) → ε
...
S(X₁X₂X₃X₄) → PRELS(X₁)VP₂(X₁,X₄) VAFIN(X₃)
VP₂(X₁X₂X₃,X₄X₅) → ADV(X₁) ADV(X₂)
                              VP₂(X₃,X₄) VMINF(X₅)
VP₂(X₁,X₂) → NN(X₁) VAINF(X₂)
```

We distinguish different usages of the same category depending on their numbers of yield components. E.g., we distinguish non-terminals VP_1, VP_2, ... depending on the arity of the VP. We define $cat(A)$ for $A \in N$ as the category of A, independent from the arity, e.g., $cat(VP_2) = VP$.

In terms of simple RCG, synchronous rewriting means that in a single clause distinct variables occurring in two different arguments of the LHS predicate are passed to two different arguments of the same RHS predicate. We call this **recursive**
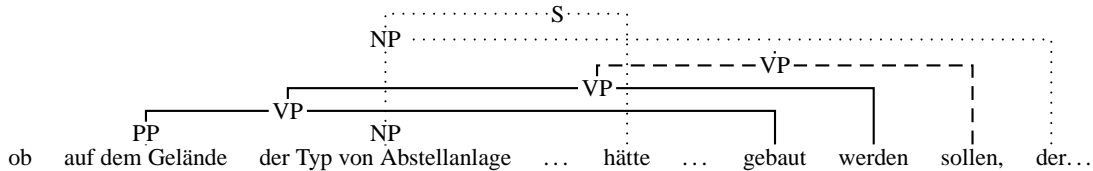
Figure 2: Iterable treebank example for synchronous rewriting

if, by a sequence of synchronous rewriting steps, we can reach the same two arguments of the same predicate again. Derivations using such cycles of synchronous rewriting lead exactly to the recursive synchronous rewriting trees characterized in section 2. In the following, we check to which extent the extracted simple RCG allows for such cycles.

In order to detect synchronous rewriting in a simple k-RCG G, we build a labeled directed graph $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}}, l)$ from the grammar with $V_{\mathcal{G}}$ a set of nodes, $E_{\mathcal{G}}$ a set of arcs and $l :$ $V_{\mathcal{G}} \to N' \times \{0, \ldots, k\} \times \{0, \ldots, k\}$ where $N' = \{cat(A) \,|\, A \in N\}$ a labeling function. $\mathcal{G}$ is constructed as follows. For each clause $A_0(\vec{\alpha}) \to A_1(\vec{\alpha_1}) \ldots A_m(\vec{\alpha_m}) \in P$ we consider all pairs of variables X_s, X_t for which the following conditions hold: (i) X_s and X_t occur in different arguments i and j of A_0, $1 \leq i < j \leq dim(A_0)$; *and* (ii) X_s and X_t occur in different arguments q and r of the same occurrence of predicate A_p in the RHS, $1 \leq q < r \leq dim(A_p)$ and $1 \leq p \leq m$. For each of these pairs, two nodes with labels $[cat(A_0), i, j]$ and $[cat(A_p), q, r]$, respectively, are added to $V_{\mathcal{G}}$ (if they do not yet exist, otherwise we take the already existing nodes) and a directed arc from the first node to the second node is added to $E_{\mathcal{G}}$. The intuition is that an arc in $\mathcal{G}$ represents one or more clauses from the grammar in which a gap between two variables in the LHS predicate is transferred to the same RHS predicate. To detect recursive synchronous rewriting, we then need to discover all elementary cycles in $\mathcal{G}$, i.e., all cycles in which no vertex appears twice. In order to accomplish this task efficiently, we exploit the algorithm presented in Johnson (1975). On a grammar extracted from NeGra (19,100 clauses), the algorithm yields a graph with 28 nodes containing 206,403 cycles of an average length of 12.86 and a maximal length of 28.

4 Conclusion

The starting point of this paper was the question whether synchronous rewriting is a necessary feature of grammer formalisms for modelling natu-

ral languages. In order to answer this question, we have characterized synchronous rewriting in terms of properties of treebank trees with crossing branches. Experiments have shown that recursive cases of synchronous rewriting occur in treebanks for German which leads to the conclusion that, in order to model these data, we need formalisms that allow for synchronous rewriting. In a second part, we have extracted a simple RCG from these treebanks and we have characterized the grammar properties that are necessary to obtain recursive synchronous rewriting. We then have investigated the extent to which a grammar extracted from Ne-Gra allows for recursive synchronous rewriting.

References

Pierre Boullier. 2000. Range concatenation grammars. In *Proceedings of IWPT*.

Harry Bunt, Jan Thesingh, and Ko van der Sloot. 1987. Discontinuous constituents in trees, rules and parsing. In *Proceedings of EACL*.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*.

Donald B. Johnson. 1975. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*.

Marco Kuhlmann. 2007. *Dependency Structures and Lexicalized Grammars*. Dissertation, Saarland University.

Wolfgang Maier and Anders Søgaard. 2008. Treebanks and mild context-sensitivity. In *Proceedings of Formal Grammar*.

Joakim Nivre. 2006. *Inductive Dependency Parsing*. Springer.

Oliver Plaehn. 2004. Computing the most probable parse for a discontinuous phrase-structure grammar. In *New developments in parsing technology*. Kluwer.

H. Seki, T. Matsumura, M. Fujii, and T. Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*.

K. Vijay-Shanker, David Weir, and Aravind Joshi. 1987. Characterising structural descriptions used by various formalisms. In *Proceedings of ACL*.