Hybrid Grammars for Discontinuous Parsing

Mark-Jan Nederhof
School of Computer Science
University of St Andrews
KY16 9SX, UK

Heiko Vogler
Department of Computer Science
Technische Universität Dresden
D-01062 Dresden, Germany

Abstract

We introduce the concept of hybrid grammars, which are extensions of synchronous grammars, obtained by coupling of lexical elements. One part of a hybrid grammar generates linear structures, another generates hierarchical structures, and together they generate discontinuous structures. This formalizes and generalizes some existing mechanisms for dealing with discontinuous phrase structures and non-projective dependency structures. Moreover, it allows us to separate the degree of discontinuity from the time complexity of parsing.

1 Introduction

Discontinuous phrases occur frequently in languages with relatively free word order, and adequate description of their structure requires special care (Kathol and Pollard, 1995; Müller, 2004). Even for languages such as English, with a relatively rigid word order, there is a clear need for discontinuous structures (McCawley, 1982; Stucky, 1987).

Early treebanks for English (Marcus et al., 1993) have often represented discontinuity in a way that makes it tempting to ignore it altogether, certainly for the purposes of parsing, whereas recent approaches tend to represent discontinuity in a more overt form, sometimes after transformation of existing treebanks (Choi and Palmer, 2010; Evang and Kallmeyer, 2011). In many modern treebanks, discontinuous structures have been given a prominent status (Böhmová et al., 2000).

Classes of trees without discontinuity can be specified as the sets of parse trees of context-free grammars (CFGs). Somewhat larger classes can be specified by tree substitution grammars (Sima'an et al., 1994) and regular tree grammars (Brainerd, 1969; Gécseg and Steinby, 1997). Practical parsers for these three formalisms have running time $\mathcal{O}(n^3)$, where n is the length of the input sentence. Discontinuous structures go beyond their strong generative capacity however. Similarly, non-projective dependency structures cannot be obtained by traditional dependency grammars. See (Rambow, 2010) for discussion of the relation between constituent and dependency structures and see (Maier and Lichte, 2009) for a comparison of discontinuity and non-projectivity.

One way to solve the above problems has been referred to as *pseudo-projectivity*, i.e. a parser produces a projective structure, which in a second phase is transformed into a non-projective structure (Kahane et al., 1998; McDonald and Pereira, 2006; Nivre and Nilsson, 2005). In particular, this may involve *lifting*, whereby one end point of a dependency link moves across a path of nodes. A related idea for discontinuous phrase structure is the reversible splitting conversion of (Boyd, 2007). See also (Johnson, 2002; Campbell, 2004; Gabbard et al., 2006).

As shown by (Nivre, 2009), the second phase of pseudo-projective dependency parsing can be interleaved with the first, by replacing the usual one-way input tape by an additional stack, or *buffer*. Where

Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, pages 1370–1381, Dublin, Ireland, August 23-29 2014.

non-topmost positions from the parsing stack are moved back to the buffer, input positions are effectively swapped and non-projective dependency structures arise.

Tree adjoining grammars (TAGs) can describe strictly larger classes of word order phenomena than CFGs (Rambow and Joshi, 1997). TAG parsers have a time complexity of $\mathcal{O}(n^6)$ (Vijay-Shankar and Joshi, 1985). However, the *derived* trees they generate are still continuous. Although their *derivation* trees may be argued to be discontinuous, these by themselves are not normally the desired syntactic structures. It was argued by (Becker et al., 1991) that further additions to TAGs are needed to obtain adequate descriptions of scrambling phenomena.

An alternative is proposed by (Kallmeyer and Kuhlmann, 2012): a transformation is added that turns a derivation tree of a (lexicalized) TAG into a non-projective dependency structure. A very similar mechanism is used to obtain non-projective dependency structures using linear context-free rewriting systems (LCFRSs) (Kuhlmann, 2013) that are lexicalized. In a LCFRS the synthesis of strings is normally specified by yield functions associated with rules. By an additional interpretation of the templates of these yield functions in the algebra of dependency trees (with the overt lexical items as roots), the LCFRS generates both strings and (possibly non-projective) dependency structures.

However, the running time of LCFRS parsers is generally very high, still polynomial in the sentence length, but with a degree determined by properties of the grammar; difficulties involved in running LCFRS parsers for natural languages are described by (Kallmeyer and Maier, 2013).

It follows from the above that there is considerable freedom in the design of parsers that produce discontinuous structures for given input sentences. One can distinguish between two main issues. The first is the formalism that guides the parsing of the input. This determines a class of input (string) languages, which can be that of the context-free languages, or tree adjoining languages, etc. We assume parsing with any of these formalisms results in derivations of some sort. The second main issue is the mechanism that translates such derivations into discontinuous structures.

This leads to a number of open questions that are all related. First, what is, or should be, the division of labor between the parser producing the derivations and the mechanism turning those derivations into discontinuous structures? If we want to achieve high degrees of discontinuity in the output structures, should the formalism for the input language be much more powerful than, say, context-free? Or can highly discontinuous structures be obtained equally well through ordinary CFGs in combination with an advanced mechanism producing discontinuous structures out of derivations?

Second, how should one approach the problem of finding the grammar (and grammar class) for the input language and the mapping from derivations to structures if the only thing that is given is a treebank? A third question is which formalisms are suitable to formally describe mappings from derivations to discontinuous structures. Lastly, can we characterize the classes of output (tree-)languages for various combinations of input grammars and derivation-to-structure mappings?

In this paper we provide one possible answer to these questions by a new type of formalism, which we call *hybrid grammars*. Such a grammar consists of a string grammar and a tree grammar. Derivations are coupled so as to achieve synchronous rewriting. The input string language and the output tree language are thereby straightforwardly defined. Different from synchronous grammars (Shieber and Schabes, 1990; Satta and Peserico, 2005) is that occurrences of terminal symbols are also coupled. Thereby the linear order of the symbols in a derived string imposes an order on the coupled symbols in the synchronously derived tree; this allows a straightforward specification of a discontinuous structure.

One can define a hybrid grammar consisting of a simple macro grammar (Fischer, 1968) and a simple context-free tree grammar (Rounds, 1970), but various other combinations of a string grammar and a tree grammar are possible as well. Due to lack of space we will here concentrate on only one kind of hybrid grammar, namely that consisting of a LCFRS as string grammar and a form of definite clause program as tree grammar. We will show that hybrid grammars that induce (finite) sets of hybrid trees can always be constructed, even if the allowable derivations are severely restricted, and we discuss experiments. Lastly, a negative result will be given, which shows that a certain linguistic phenomenon cannot be handled if the string grammar is too restricted.

We cast our definitions in terms of *hybrid trees*, of which discontinuous phrase structures and non-

projective dependency structures are special cases.[1] Thereby the generality of the framework is demonstrated.

2 Preliminaries

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ and $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. For each $n \in \mathbb{N}_+$, we let $[n]$ stand for the set $\{1, \ldots, n\}$, and we let $[0]$ stand for $\emptyset$. We write $[n]_0$ to denote $[n] \cup \{0\}$. We fix an infinite list $x_1, x_2, \ldots$ of pairwise distinct *variables*. We let $X = \{x_1, x_2, x_3, \ldots\}$ and $X_k = \{x_1, \ldots, x_k\}$ for each $k \in \mathbb{N}$.

A *ranked set* Δ is a set of symbols, associated with a rank function assigning a number $\mathrm{rk}_\Delta(\delta) \in \mathbb{N}$ to each symbol $\delta \in \Delta$. A *ranked alphabet* is a ranked set with a finite number of symbols. We let $\Delta^{(k)}$ denote $\{\delta \in \Delta \mid \mathrm{rk}_\Delta(\delta) = k\}$.

The following definitions were inspired by (Seki and Kato, 2008). The sets of *terms* and *sequence-terms* (*s-terms*) over ranked set Δ, with variables in some set $Y \subseteq X$, are denoted by $T_\Delta(Y)$ and $T^*_\Delta(Y)$, respectively, and defined inductively as follows:
(i) $Y \subseteq T_\Delta(Y)$,
(ii) if $k \in \mathbb{N}$, $\delta \in \Delta^{(k)}$ and $s_i \in T^*_\Delta(Y)$ for each $i \in [k]$, then $\delta(s_1, \ldots, s_k) \in T_\Delta(Y)$, and
(iii) if $n \in \mathbb{N}$ and $t_i \in T_\Delta(Y)$ for each $i \in [n]$, then $\langle t_1, \ldots, t_n \rangle \in T^*_\Delta(Y)$.
We let T^*_Δ and T_Δ stand for $T^*_\Delta(\emptyset)$ and $T_\Delta(\emptyset)$ respectively. Throughout this paper, we use variables such as s and s_i for s-terms and variables such as t and t_i for terms. The justification for using s-terms as defined here is that they provide the required flexibility for dealing with both strings ($\Delta = \Delta^{(0)}$) and unranked trees ($\Delta = \Delta^{(1)}$), in combination with derivational nonterminals.

Concatenation of s-terms is given by $\langle t_1, \ldots, t_n \rangle \cdot \langle t_{n+1}, \ldots, t_{n+m} \rangle = \langle t_1, \ldots, t_{n+m} \rangle$. Sequences such as $s_1, \ldots, s_k$ or $x_1, \ldots, x_k$ will typically be abbreviated to $s_{1,k}$ or $x_{1,k}$, respectively. For $\delta \in \Delta^{(0)}$ we sometimes abbreviate $\delta()$ to δ.

In examples we also abbreviate $\langle t_1, \ldots, t_n \rangle$ to $t_1 \cdots t_n$, that is, omitting the angle brackets and commas. Moreover, we sometimes abbreviate $\delta(\langle\rangle)$ to δ. Whether δ then stands for $\delta(\langle\rangle)$ or for $\delta()$ depends on whether $\delta \in \Delta^{(1)}$ or $\delta \in \Delta^{(0)}$, which will be clear from the context.

Subterms in terms or s-terms are identified by *positions*; these can be formalized by a suitable refinement of the familiar notion of Gorn address. The set of all positions in term t or in s-term s is denoted by $\mathrm{pos}(t)$ or $\mathrm{pos}(s)$, respectively. The subset of $\mathrm{pos}(t)$ consisting of all positions where the label is in some set $\Gamma \subseteq \Delta$ is denoted by $\mathrm{pos}_\Gamma(t)$.

3 Hybrid trees

The purpose of this section is to unify existing notions of non-projective dependency structures and discontinuous phrase structures, formalized using s-terms.

We fix an alphabet $\Delta = \Delta^{(1)}$ and a subset $\Gamma \subseteq \Delta$. A *hybrid tree* over (Γ, Δ) is a pair $h = (s, \leq_s)$, where $s \in T^*_\Delta$ and $\leq_s$ is a total order on $\mathrm{pos}_\Gamma(s)$. In words, a hybrid tree combines hierarchical structure, in the form of an s-term over the full alphabet Δ, with a linear structure, which can be seen as a string over $\Gamma \subseteq \Delta$. This string will be denoted by $\mathrm{str}(h)$.

For discontinuous phrase structures, the elements of Γ would typically represent lexical items, and the elements of $\Delta \setminus \Gamma$ would typically represent syntactic categories. For non-projective dependency structures, Δ would be equal to Γ. Simple examples of discontinuous phrase structures are presented in Figures 1 and 2.

4 Basic grammatical formalisms

The concept of hybrid grammars is illustrated in Section 5, by coupling a class of string grammars and a class of tree grammars.

[1] Moreover, we need to avoid any confusion with the term "discontinuous tree" from (Bunt, 1996), which is characterized by the notion of "context daughter", which is absent from our framework. The term "hybrid tree" was used before by (Lu et al., 2008), also for a mixture of a tree structure and a linear structure, generated by a probabilistic model. However, the linear 'surface' structure was obtained by a simple left-to-right tree traversal, whereas a meaning representation was obtained by a slightly more flexible traversal of the same tree. The emphasis in the current paper is rather on separating the linear structure from the tree structure.

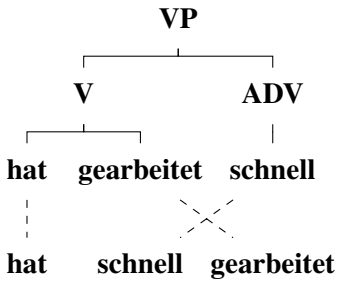

Figure 1: Hybrid tree for German "[...] hat schnell gearbeitet" ("[...] has worked quickly"), after (Seifert and Fischer, 2004). The bottom line indicates the word order in German. (Alternative analyses exist that do not require discontinuity; we make no claim the structure above is the most adequate.)

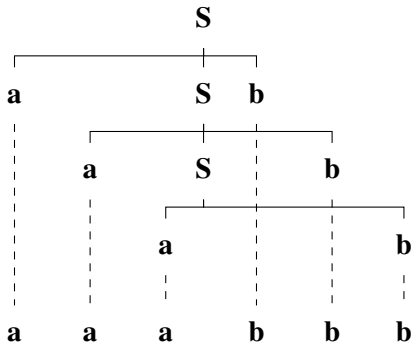

Figure 2: Abstract representation of cross-serial dependencies in Dutch (Bresnan et al., 1982).

4.1 Linear context-free rewriting systems

Much as in (Vijay-Shanker et al., 1987), we define a *linear context-free rewriting system* (LCFRS) as a tuple $G = (N, S, \Sigma, P)$, where N is a ranked alphabet of *nonterminals*, $S \in N^{(1)}$ is the *start symbol*, $\Sigma = \Sigma^{(0)}$ is a ranked alphabet of *terminals* ($\Sigma \cap N = \emptyset$), and P is a finite set of *rules*, each of the form:

$$A_0(s_{1,k_0}) \rightarrow \langle A_1(x_{1,m_1}), A_2(x_{m_1+1,m_2}), \ldots, A_n(x_{m_{n-1}+1,m_n})\rangle \tag{1}$$

where $n \in \mathbb{N}$, $A_i \in N^{(k_i)}$ for each $i \in [n]_0$, and $m_i = \sum_{j:1\leq j\leq i} k_j$ for each $i \in [n]$, and $s_j \in T^*_\Sigma(X_{m_n})$ for each $j \in [k_0]$. In words, the right-hand side is an s-term consisting of nonterminals A_i ($i \in [n]$), with distinct variables as arguments; there are m_n variables altogether, which is the sum of the ranks k_i of all A_i ($i \in [n]$). The left-hand side is an occurrence of A_0 with each argument being a string of variables and terminals. Furthermore, we demand that each x_j ($j \in [m_n]$) occurs exactly once in the left-hand side. The largest rank of any nonterminal is called the *fanout* of the grammar.

A *rule instance* is obtained by choosing a rule of the above form, and consistently substituting variables with s-terms in T^*_Σ (which are strings due to the terminals having rank 0). The language induced is the set of s-terms s such that $\langle S(s)\rangle \Rightarrow^*_G \langle\rangle$, where $\Rightarrow_G$ is the 'derives' relation that uses rule instances. For given s, the set of all LCFRS derivations $\langle S(s)\rangle \Rightarrow^*_G \langle\rangle$ (in compact tabular form) can be obtained in polynomial time in the length of s (Seki et al., 1991).

Example 1

An example of a LCFRS is presented on the right. Terminals are lower case bold letters and nonterminals are upper-case italic letters. All derived strings are of the form $\mathbf{a}^m\mathbf{c}^n\mathbf{b}^m\mathbf{d}^n$ with $m, n \in \mathbb{N}$. The linguistic relevance lies in cross-serial dependencies in Swiss German (Shieber, 1985).

$$\begin{aligned}
S(x_1x_3x_2x_4) &\rightarrow A(x_1, x_2)\ B(x_3, x_4)\\
A(\mathbf{a}x_1, \mathbf{b}x_2) &\rightarrow A(x_1, x_2)\\
A(\langle\rangle, \langle\rangle) &\rightarrow \langle\rangle\\
B(\mathbf{c}x_1, \mathbf{d}x_2) &\rightarrow B(x_1, x_2)\\
B(\langle\rangle, \langle\rangle) &\rightarrow \langle\rangle
\end{aligned}$$

4.2 Definite clause programs

In this section we describe a particular kind of definite clause programs. Our definition is inspired by (Deransart and Małuszynski, 1985), which investigated the relation between logic programs and attribute grammars, together with the "syntactic single use requirement" from (Giegerich, 1988). The values produced are trees (or to be more precise s-terms).

A *simple definite clause program* (sDCP) is a tuple $G = (N, S, \Sigma, P)$, where N is a ranked alphabet of *nonterminals* and $\Sigma = \Sigma^{(1)}$ is a ranked alphabet of *terminals*.[2] Moreover, each nonterminal $A \in N$ has a fixed i-rank (the number of *inherited* arguments) and a fixed s-rank (the number of *synthesized* arguments), denoted by i-rk(A) and s-rk(A), respectively, satisfying i-rk(A) + s-rk(A) = $\mathrm{rk}_N(A)$. In our notation, the inherited arguments precede the synthesized arguments. The *start symbol* S has only one argument, which is synthesized, i.e. $\mathrm{rk}_N(S) =$ s-rk$(S) = 1$ and i-rk$(S) = 0$.

A rule is of the form:

$$A_0(x^{(0)}_{1,k_0}, s^{(0)}_{1,k'_0}) \to \langle A_1(s^{(1)}_{1,k_1}, x^{(1)}_{1,k'_1}), \ldots, A_n(s^{(n)}_{1,k_n}, x^{(n)}_{1,k'_n})\rangle \tag{2}$$

where $n \in \mathbb{N}$, $k_i =$ i-rk(A_i) and $k'_i =$ s-rk(A_i), for $i \in [n]_0$. The set of variables occurring in the lists $x^{(0)}_{1,k_0}$ and $x^{(i)}_{1,k'_i}$ ($i \in [n]$) equals X_m, where $m = k_0 + \sum_{i\in[n]} k'_i$. In other words, every variable from X_m occurs exactly once in all these lists together. This is where values 'enter' the rule. Further, the s-terms in $s^{(0)}_{1,k'_0}$ and $s^{(i)}_{1,k_i}$ ($i \in [n]$) are in $T^*_\Sigma(X_m)$ and together contain each variable in X_m exactly once. This is where values are combined and 'exit' the rule.

The 'derives' relation $\Rightarrow_G$ and other relevant notation are defined as for LCFRSs (where the s-terms in arguments are now trees due to the terminals having rank 1). If the rules in a derivation are given, then the relevant rule instances are uniquely determined, and can be computed in linear time in the size of the derivation, provided the sDCP contains no cycles. The existence of cycles is decidable, as we know from the literature on attribute grammars. There are sufficient conditions for absence of cycles, such as the grammar being L-attributed (Bochmann, 1976). In this article, we will assume that sDCPs contain no cycles.

Example 2

An example of a sDCP is presented on the right, where the first argument of B is inherited and all other arguments are synthesized. A derived s-term is e.g. **c B**(**c B**(**a A**($\langle\rangle$) **b**) **d**) **d**.

$$\begin{aligned}
S(x_2) &\to A(x_1)\ B(x_1, x_2)\\
A(\mathbf{a}\ \mathbf{A}(x_1)\ \mathbf{b}) &\to A(x_1)\\
A(\langle\rangle) &\to \langle\rangle\\
B(x_1, \mathbf{c}\ \mathbf{B}(x_2)\ \mathbf{d}) &\to B(x_1, x_2)\\
B(x_1, x_1) &\to \langle\rangle
\end{aligned}$$

5 Hybrid grammars

We couple derivations in two grammars in a way similar to how this is commonly done for synchronous grammars, namely by *indexed* symbols. However, we apply the mechanism not only to derivational nonterminals but also to terminals.

Let Γ be a ranked alphabet. We define the ranked set $\mathcal{I}(\Gamma) = \{\gamma^{[u]} \mid \gamma \in \Gamma, u \in \mathbb{N}_+\}$, with $\mathrm{rk}_{\mathcal{I}(\Gamma)}(\gamma^{[u]}) = \mathrm{rk}_\Gamma(\gamma)$. Let Δ be another ranked alphabet ($\Delta \cap \Gamma = \emptyset$) and $Y \subseteq X$, with X as in Section 2. We let $\mathcal{I}^*_{\Gamma,\Delta}(Y)$ be the set of all s-terms $s \in T^*_{\mathcal{I}(\Gamma)\cup\Delta}(Y)$ in which each index occurs at most once.

For an s-term s, let $ind(s)$ be the set of all indices occurring in s. The deindexing function $\mathcal{D}$ removes all indices from an s-term $s \in \mathcal{I}^*_{\Gamma,\Delta}(Y)$ to obtain $\mathcal{D}(s) \in T^*_{\Gamma\cup\Delta}(Y)$. The set $\mathcal{I}_{\Gamma,\Delta}(Y) \subseteq T_{\mathcal{I}(\Gamma)\cup\Delta}(Y)$ of terms with indexed symbols is defined much as above. We let $\mathcal{I}^*_{\Gamma,\Delta} = \mathcal{I}^*_{\Gamma,\Delta}(\emptyset)$ and $\mathcal{I}_{\Gamma,\Delta} = \mathcal{I}_{\Gamma,\Delta}(\emptyset)$.

A *LCFRS/sDCP hybrid grammar* (HG) is a tuple $G = ((N_1, S_1, \Gamma), (N_2, S_2, \Sigma), P)$, subject to the following restrictions. The objects Γ and Σ are ranked alphabets with $\Gamma = \Gamma^{(0)}$ and $\Sigma = \Sigma^{(1)}$. As mere sets of symbols, we demand $\Gamma \subseteq \Sigma$ but the rank functions associated with Γ and Σ differ. Let Δ be the ranked alphabet $\Sigma \setminus \Gamma$, with $\mathrm{rk}_\Delta(\delta) = 1$ for $\delta \in \Delta$.

The *hybrid rules* in P are of the form $[\rho_1, \rho_2]$ where ρ_1 has the form in Equation (1) of an LCFRS rule except that $s_i \in \mathcal{I}^*_{\Gamma,\emptyset}(X_{m_n})$ ($i \in [k_0]$) and $A_i \in \mathcal{I}(N_1)$ ($i \in [n]$) and each index in ρ_1 occurs exactly once, and ρ_2 has the form in Equation (2) of a sDCP rule except that the s-terms in $s^{(0)}_{1,k'_0}$ and $s^{(i)}_{1,k_i}$ ($i \in [n]$) are in $\mathcal{I}^*_{\Gamma,\Delta}(X_m)$ and $A_i \in \mathcal{I}(N_2)$ ($i \in [n]$) and each index in ρ_2 occurs exactly once. We require that $ind(\rho_1) = ind(\rho_2)$ and each index either couples a pair of identical terminals or couples a pair of (possibly distinct) nonterminals.

[2]The term 'simple' here has a more restrictive meaning than the term with the same name in (Deransart and Małuszynski, 1985).

Let P_1 and P_2 be the sets of all $\mathcal{D}(\rho_1)$ and $\mathcal{D}(\rho_2)$, respectively, of some hybrid rule $[\rho_1, \rho_2]$. Then we refer to the LCFRS (N_1, S_1, Γ, P_1) and the sDCP (N_2, S_2, Σ, P_2) as the first and second *components*, respectively, of G.

In order to define the 'derives' relation $\Rightarrow_G$, we need rule instantiation as before, in combination with *reindexing*, which is a common notion for synchronous grammars. This allows specification of a set of pairs $[s_1, s_2] \in \mathcal{I}^*_{\Gamma,\emptyset} \times \mathcal{I}^*_{\Gamma,\Delta}$ which are such that $[\langle S_1^{\boxed{1}}(s_1)\rangle, \langle S_2^{\boxed{1}}(s_2)\rangle] \Rightarrow^*_G [\langle\rangle, \langle\rangle]$. For each such pair we can construct a hybrid tree $(s, \leq_s)$ over (Γ, Σ), where $s = \mathcal{D}(s_2)$, and $\leq_s$ is defined as follows. If there is a combination of positions p_1, p'_1, p_2, p'_2 such that at p_1 in s_1 we find the same label as at p_2 in s_2 (this label must then be in $\mathcal{I}(\Gamma)$), and at p'_1 in s_1 we find the same label as at p'_2 in s_2, and p_1 occurs to the left of p'_1, then $p_2 \leq_s p'_2$. The language induced by G is defined as the set of all such hybrid trees.

Given an input string, the desired hybrid trees can be effectively enumerated. To be exact, after construction of the parse table by a LCFRS parser, which takes polynomial time in the length of the string, synchronous derivations can be enumerated. Extracting a single derivation from the table requires linear time in the size of that derivation. Given a derivation, an s-term can be constructed in linear time in the size of that derivation, applying sDCP rules in the second component. This s-term, in combination with the input string and the indices linking the two is then easily extended to a hybrid tree as outlined above.

Example 3

The hybrid tree in Figure 1 is obtained by the HG on the right. (All arguments in the second component are synthesized.) We derive:

$$[\mathit{VP}(x_1x_2x_3) \rightarrow V^{\boxed{1}}(x_1, x_3)\ \mathit{ADV}^{\boxed{2}}(x_2), \mathit{VP}(\mathbf{VP}(x_1x_2)) \rightarrow V^{\boxed{1}}(x_1)\ \mathit{ADV}^{\boxed{2}}(x_2)]$$
$$[V(\mathbf{h}^{\boxed{1}}, \mathbf{g}^{\boxed{2}}) \rightarrow \langle\rangle, V(\mathbf{V}(\mathbf{h}^{\boxed{1}}\mathbf{g}^{\boxed{2}})) \rightarrow \langle\rangle]$$
$$[\mathit{ADV}(\mathbf{s}^{\boxed{1}}) \rightarrow \langle\rangle, \mathit{ADV}(\mathbf{ADV}(\mathbf{s}^{\boxed{1}})) \rightarrow \langle\rangle]$$

$$[\mathit{VP}^{\boxed{1}}(\mathbf{h}^{\boxed{2}}\mathbf{s}^{\boxed{3}}\mathbf{g}^{\boxed{4}}), \mathit{VP}^{\boxed{1}}(\mathbf{VP}(\mathbf{V}(\mathbf{h}^{\boxed{2}}\mathbf{g}^{\boxed{4}})\ \mathbf{ADV}(\mathbf{s}^{\boxed{3}})))] \Rightarrow$$
$$[V^{\boxed{1}}(\mathbf{h}^{\boxed{2}}, \mathbf{g}^{\boxed{4}})\ \mathit{ADV}^{\boxed{5}}(\mathbf{s}^{\boxed{3}}), V^{\boxed{1}}(\mathbf{V}(\mathbf{h}^{\boxed{2}}\mathbf{g}^{\boxed{4}}))\ \mathit{ADV}^{\boxed{5}}(\mathbf{ADV}(\mathbf{s}^{\boxed{3}}))] \Rightarrow$$
$$[\mathit{ADV}^{\boxed{5}}(\mathbf{s}^{\boxed{3}}),\ \mathit{ADV}^{\boxed{5}}(\mathbf{ADV}(\mathbf{s}^{\boxed{3}}))] \Rightarrow [\langle\rangle, \langle\rangle]$$

Note that in the LCFRS that is the first component of the HG above, nonterminal V has rank 2. On the right is an alternative HG deriving the same hybrid tree, but now with all LCFRS nonterminals having rank 1, by which we obtain a syntactic variant of a CFG. Yet another HG for the same hybrid tree will be discussed in the next section, where we will see that the first and second components can be disconnected even further, departing from the traditional way of LCFRS parsing.

$$[\mathit{VP}(x_1) \rightarrow V^{\boxed{1}}(x_1), \mathit{VP}(\mathbf{VP}(x_1)) \rightarrow V^{\boxed{1}}(x_1)]$$
$$[V(\mathbf{h}^{\boxed{1}}x_1\mathbf{g}^{\boxed{2}}) \rightarrow \mathit{ADV}^{\boxed{3}}(x_1), V(\mathbf{V}(\mathbf{h}^{\boxed{1}}\mathbf{g}^{\boxed{2}})\ x_1) \rightarrow \mathit{ADV}^{\boxed{3}}(x_1)]$$
$$[\mathit{ADV}(\mathbf{s}^{\boxed{1}}) \rightarrow \langle\rangle, \mathit{ADV}(\mathbf{ADV}(\mathbf{s}^{\boxed{1}})) \rightarrow \langle\rangle]$$

Example 4

Hybrid trees as in Figure 2 can be obtained by the HG on the right.

$$[A(x_1x_2) \rightarrow S^{\boxed{1}}(x_1, x_2), A(x_1) \rightarrow S^{\boxed{1}}(x_1)]$$
$$[S(\mathbf{a}^{\boxed{1}}x_1, \mathbf{b}^{\boxed{2}}x_2) \rightarrow S^{\boxed{3}}(x_1, x_2), S(\mathbf{S}(\mathbf{a}^{\boxed{1}}x_1\mathbf{b}^{\boxed{2}}) \rightarrow S^{\boxed{3}}(x_1)]$$
$$[S(\langle\rangle, \langle\rangle) \rightarrow \langle\rangle, S(\langle\rangle) \rightarrow \langle\rangle]$$

6 Grammar induction

We define a *recursive partitioning* of a string $s = \alpha_1 \cdots \alpha_n$ as a tree whose nodes are labeled with subsets of $[n]$. The root is labeled with $[n]$. Each leaf is labeled with a single element of $[n]$. Each internal node is labeled with the union of the labels of its children, which furthermore must be disjoint. We say a subset of $[n]$ has *fanout* k if k is the smallest number such that it can be written as the union of k sets of consecutive numbers.

A derivation of an LCFRS relates straightforwardly to a recursive partitioning. Consider for example the derivation of string **h s g** by the LCFRS that is the first component of the first HG in Example 3. The root would be labeled $\{1, 2, 3\}$, with children labeled $\{1, 3\}$ and $\{2\}$. The node labeled $\{1, 3\}$ has children labeled $\{1\}$ and $\{3\}$. The fanout of $\{1, 3\}$ is 2, whereas it is 1 for all other node labels. One may also extract a recursive partitioning directly from a hybrid tree, by associating each node with the set of positions of terminals that it dominates. For example, Figure 1 gives rise to the same recursive partitioning as the one mentioned above.

One central observation of this paper is that for any hybrid tree $h = (s, \leq_s)$ and any recursive partitioning of $\mathrm{str}(h)$, not necessarily extracted from h, we can construct a hybrid grammar G allowing a derivation of h, and moreover, the first (LCFRS) component of that derivation parses $\mathrm{str}(h)$ according to the given recursive partitioning. This observation holds for both dependency structures and constituent structures. The proof for dependency structures is quite technical however, and requires that the second (sDCP) component of a hybrid grammar has rules with inherited arguments. For lack of space, we can only give an outline for constituent structures, or in other words, we consider only input hybrid trees over (Γ, Δ) where labels from Γ occur exclusively at the leaves. In the resulting hybrid grammars, all sDCP rules will have only synthesized arguments.

The intuition is the following. For each node of the given recursive partitioning, the numbers in its label correspond to leaves of s, for the given hybrid tree $h = (s, \leq_s)$. There is a smallest number of maximal disjoint subtrees in s that together contain all those leaves and no others. If we now relate a parent node of the recursive partitioning to its child nodes, then we see that the relevant disjoint subtrees in s for the children can be combined to give the relevant disjoint subtrees for the parent, possibly adding further internal nodes. This process can be expressed in terms of a hybrid rule. Each pair consisting of a hybrid tree and a recursive partitioning gives rise to a number of hybrid rules. For a collection of such pairs, we can combine all the rules into a hybrid grammar.

Example 5 Consider again the hybrid tree in Figure 1, in combination with a recursive partitioning whose root has children labeled $\{1, 2\}$ and $\{3\}$. The relevant disjoint subtrees for $\{1, 2\}$ are **hat** and **ADV**(**schnell**) and for $\{3\}$ there is the subtree **gearbeitet**. (In a real-world grammar we would have parts of speech occurring above all the words.) An appropriate hybrid rule that both respects the recursive partitioning (by the first component LCFRS rule) and puts together relevant parts of the hybrid tree (by the second component sDCP rule) would be of the form:

$$[A(x_1x_2) \rightarrow B^{\boxed{1}}(x_1)\ C^{\boxed{2}}(x_2), A(\mathbf{VP}(\mathbf{V}(x_1x_3)x_2)) \rightarrow B^{\boxed{1}}(x_1, x_2)\ C^{\boxed{2}}(x_3)]$$

Here A, B and C should to be chosen to be consistent with neighboring nodes in the recursive partitioning, to be discussed next. An alternative recursive partitioning whose root has children labeled $\{1, 3\}$ and $\{2\}$ leads to the first hybrid rule in Example 3 (apart from nonterminal names).

We have experimented with two ways of naming nonterminals in the derived hybrid rules. The first encodes the list of labels of the roots of the relevant disjoint subtrees. In the above example, we would have a name such as $\langle \mathbf{hat}, \mathbf{ADV} \rangle$ for A. For fanout greater than 1, the locations of the 'gaps' are explicitly indicated. For example, we might have $\langle \mathbf{hat}, \mathit{gap}, \mathbf{gearbeitet} \rangle$. We will call this *strict* labeling. The second, and less precise, way is to replace lists of labels of siblings by a single name of the form children-of(X), where X is the label of the parent. We will call this *child* labeling.

Because our construction of hybrid grammars works for all recursive partitionings, there is no need to limit ourselves to those extracted directly from the hybrid trees. Moreover, a given recursive partitioning can be transformed into a similar but different one in which fanout is restricted to some given value $k \geq 1$. One possible procedure is to start at the root. If the label J of the present node is a singleton, then we stop. Otherwise, we search breadth-first through the subtree of the present node to identify a descendant such that both its label J' and $J \setminus J'$ have fanout not exceeding k. (It is easy to see such a node always exists: ultimately breadth-first search will reach the leaves, which are labeled with singletons.) The present node is now given two children, the first is the node labeled J' that we identified above, and the second is a copy of the present subtree, but with J' subtracted from the label of every node. (Nodes

labeled with the empty set are removed, and if a node has the same label as its parent then the two are collapsed.) We repeat the procedure for both children recursively. Note that with $k = 1$, we can induce a 'CFG/sDCP' hybrid grammar, that is, with the first component having fanout 1.

Example 6

The recursive partitioning in the left half of Figure 3 has a node labeled $\{1, 3, 6, 7\}$, with fanout 3. With $J = \{1, 2, 3, 5, 6, 7\}$ and $k = 2$, one possible choice for J' is $\{3, 7\}$, as then both J' and $J \setminus J' = \{1, 2, 5, 6\}$ have fanout not exceeding 2. This leads to the partitioning in the right half of the figure. Because now all node labels have fanout not exceeding 2, recursive traversal will make no further changes. Other valid choices for J' would be $\{2\}$ and $\{5\}$. Not a valid choice for J' would be $\{1, 6\}$, as $J \setminus \{1, 6\} = \{2, 3, 5, 7\}$, which has fanout 3.

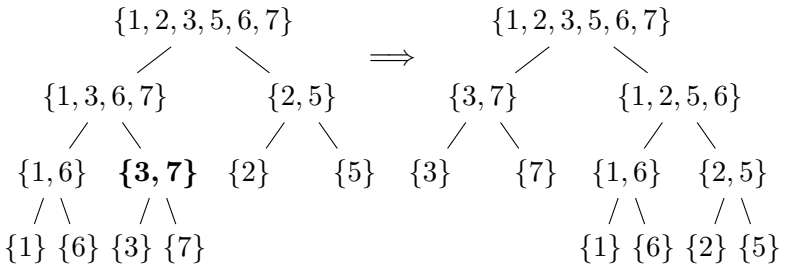

Figure 3: Transformation of recursive partitioning to restrict fanout to 2.

Our procedure ensures that subsequent grammar induction leads to binary grammars. Note that this contrasts with binarization algorithms (Gómez-Rodríguez and Satta, 2009; Gómez-Rodríguez et al., 2009) that are applied *after* a grammar is obtained. Unlike (van Cranenburgh, 2012), our objective is not to obtain a 'coarse' grammar for the purpose of coarse-to-fine parsing.

In experiments we also considered the *right-branching* partitioning, whose internal node labels are $\{m, m + 1, \ldots, n\}$, with children labeled $\{m\}$ and $\{m + 1, \ldots, n\}$. Similarly, there is a left-branching recursive partitioning. In this way, we can induce a 'FA/sDCP' hybrid grammar, with the first component having finite-state power, which means we can parse in linear time.

7 Experiments

The theory developed above shows that hybrid grammars allow considerable flexibility in the first component, leading to a wide range of different time complexities of parsing while, at least potentially, the same kinds of discontinuous structures can be obtained. We have run experiments to measure what impact different choices of the first component have on recall/precision and the degree of discontinuity.

The training data consisted of the first 7000 trees of the TIGER treebank (Brants et al., 2004). From these, recursive partitionings were straightforwardly obtained, and transformed for different values of k. Also the left-branching and right-branching recursive partitionings were considered. Hybrid grammars were then extracted using strict or child labeling. Probabilities of rules were determined by relative frequency estimation, without any smoothing techniques.

Test sentences were taken from the next 500 trees, excluding sentences of length greater than 20 and those where a single tree did not span the entire sentence, leaving 324 sentences. Parsing was on (gold standard) parts of speech rather than words. All punctuation was ignored. Labeled recall, precision and F-measure were computed on objects each consisting of the label of a node and a sequence of pairs of input positions delimiting substrings covered by that node. The algorithms were implemented in Python and the experiments were carried out on a desktop with four 3.1GHz Intel Core i5 CPUs.

Results are reported in Table 1. The choice of $k = 1$ can be seen as a baseline, the first component then being restricted to context-free power. Note that $k = 1, 2, 3$ imply parsing complexities $\mathcal{O}(n^3)$, $\mathcal{O}(n^6)$, $\mathcal{O}(n^9)$, respectively.

In the case of strict labeling, the change from $k = 1$ to $k = 2$ leads to significant changes in running time, but that from $k = 2$ to $k = 3$ less so, which can be explained by the smaller number of constituents that have two gaps, compared to those with zero or one gap. There was no significant change, neither in running time nor in F-measure, for values of k greater than 4, and therefore these values were omitted

here. Note that for $k = \infty$ one would obtain the conventional technique of discontinuous parsing using LCFRSs. For the right-branching recursive partitionings, the running time is significantly higher than that for the left-branching ones, although it is linear-time in both cases. This is due to the directional bias of the implemented parsing strategy. In order to allow a straightforward comparison we have taken the same parsing strategy in all cases. Note the large number of parse failures for the right-branching and left-branching partitionings, which is explained by the large number of very specific nonterminals.

Child labeling leads to much smaller numbers of nonterminals, and thereby also to more ambiguity, and as a result the increase from time complexity $\mathcal{O}(n^3)$ to $\mathcal{O}(n^6)$ is more noticeable in terms of the actual running time. Therefore carrying out the experiment for $k \geq 3$ was outside our reach. Surprisingly, the right-branching partitioning performed very well in this case, with a relatively low number of parse failures, F-measure competing with $k = 1, 2, 3, 4$ and strict labeling, although it is clearly worse than that with $k = 1, 2$ and child labeling, and running time smaller than in the case of any of the hybrid grammars where the first component has power beyond that of finite automata.

	fail	R	P	F1	# gaps	secs
strict labeling						
$k = 1$	16	73.0	70.4	71.2	0.0075	442
$k = 2$	12	73.1	70.7	71.4	0.0111	2,580
$k = 3$	12	73.1	70.7	71.4	0.0121	2,942
$k = 4$	12	73.1	70.7	71.4	0.0127	2,828
r-branch	151	65.6	62.4	63.2	0.0118	775
l-branch	266	82.0	78.9	79.5	0.0124	24
child labeling						
$k = 1$	4	74.3	74.2	73.9	0.0120	939
$k = 2$	4	75.0	75.1	74.7	0.0125	58,164
r-branch	15	73.1	73.0	72.6	0.0117	319
l-branch	56	75.7	76.6	75.7	0.0114	183

Table 1: Number of parse failures, recall, precision, F-measure, average number of gaps per constituent, and running time.

Child labeling generally gave better F-measure than strict labeling (ignoring strict labeling and left-branching partitioning, where the many parse failures distort the recall and precision). This seems to be due to the more accurate parameter estimation that was possible for the smaller numbers of rules obtained with child labeling.

The differences in F-measure are relatively small for varying k. This can be explained by the relatively small portion of discontinuous structures in the test set. We have looked closer at discontinuity in the test set in two ways. First, we measured the average number of gaps per constituent, which in the gold standard was 0.0171. None of the hybrid grammars came close to achieving this, but we do observe that more discontinuity is obtained for higher values of k. Secondly, we reran the experiments for only the 75 sentences out of the aforementioned 324 where the gold structure had at least one discontinuous phrase. For this smaller set, F1 increases from 59.5 ($k = 1$) to 61.9 ($k = 2, 3, 4$) for strict labeling, and it increases from 64.4 ($k = 1$) to 66.5 ($k = 2$) for child labeling. This suggests that with higher k, the additional discontinuous structures found have at least some overlap with those of the gold standard. Note again that there is no a priori bound on the fanout of produced hybrid trees, even when the first component has finite-state power, but the ability to abstract away from discontinuous structures in the training set seems to be enhanced if the first component is more powerful. This is consistent with observations made by (van Cranenburgh, 2012).

8 Limitations

The theory from Section 6 does not necessarily mean that any language of hybrid trees can be induced by a HG whose first-component LCFRS has arbitrarily low fanout. We illustrate this by means of the language of hybrid trees generated by the HG of Example 4, in which the LCFRS has fanout 2. No CFG/sDCP grammar in fact exists for the same language, or in other words, the fanout of the first-component LCFRS cannot be reduced to 1, regardless of how we choose the second-component sDCP.

For a proof, assume that a CFG/sDCP grammar does exist. Let m be the maximum number of members in the right-hand side of any CFG rule. Let k be the maximum rank of any nonterminal in the second-component sDCP. Now consider a CFG/sDCP derivation for a hybrid tree with yield $a^n b^n$, where $n \geq$

$2 \cdot k \cdot m$. In a top-down traversal, identify the first CFG nonterminal occurrence that covers a substring of the input string that has a length smaller than or equal to $n/2$ and greater than k. This substring may contain occurrences of a and of b, but because its length is at most $n/2$, there will not be any pair consisting of an occurrence of a and an occurrence of b that are both part of that substring, and that have a common parent labeled S in the hybrid tree. This means that more than k tree fragments or tree nodes with missing child nodes are involved, which translate to more than k synthesized or inherited arguments, contradicting the assumptions.

9 Conclusions

We have presented hybrid grammars as a novel framework for describing languages of discontinuous syntactic structures. This framework sheds light on the relation between various existing techniques, but it also offers potential for development of novel techniques. Much of what we have shown is merely an illustration of particular instances of this framework. For example, next to the hybrid grammars discussed here, we can consider those with macro grammars as first component, or simple context-free tree grammars as second component. Many variations exist on the illustrated grammar induction technique. For example, next to our strict labeling and child labeling, one can consider approaches using latent variables, combined with expectation-maximization.

Acknowledgments

We thank the anonymous reviewers for many helpful comments.

References

T. Becker, A.K. Joshi, and O. Rambow. 1991. Long-distance scrambling and Tree Adjoining Grammars. In *Fifth EACL*, pages 21–26.

G.V. Bochmann. 1976. Semantic evaluation from left to right. *Communications of the ACM*, 19(2):55–62.

A. Böhmová, J. Hajič, E. Hajičová, and B. Hladká. 2000. The Prague dependency treebank: A tree-level annotation scenario. In A. Abeillé, editor, *Treebanks: Building and using syntactically annotated corpora*, pages 103–127. Kluwer, Dordrecht.

A. Boyd. 2007. Discontinuity revisited: An improved conversion to context-free representations. In *Linguistic Annotation Workshop, at ACL 2007*, pages 41–44.

W.S. Brainerd. 1969. Tree generating regular systems. *Information and Control*, 14:217–231.

S. Brants, S. Dipper, P. Eisenberg, S. Hansen-Schirra, E. König, W. Lezius, C. Rohrer, G. Smith, and H. Uszkoreit. 2004. TIGER: Linguistic interpretation of a German corpus. *Research on Language and Computation*, 2:597–620.

J. Bresnan, R.M. Kaplan, S. Peters, and A. Zaenen. 1982. Cross-serial dependencies in Dutch. *Linguistic Inquiry*, 13(4):613–635.

H. Bunt. 1996. Formal tools for describing and processing discontinuous constituency structure. In H. Bunt and A. van Horck, editors, *Discontinuous Constituency*, pages 63–84. Mouton de Gruyter.

R. Campbell. 2004. Using linguistic principles to recover empty categories. In *42nd Annual Meeting of the ACL*, pages 645–652.

J.D. Choi and M. Palmer. 2010. Robust constituent-to-dependency conversion for English. In *Ninth International Workshop on Treebanks and Linguistic Theories*, pages 55–66.

P. Deransart and J. Małuszynski. 1985. Relating logic programs and attribute grammars. *Journal of Logic Programming*, 2:119–155.

K. Evang and L. Kallmeyer. 2011. PLCFRS parsing of English discontinuous constituents. In *12th International Conference on Parsing Technologies*, pages 104–116.

M.J. Fischer. 1968. Grammars with macro-like productions. In *IEEE Conference Record of 9th Annual Symposium on Switching and Automata Theory*, pages 131–142.

R. Gabbard, S. Kulick, and M. Marcus. 2006. Fully parsing the Penn Treebank. In *Human Language Technology Conference of the NAACL, Main Conference*, pages 184–191.

F. Gécseg and M. Steinby. 1997. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Vol. 3*, chapter 1, pages 1–68. Springer, Berlin.

R. Giegerich. 1988. Composition and evaluation of attribute coupled grammars. *Acta Informatica*, 25:355–423.

C. Gómez-Rodríguez and G. Satta. 2009. An optimal-time binarization algorithm for linear context-free rewriting systems with fan-out two. In *47th ACL and 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 985–993.

C. Gómez-Rodríguez, M. Kuhlmann, G. Satta, and D. Weir. 2009. Optimal reduction of rule length in linear context-free rewriting systems. In *Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the ACL*, pages 539–547.

M. Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *40th ACL*, pages 136–143.

S. Kahane, A. Nasr, and O. Rambow. 1998. Pseudo-projectivity, a polynomially parsable non-projective dependency grammar. In *36th ACL and 17th International Conference on Computational Linguistics*, volume 1, pages 646–652.

K. Kallmeyer and M. Kuhlmann. 2012. A formal model for plausible dependencies in lexicalized tree adjoining grammar. In *Eleventh International Workshop on Tree Adjoining Grammar and Related Formalisms*, pages 108–116.

L. Kallmeyer and W. Maier. 2013. Data-driven parsing using probabilistic linear context-free rewriting systems. *Computational Linguistics*, 39(1):87–119.

A. Kathol and C. Pollard. 1995. Extraposition via complex domain formation. In *33rd ACL*, pages 174–180.

M. Kuhlmann. 2013. Mildly non-projective dependency grammar. *Computational Linguistics*, 39(2):355–387.

W. Lu, H.T. Ng, W.S. Lee, and L.S. Zettlemoyer. 2008. A generative model for parsing natural language to meaning representations. In *Conference on Empirical Methods in Natural Language Processing*, pages 783–792.

W. Maier and T. Lichte. 2009. Characterizing discontinuity in constituent treebanks. In P. de Groote, M. Egg, and L. Kallmeyer, editors, *14th Conference on Formal Grammar*, volume 5591 of *Lecture Notes in Artificial Intelligence*, Bordeaux, France.

M.P. Marcus, B. Santorini, and M.A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330.

J.D. McCawley. 1982. Parentheticals and discontinuous constituent structure. *Linguistic Inquiry*, 13(1):91–106.

R. McDonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *11th EACL*, pages 81–88.

S. Müller. 2004. Continuous or discontinuous constituents? a comparison between syntactic analyses for constituent order and their processing systems. *Research on Language and Computation*, 2:209–257.

J. Nivre and J. Nilsson. 2005. Pseudo-projective dependency parsing. In *43rd ACL*, pages 99–106.

J. Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Joint Conference of the 47th ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359.

O. Rambow and A.K. Joshi. 1997. A formal look at dependency grammars and phrase structure grammars with special consideration of word-order phenomena. In L. Wenner, editor, *Recent Trends in Meaning-Text Theory*. John Benjamin.

O. Rambow. 2010. The simple truth about dependency and phrase structure representations: An opinion piece. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the ACL, Main Conference*, pages 337–340.

W.C. Rounds. 1970. Mappings and grammars on trees. *Mathematical Systems Theory*, 4:257–287.

G. Satta and E. Peserico. 2005. Some computational complexity results for synchronous context-free grammars. In *Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 803–810.

S. Seifert and I. Fischer. 2004. Parsing string generating hypergraph grammars. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *2nd International Conference on Graph Transformations*, volume 3256 of *Lecture Notes in Computer Science*, pages 352–267. Springer-Verlag.

H. Seki and Y. Kato. 2008. On the generative power of multiple context-free grammars and macro grammars. *IEICE Transactions on Information and Systems*, E91-D:209–221.

H. Seki, T. Matsumura, M. Fujii, and T. Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229.

S.M. Shieber and Y. Schabes. 1990. Synchronous tree-adjoining grammars. In *Papers presented to the 13th International Conference on Computational Linguistics*, volume 3, pages 253–258.

S.M. Shieber. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–343.

K. Sima'an, R. Bod, S. Krauwer, and R. Scha. 1994. Efficient disambiguation by means of stochastic tree substitution grammars. In *International Conference on New Methods in Language Processing*, pages 50–58.

S. Stucky. 1987. Configurational variation in English. In G.J. Huck and A.E. Ojeda, editors, *Discontinuous Constituency*, volume 20 of *Syntax and Semantics*, pages 377–404. Academic Press.

A. van Cranenburgh. 2012. Efficient parsing with linear context-free rewriting systems. In *13th EACL*, pages 460–470.

K. Vijay-Shankar and A.K. Joshi. 1985. Some computational properties of tree adjoining grammars. In *23rd ACL*, pages 82–93.

K. Vijay-Shanker, D.J. Weir, and A.K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *25th ACL*, pages 104–111.