Transformation of Recursive Partitionings during the Induction of Hybrid Grammars

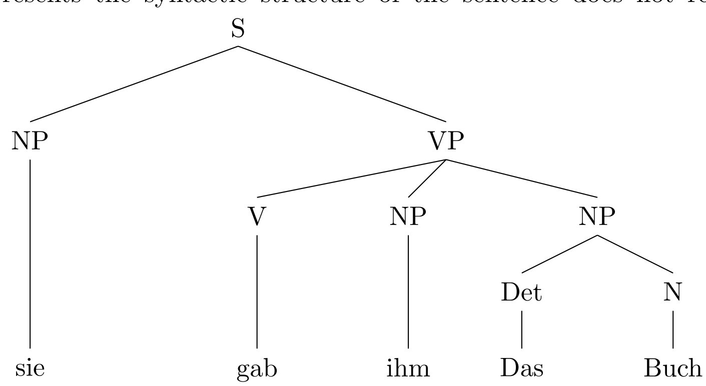
Johann Seltmann

May 28, 2017

1 Introduction

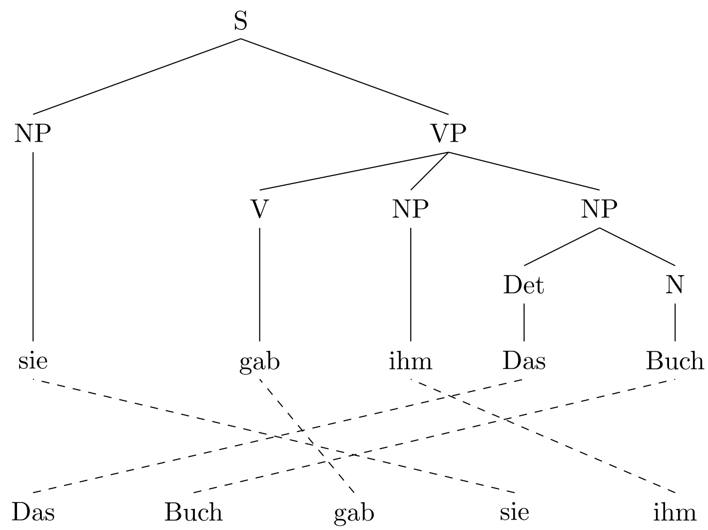
In natural language processing, syntactical structures are often represented as trees. However, in some languages, it is possible to splice a syntactical structure and put the words belonging to it at separate positions in the sentence. This phenomenon is called *discontinuity*. A classical parse tree cannot represent a discontinuous structure.

Example 1.1. In the German sentence ”Das Buch gab sie ihm” (”She gave him the book.”, literally ”The book gave she him”), the verb phrase ”gab ihm das Buch” is broken by the subject of the sentence. The tree that represents the syntactic structure of the sentence does not represent that fact:



Nederhof and Vogler [7] introduce the concepts of *hybrid trees* and *hybrid grammars* which can represent discontinuity by coupling a tree and a string.

Example 1.2. A hybrid tree that represents both the sentence from Example 1.1 with its correct word order and its parse tree.



Gebhardt et al. [5] formally introduce algorithms for the induction of hybrid grammars using *recursive partitionings*. The properties of the recursive partitioning influence the properties of the resulting grammar. Therefore, they also introduce an algorithm for the transformation of recursive partitionings in order to induce a grammar with more desirable properties. In this thesis, I explore different instances of that algorithm and how these instances influence the resulting hybrid grammar.

In Section 2 I will give basic notations and definitions. In Section 3 I will show the transformation algorithm from [5]. Then I will give an overview of my implementation (Section 4), which is based on the implementation by Gebhardt et al. [5]. Finally, I will give my experimental results in Section 5.

2 Preliminaries

Let $\mathbb{N} = \{0, 1, 2, ...\}$. For each $k \in \mathbb{N}$, let $[k] = \{1, 2, ..., k\}$ and $[k]_0 = [k] \cup \{0\}$.

A *ranked alphabet* is a tuple (Σ, rk), where Σ is a finite set and rk is a mapping $rk : \Sigma \to \mathbb{N}$. Instead of (Σ, rk), we will just write Σ. Let $k \in \mathbb{N}$. Then $\Sigma^{(k)}$ is the set of all $\sigma \in \Sigma$ with $rk(\sigma) = k$. An *alphabet* is a ranked alphabet Σ, where for all $\sigma \in \Sigma$, $rk(\sigma) = 0$.

We define $X = \{x_1, x_2, ...\}$ as a set of variables. For all $k > 0$ $X_k = \{x_1, ..., x_k\}$.

Let Σ be a ranked alphabet, $Y \subseteq X$. The set of all *trees over* Σ *and* Y, denoted $T_\Sigma(Y)$, is the smallest set S such that,

i) $Y \subseteq S$, and

ii) for all $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$ and $\xi_1, ..., \xi_k \in S$, $\sigma(\xi_1, ..., \xi_k) \in S$.

Instead of $T_\Sigma(\emptyset)$, we will write T_Σ.

Let $k \in \mathbb{N}$, Σ a ranked alphabet, $\sigma \in \Sigma^{(k)}$, $Y \subseteq X$, and $\xi, \xi_1, ..., \xi_k \in T_\Sigma(Y)$. Then the *set of positions of* ξ is defined as

$$pos(\xi) \quad = \quad \begin{cases} \{\varepsilon\} & \text{if } \xi \in Y \text{, and} \\ \{\varepsilon\} \cup \bigcup_{i=1}^{k} \{iw | w \in pos(\xi_i)\} & \text{if } \xi = \sigma(\xi_1, ..., \xi_k). \end{cases}$$

I will also call a position of ξ a *node* of ξ. Let $p, q \in pos(\xi)$. q is a *child* of p if there is an $i \in \mathbb{N}$ such that $q = pi$. Then p is called the *parent* of q. A *leaf* is a node of ξ that has no children. Let $p \in pos(\xi)$. The *subtree of* ξ *at position* p, denoted by $\xi|_p$, is defined recursively as

$$\xi|_p \quad = \quad \begin{cases} \xi & \text{if } p = \varepsilon \text{, and} \\ \xi_i|_w & \text{if for some } i \in [k] \\ & \text{ and } w \in pos(\xi_i),\ p = iw \text{ and } \xi = \sigma(\xi_1, ..., \xi_k). \end{cases}$$

The *label of* ξ *at position* p, denoted by $\xi(p)$, is defined as

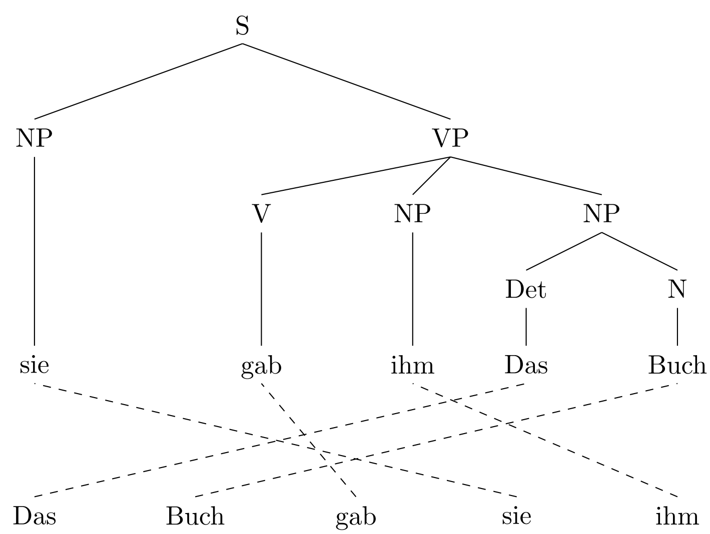
$$\xi(p) \quad = \quad \begin{cases} \sigma & \text{if } p = \varepsilon \text{ and } \xi = \sigma(\xi_1, ..., \xi_k) \text{, and} \\ \xi_i|_w & \text{if for some } i \in [k] \\ & \text{ and } w \in pos(\xi_i),\ p = iw \text{ and } \xi = \sigma(\xi_1, ..., \xi_k). \end{cases}$$

Let $Y \subseteq X$, $\xi = \sigma(\xi_1, ..., \xi_k), \zeta \in T_\Sigma(Y)$, and $p \in pos(\xi)$. Then the substitution of ξ at position ζ with ζ, denoted by $\xi[\zeta]_p$, is defined as

$$\xi[\zeta]_p \quad = \quad \begin{cases} \zeta & \text{if } p = \varepsilon \text{, and} \\ \sigma(\xi_1, ..., \xi_{j-1}, \xi_j[\zeta]_q, \xi_{j+1}, ...\xi_k) & \text{if for some } j \in [k] \\ & \text{ and } q \in pos(\xi_j),\ p = jq. \end{cases}$$

Hybrid trees are introduced by Nederhof and Vogler [7]. We will be using a notation without s-terms. Let Σ be a ranked alphabet and $\Gamma \subseteq \Sigma$. A *hybrid tree over* (Γ, Σ) is a tuple $h = (\xi, \leq_\xi)$ where $\xi \in T_\Sigma$ and $\leq_\xi$ is a total order on $pos_\Gamma(\xi) = \{p \in pos(\xi) | \xi(p) \in \Gamma\}$. A hybrid tree h is a dependency structure if $\Gamma = \Sigma$. Let $pos_\Gamma(\xi) = \{p_1, ..., p_n\}$ with $p_i \leq_\xi p_{i+1}$ for $i \in [n-1]$. Then $str(h) = \xi(p_1)...\xi(p_n)$.

Example 2.1. This is an example for a hybrid tree $h = (\xi, \leq_\xi)$ where ξ is the tree from Example 1.1 and $\leq_\xi$ is represented by the string "Das Buch gab sie ihm" at the bottom.



2.1 Hybrid grammars

Let Σ be a ranked alphabet and $\Gamma \subseteq \Sigma$. A hybrid grammar over (Γ, Σ) combines a tree grammar and a string grammar such that for each tree ξ produced by the tree grammar, the string grammar produces a string s with $s = str(h)$ for a hybrid tree $h = (\xi, \leq_\xi)$.

Gebhardt et al. [5] describe several possible combinations of string and tree grammars for hybrid grammars. We will be focusing on hybrid grammars, that combine linear context-free rewriting systems (LCFRS) and simple definite clause programs (sDCP), here, since that is the type of hybrid grammar used in the implementation. In order to define this special kind of hybrid grammar, I first recall the definitions of LCFRS and sDCP.

2.1.1 Linear context-free rewriting systems

LCFRS were first introduced by Vijay-Shanker et al. [10]. We will be using the notation of [5] here.

A *linear context-free rewriting system* (LCFRS) is a tuple $G = (N, \Sigma, S, R)$ where

- N is a ranked alphabet of *nonterminals*,
- Σ is a ranked alphabet of *terminals* with $\Sigma \cap N = \emptyset$ and for all $\sigma \in \Sigma$ $rk(\sigma) = 0$,
- $S \in N^{(1)}$ is the *initial nonterminal*, and
- R is a finite set of *rules*, specified in the following.

Each rule is of the form

$$A_0(s_1, ..., s_{k_0}) \to A_1(x_1, ..., x_{m_1}), A_2(x_{m_1+1}, ..., x_{m_2}), ..., A_n(x_{m_{n-1}+1}, ..., x_{m_n})$$

where $n \in \mathbb{N}$, $i \in [n]_0$, $k_i = rk(A_i)$, $m_i = \sum_{i' \in [i]} k_{i'}$, $j \in [k_0]$, $s_j \in (\Sigma \cup (X_{m_n}))^*$, and for each $l \in [m_n]$, x_l occurs exactly once in $s_1, ..., s_{k_0}$.

Let $m \in \mathbb{N}$ and $\rho \in R$ such that ρ contains m variables. A *rule instance* of ρ can be obtained by choosing an $r_i \in \Sigma^*$ for each $i \in [m]$ and replacing both occurences of x_i in ρ with r_i.

Let $s_1, s_2 \in (\Sigma \cup X)^*$, $\rho \in R$, and $t \to r$ be a rule instance of ρ. Then the *derivation relation* is defined as $s_1 \cdot t \cdot s_2 \Rightarrow_G^\rho s_1 \cdot r \cdot s_2$.

The string language produced by an LCRFS $G = (N, \Sigma, S, R)$ is $L(G) = \{s \in \Sigma^* | S(s) \Rightarrow_G^* \varepsilon\}$.

Let $A \in N$. We call $rk(A)$ the *fanout of A*. The *fanout of the LCFRS G* is the maximal fanout of its nonterminals.

Example 2.2. The following LCFRS G produces the string "Das Buch gab sie ihm". $G = (N, \Sigma, S, R)$, where

- $N = \{S, NP, VP, V, Det, N\}$ with $rk(VP) = 2$ and for all $A \in N \setminus \{VP\}$: $rk(A) = 1$,
- $\Sigma = \{Buch, sie, ihm, gab, Das\}$, and
- $R = \{\rho_1, ..., \rho_8\}$ as specified in the following.

$$\begin{aligned}
\rho_1 &= S(x_2 x_1 x_3) \to NP(x_1) VP(x_2, x_3) \\
\rho_2 &= VP(x_3 x_1, x_2) \to V(x_1) NP(x_2) NP(x_3) \\
\rho_3 &= NP(x_1 x_2) \to Det(x_1) N(x_2) \\
\rho_4 &= NP(sie) \to \varepsilon \\
\rho_5 &= V(gab) \to \varepsilon \\
\rho_6 &= Det(Das) \to \varepsilon \\
\rho_7 &= N(Buch) \to \varepsilon \\
\rho_8 &= NP(ihm) \to \varepsilon
\end{aligned}$$

The string can be derived as follows:

$$\begin{aligned}
& S(\text{Das Buch gab sie ihm}) \\
& \Rightarrow_G^{\rho_1} NP(\text{sie}) VP(\text{Das Buch gab}, \text{ihm}) \\
& \Rightarrow_G^{\rho_4} VP(\text{Das Buch gab}, \text{ihm}) \\
& \Rightarrow_G^{\rho_2} V(\text{gab}) NP(\text{Das Buch}) NP(\text{ihm}) \\
& \Rightarrow_G^{\rho_5} NP(\text{Das Buch}) NP(\text{ihm}) \\
& \Rightarrow_G^{\rho_8} NP(\text{Das Buch}) \\
& \Rightarrow_G^{\rho_3} Det(\text{Das}) N(\text{Buch}) \\
& \Rightarrow_G^{\rho_6} N(\text{Buch}) \\
& \Rightarrow_G^{\rho_7} \varepsilon
\end{aligned}$$

2.1.2 Simple definite clause programs

Definite clause programs are introduced in [3]. We will use a notation based on the one in [7].

A *simple definite clause program* (sDCP) is a tuple $G = (N, \Sigma, S, P)$ where

- N is a ranked alphabet of *nonterminals*,
- Σ is a ranked alphabet of *terminals*,
- $S \in N$ is the *initial nonterminal*, and
- P is a set of *rules*, described in the following.

Each $A \in N$ is assigned a number of synthesized arguments (its s-rank, denoted by $s\text{-}rk(A)$) and a number of inherited arguments (its i-rank, denoted by $i\text{-}rk(A)$) where $rk(A) = s\text{-}rk(A) + i\text{-}rk(A)$. For the initial nonterminal we demand $rk(S) = s\text{-}rk(S) = 1$. Each rule has the form

$$A_0(x_1^{(0)}, ..., x_{k_0}^{(0)}, s_1^{(0)}, ..., s_{k_0'}^{(0)}) \to \\ A_1(x_1^{(1)}, ..., x_{k_1}^{(1)}, s_1^{(1)}, ..., s_{k_1'}^{(1)}) ... A_n(x_1^{(n)}, ..., x_{k_n}^{(n)}, s_1^{(n)}, ..., s_{k_n'}^{(n)})$$

where $n \in \mathbb{N}$, $k_i = rk(A_i)$ ($i \in [n]_0$), $m = \sum_{i \in [n]_0} k_i$, $\bigcup_{j=0}^{n} \bigcup_{i=1}^{k_j} x_i^{(j)} = X_m$, $s_1^{(i)}, ..., s_{k_i'}^{(i)} \in T_\Sigma(X_m)$ ($i \in [n]_0$) such that all $s_j^{(i)}$ ($i \in [n]_0, j \in [k_i']$) together contain each variable in X_m exactly once.

Let $\rho \in P$ and $m \in \mathbb{N}$ such that ρ contains m variables. A rule instance of ρ can be obtained by choosing an $r_i \in T_\Sigma$ for each variable x_i ($i \in [m]$) and replacing all occurences of x_i with r_i.

Let $m \in \mathbb{N}$, $s_1, s_2 \in T_\Sigma(X_m)$, $\rho \in P$, and $t \to r$ a rule instance of ρ. Then the derivation relation can be defined as $s_1 t s_2 \Rightarrow_G^\rho s_1 r s_2$

The language accepted by an sDCP G is $L(G) = \{s \in T_\Sigma | s \Rightarrow_G^* \varepsilon\}$.

Example 2.3. The following sDCP G accepts the tree from Example 1.1.

$G = (N, \Sigma, S, P)$, where

- $N = \{S, NP, VP, V, Det, N\}$ where for all $A \in N : rk(A) = 1$,
- $\Sigma = \{\mathbf{S}, \mathbf{NP}, \mathbf{VP}, \mathbf{V}, \mathbf{Det}, \mathbf{N}, \mathbf{Das}, \mathbf{Buch}, \mathbf{gab}, \mathbf{sie}, \mathbf{ihm}\}$, and
- $P = \{\rho_1, ..., \rho_8\}$, as specified in the following.

$$\rho_1 = S\Big(\mathbf{S}(x_1 x_2)\Big) \to NP(x_1)VP(x_2)$$
$$\rho_2 = VP\Big(\mathbf{VP}(x_1 x_2 x_3)\Big) \to V(x_1)NP(x_2)NP(x_3)$$
$$\rho_3 = NP\Big(\mathbf{NP}(x_1 x_2)\Big) \to Det(x_1)N(x_2)$$
$$\rho_4 = NP\Big(\mathbf{NP}(\mathbf{sie})\Big) \to \varepsilon$$

$$\rho_5 = V\Big(\mathbf{V}(\mathbf{gab})\Big) \to \varepsilon$$

$$\rho_6 = Det\Big(\mathbf{Det}(\mathbf{Das})\Big) \to \varepsilon$$

$$\rho_7 = N\Big(\mathbf{N}(\mathbf{Buch})\Big) \to \varepsilon$$

$$\rho_8 = NP\Big(\mathbf{NP}(\mathbf{ihm})\Big) \to \varepsilon$$

The tree can be derived as follows:

$$S\Big(\mathbf{S}(\mathbf{NP}(\mathbf{sie}), \mathbf{VP}(\mathbf{V}(\mathbf{gab}), \mathbf{NP}(\mathbf{ihm}), \mathbf{NP}(\mathbf{Det}(\mathbf{Das}), \mathbf{N}(\mathbf{Buch}))))\Big)$$
$$\Rightarrow_G^{\rho_1} NP\Big(\mathbf{NP}(\mathbf{sie})\Big), VP\Big(\mathbf{VP}(\mathbf{V}(\mathbf{gab}), \mathbf{NP}(\mathbf{ihm}), \mathbf{NP}(\mathbf{Det}(\mathbf{Das}), \mathbf{N}(\mathbf{Buch})))\Big)$$
$$\Rightarrow_G^{\rho_4} VP\Big(\mathbf{VP}(\mathbf{V}(\mathbf{gab}), \mathbf{NP}(\mathbf{ihm}), \mathbf{NP}(\mathbf{Det}(\mathbf{Das}), \mathbf{N}(\mathbf{Buch})))\Big)$$
$$\Rightarrow_G^{\rho_2} V\Big(\mathbf{V}(\mathbf{gab})\Big)NP\Big(\mathbf{NP}(\mathbf{ihm})\Big)NP\Big(\mathbf{NP}(\mathbf{Det}(\mathbf{Das}), \mathbf{N}(\mathbf{Buch}))\Big)$$
$$\Rightarrow_G^{\rho_5} NP\Big(\mathbf{NP}(\mathbf{ihm})\Big)NP\Big(\mathbf{NP}(\mathbf{Det}(\mathbf{Das}), \mathbf{N}(\mathbf{Buch}))\Big)$$
$$\Rightarrow_G^{\rho_8} NP\Big(\mathbf{NP}(\mathbf{Det}(\mathbf{Das}), \mathbf{N}(\mathbf{Buch}))\Big)$$
$$\Rightarrow_G^{\rho_3} Det\Big(\mathbf{Det}(\mathbf{Das})\Big)N\Big(\mathbf{N}(\mathbf{Buch})\Big)$$
$$\Rightarrow_G^{\rho_6} N\Big(\mathbf{N}(\mathbf{Buch})\Big)$$
$$\Rightarrow_G^{\rho_7} \varepsilon$$

2.1.3 LCFRS/sDCP hybrid grammars

Hybrid grammars are introduced by [7]. Here, we are using the notation of [5]

In order to link an LCFRS and an sDCP together we index the terminals and nonterminals of both grammars. Let Ω be a ranked alphabet. The ranked alphabet $I(\Omega)$ is defined as $I(\Omega) = \{\omega^{\boxed{u}} | \omega \in \Omega, u \in \mathbb{N}_+\}$ with $rk_{I(\Omega)}(\omega^{\boxed{u}}) = rk_\Omega(\omega)$. Let Δ be a ranked alphabet such that $\Delta \cap \Omega = \emptyset$ and $Y \subseteq X$. We define $I_{\Omega,\Delta}(Y)$ as the set of all $t \in T_{I(\Omega)\cup\Delta}(Y)$ where each index occurs at most once. Instead of $I_{\Omega,\Delta}(\emptyset)$, I will be writing $I_{\Omega,\Delta}$. The *deindexing function* D removes all indices from a tree $t \in T_{I(\Omega)\cup\Delta}(Y)$. For a tree $t \in T_{I(\Omega)\cup\Delta}(Y)$, $ind(t)$ is the set of all indices occuring in t. Let $k \in \mathbb{N}$ and $t_1, ..., t_k \in T_{I(\Omega)\cup\Delta}(Y)$. Then, $D(t_1...t_k) = D(t_1)...D(t_k)$ and $ind(t_1...t_k) = \bigcup_{i\in[k]} ind(t_k)$. For a string $s \in (I(\Omega) \cup \Delta)^*$, $D(s)$ and $ind(s)$ are defined in the same way.

An *LCFRS/sDCP hybrid grammar* is a tuple $G = (N, S, (\Gamma, \Sigma), P)$ where

- N is an alphabet of *nonterminals*,
- Γ, Σ are ranked alphabets of *terminals* with $\Gamma \subseteq \Sigma^{(0)}$,
- $S \in N$ is the *initial nonterminal*, and
- P is a set of *rules*, described in the following.

Let Δ be the ranked alphabet $\Sigma \setminus \Gamma$, where for each $\delta \in \Delta$, $rk_\Delta(\delta) = rk_\Sigma(\delta)$. Let $n, k, l, m, p, q \in \mathbb{N}$, $Y, Z \subseteq X$, $s_1, ..., s_k \in (\Gamma \cup Y)^*$, $t_1, ..., t_l \in I_{\Gamma,\Sigma}(Z)$, $i_1, ..., i_p \in I_{N,\emptyset}(Y)$, $r_1 = i_1...i_p$, $i'_1, ..., i'_q \in I_{N\cup\Gamma,\Delta}(Z)$, and $r_2 = i'_1...i'_q$. Each rule is of the form $[A(s_1, ..., s_k) \to r_1, A(t_1, ..., t_l) \to r_2]$ such that $D(A(s_1, ..., s_k)) \to D(r_1)$ has the form of an LCFRS rule and $D(A(t_1, ..., t_l)) \to D(r_2)$ has the form of an sDCP rule. We also demand that each index couples two identical symbols. Let P_1 be the set of all $D(A(s_1, ..., s_k)) \to D(r_1)$ where $A(s_1, ..., s_k) \to r_1$ is the first component of a rule in P. P_2 is similarly defined for the second component of rules. We demand that (N, S, Γ, P_1) is an LCFRS and (N, S, Σ, P_2) is an sDCP. These two grammars are called the *first* and *second component* of the hybrid grammar.

In order to define the derivation relation, we need to define reindexing functions for terminals and nonterminals. The reason for this is that one rule can be applied multiple times during the derivation of a word. This can lead to a situation where, in a sentential form of the first or second component of a hybrid grammar, there are multiple occurrences of the same symbol with the same index. To avoid such a situation the indices need to be reassigned when applying a rule. For the nonterminal reindexing function we define a set $U \subseteq \mathbb{N}_+$ of existing indices. The *nonterminal reindexing function* f replaces indices in each rule such that the new indices are not already contained in U. A *terminal reindexing function* replaces the indices at the terminals in the rule with the indices of the corresponding nonterminals in the sentential form.

We can now define the derivation relation as follows. We let

$$[s''_1...s''_{i_s} A^{\boxed{u}}(s'_1, ..., s'_k)s''_{i_s+1}...s''_{j_s}, t''_1...t''_{i_t} A^{\boxed{u}}(t'_1, ..., t'_k)t''_{i_t+1}...t''_{j_t}] \Rightarrow^{u,g}_G$$
$$[s''_1...s''_{i_s} \cdot r'_1 \cdot s''_{i_s+1}...s''_{j_s}, t''_1...t''_{i_t} r'_2 t''_{i_t+1}...t''_{j_t}]$$

for every $i_s, j_s, i_t, j_t \in \mathbb{N}$ (with $i_s \leq j_s, i_t \leq j_t$), $s''_1, ..., s''_{j_s} \in I_{N\cup\Gamma,\emptyset}$, $t''_1, ..., t''_{j_t} \in I_{N\cup\Gamma,\Delta}$, $s'_1, ..., s'_k \in I(\Gamma)^*$, $t'_1, ..., t'_k \in I_{\Gamma,\Delta}$, if:

- $[A(s_1, ..., s_k) \to r_1, A(t_1, ..., t_l) \to r_2] \in P$,
- $\exists U \subseteq \mathbb{N}$, such that $U = ind(s''_1...s''_{i_s} A^{\boxed{u}}(s'_1, ..., s'_k)s''_{i_s+1}...s''_{j_s}) \setminus \{u\} = ind(t''_1...t''_{i_t} A^{\boxed{u}}(t'_1, ..., t'_k)t''_{i_t+1}...t''_{j_t}) \setminus \{u\}$,
- there is a terminal reindexing function g such that $g(s_1, ..., s_k) = (s'_1, ..., s'_k)$,
- $A(s'_1, ..., s'_k) \to r'_1$ is obtained from $g(f_U(A(s_1, ..., s_k) \to r_1))$ by replacing variables with strings in $I(\Gamma)^*$, and
- $A(t'_1, ..., t'_k) \to r'_2$ is obtained from $g(f_U(A(t_1, ..., t_k) \to r_2))$ by replacing variables with trees in $I_{\Gamma,\Delta}$.

The language accepted by an LCFRS/sDCP hybrid grammar G is the set of all hybrid trees $h = (\xi, \leq_\xi)$ with $[str(\xi), \xi] \Rightarrow^*_G [\varepsilon, \varepsilon]$

Example 2.4. We can obtain a hybrid grammar that accepts the hybrid tree from Example 2.1 by combining the LCFRS from Example 2.2 and the sDCP from Example 2.3. I will only give the set P of hybrid rules here.

$$\rho_1 = [S(x_2x_1x_3) \to NP^{\boxed{1}}(x_1)VP^{\boxed{2}}(x_2, x_3), S\Big(\mathbf{S}(x_1x_2)\Big) \to NP^{\boxed{1}}\Big(x_1\Big)VP^{\boxed{2}}\Big(x_2\Big)]$$

$$\begin{aligned}\rho_2 = [&VP(x_3x_1, x_2) \to V^{\boxed{1}}(x_1)NP^{\boxed{2}}(x_2)NP^{\boxed{3}}(x_3),\\ &VP\Big(\mathbf{VP}(x_1x_2x_3)\Big) \to V^{\boxed{1}}\Big(x_1\Big)NP^{\boxed{2}}\Big(x_2\Big)NP^{\boxed{3}}\Big(x_3\Big)]\end{aligned}$$

$$\rho_3 = [NP(x_1x_2) \to Det^{\boxed{1}}(x_1), N^{\boxed{2}}(x_2), NP\Big(\mathbf{NP}(x_1x_2)\Big) \to Det^{\boxed{1}}\Big(x_1\Big)N^{\boxed{2}}\Big(x_2\Big)]$$

$$\rho_4 = [NP(\mathbf{sie}^{\boxed{1}}) \to \varepsilon, NP\Big(\mathbf{NP}(\mathbf{sie}^{\boxed{1}})\Big) \to \varepsilon]$$

$$\rho_5 = [V(\mathbf{gab}^{\boxed{1}}) \to \varepsilon, V\Big(\mathbf{V}(\mathbf{gab}^{\boxed{1}})\Big) \to \varepsilon]$$

$$\rho_6 = [Det(\mathbf{Das}^{\boxed{1}}) \to \varepsilon, Det\Big(\mathbf{Det}(\mathbf{Das}^{\boxed{1}})\Big) \to \varepsilon]$$

$$\rho_7 = [N(\mathbf{Buch}^{\boxed{1}}) \to \varepsilon, N\Big(\mathbf{N}(\mathbf{Buch}^{\boxed{1}})\Big) \to \varepsilon]$$

$$\rho_8 = [NP(\mathbf{ihm}^{\boxed{1}}) \to \varepsilon, NP\Big(\mathbf{NP}(\mathbf{ihm}^{\boxed{1}})\Big) \to \varepsilon]$$

The hybrid tree can be derived as follows (the words in the string are abbreviated by their first letter):

$$\begin{aligned}
&[S^{\boxed{1}}(\mathbf{D}^{\boxed{2}}\mathbf{B}^{\boxed{3}}\mathbf{g}^{\boxed{4}}\mathbf{s}^{\boxed{5}}\mathbf{i}^{\boxed{6}}),\\
&\qquad S^{\boxed{1}}\Big(\mathbf{S}(\mathbf{NP}(\mathbf{s}^{\boxed{5}}), \mathbf{VP}(\mathbf{V}(\mathbf{g}^{\boxed{4}}), \mathbf{NP}(\mathbf{i}^{\boxed{6}}), \mathbf{NP}(\mathbf{Det}(\mathbf{D}^{\boxed{2}}), \mathbf{N}(\mathbf{B}^{\boxed{3}}))))\Big)]\\
&\Rightarrow_G^{\boxed{1},\rho_1} [NP^{\boxed{1}}(\mathbf{s}^{\boxed{5}})VP^{\boxed{7}}(\mathbf{D}^{\boxed{2}}\mathbf{B}^{\boxed{3}}\mathbf{g}^{\boxed{4}}, \mathbf{i}^{\boxed{6}}),\\
&\qquad NP^{\boxed{1}}\Big(\mathbf{NP}(\mathbf{s}^{\boxed{5}})\Big)VP^{\boxed{7}}\Big(\mathbf{VP}(\mathbf{V}(\mathbf{g}^{\boxed{4}}), \mathbf{NP}(\mathbf{i}^{\boxed{6}}), \mathbf{NP}(\mathbf{Det}(\mathbf{D}^{\boxed{2}}), \mathbf{N}(\mathbf{B}^{\boxed{3}})))\Big)]\\
&\Rightarrow_G^{\boxed{1},\rho_4} [VP^{\boxed{7}}(\mathbf{D}^{\boxed{2}}\mathbf{B}^{\boxed{3}}\mathbf{g}^{\boxed{4}}, \mathbf{i}^{\boxed{6}}),\\
&\qquad VP^{\boxed{7}}\Big(\mathbf{VP}(\mathbf{V}(\mathbf{g}^{\boxed{4}}), \mathbf{NP}(\mathbf{i}^{\boxed{6}}), \mathbf{NP}(\mathbf{Det}(\mathbf{D}^{\boxed{2}}), \mathbf{N}(\mathbf{B}^{\boxed{3}})))\Big)]\\
&\Rightarrow_G^{\boxed{7},\rho_2} [V^{\boxed{1}}(\mathbf{g}^{\boxed{4}})NP^{\boxed{8}}(\mathbf{D}^{\boxed{2}}\mathbf{B}^{\boxed{3}})NP^{\boxed{9}}(\mathbf{i}^{\boxed{6}}),\\
&\qquad V^{\boxed{1}}\Big(\mathbf{V}(\mathbf{g}^{\boxed{4}})\Big)NP^{\boxed{9}}\Big(\mathbf{NP}(\mathbf{i}^{\boxed{6}})\Big)NP^{\boxed{8}}\Big(\mathbf{NP}(\mathbf{Det}(\mathbf{D}^{\boxed{2}}), \mathbf{N}(\mathbf{B}^{\boxed{3}}))\Big)]\\
&\Rightarrow_G^{\boxed{1},\rho_5} [NP^{\boxed{8}}(\mathbf{D}^{\boxed{2}}\mathbf{B}^{\boxed{3}})NP^{\boxed{9}}(\mathbf{i}^{\boxed{6}}),\\
&\qquad NP^{\boxed{9}}\Big(\mathbf{NP}(\mathbf{i}^{\boxed{6}})\Big)NP^{\boxed{8}}\Big(\mathbf{NP}(\mathbf{Det}(\mathbf{D}^{\boxed{2}}), \mathbf{N}(\mathbf{B}^{\boxed{3}}))\Big)]\\
&\Rightarrow_G^{\boxed{9},\rho_8} [NP^{\boxed{8}}(\mathbf{D}^{\boxed{2}}\mathbf{B}^{\boxed{3}}),\ \ NP^{\boxed{8}}\Big(\mathbf{NP}(\mathbf{Det}(\mathbf{D}^{\boxed{2}}), \mathbf{N}(\mathbf{B}^{\boxed{3}}))\Big)]\\
&\Rightarrow_G^{\boxed{8},\rho_3} [Det^{\boxed{1}}(\mathbf{D}^{\boxed{2}})N^{\boxed{10}}(\mathbf{B}^{\boxed{3}}),\ \ Det^{\boxed{1}}\Big(\mathbf{Det}(\mathbf{D}^{\boxed{2}})\Big)N^{\boxed{10}}\Big(\mathbf{N}(\mathbf{B}^{\boxed{3}})\Big)]
\end{aligned}$$

$\Rightarrow_G^{\boxed{1},\rho_6} [N^{\boxed{10}}(\mathbf{B}^{\boxed{3}}),\ \ N^{\boxed{10}}\Big(\mathbf{N}(\mathbf{B}^{\boxed{3}})\Big)]$

$\Rightarrow_G^{\boxed{10},\rho_7} [\varepsilon,\ \ \varepsilon]$

Here we used the following reindexing functions:

rule used	nonterminal reindexing	terminal reindexing
ρ_1	$f_{\{2,...,6\}}(2) = 7$	g identity
ρ_4	$f_{\{2,...,7\}}$ identity	$g(1) = 5$
ρ_2	$f_{\{2,3,4,6,7\}}(2) = 8$ $f_{\{2,3,4,6,7\}}(3) = 9$	g identity
ρ_5	$f_{\{2,3,6,8,9\}}$ identity	$g(1) = 4$
ρ_8	$f_{\{2,3,6,8\}}$ identity	$g(1) = 4$
ρ_3	$f_{\{2,3,8\}}(2) = 10$	g identity
ρ_6	$f_{\{1,2,3,10\}}$ identity	$g(1) = 2$
ρ_7	$f_{\{3,10\}}$ identity	$g(1) = 3$

2.2 Induction of hybrid grammars

Gebhardt et al. [5] formalize probabilistic hybrid grammars (already mentioned in [7]) by assigning probabilities to rules. Let $G = (N, S, (\Gamma, \Sigma), P)$ be a hybrid grammar, $A \in N$, and $P_A \subseteq P$ the set of all $\rho \in P$ where A is the nonterminal in the left-hand sides of the first and second component of ρ. G is proper if for all $B \in N$, $\sum_{\rho \in P_B} p(\rho) = 1$, where $p(\rho)$ is the probability of ρ. We demand that a probabilistic hybrid grammar is proper.

In order to induce a hybrid grammar on a corpus of hybrid trees $(\xi, \leq_\xi)$ they define recursive partitionings as a way to add structural information to $str(\xi)$.

2.2.1 Recursive Partitionings

Recursive partitionings were introduced by Nederhof and Vogler [7].

A *recursive partitioning* is a tree $\pi \in T_{P(\mathbb{N})}$ whose nodes are labeled with sets of natural number, such that

1. the root is labeled with $\{1, ..., n\}$ for some $n \in \mathbb{N}$,
2. all leaves are labeled with a set of size one,
3. the label of each non-leaf node is the union of the labels of its children,
4. the labels of the children of each non-leaf node are disjoint, and
5. each non-leaf node has at least two children.

Let $n, m \in \mathbb{N}$ with $n < m$. The *span of* n *and* m, denoted by $span(n, m)$, is the set $\{n, n + 1, ..., m\}$.

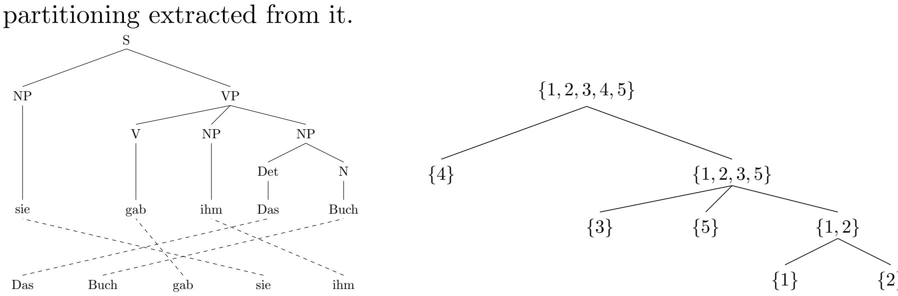
Let J be the label of a node of a recursive partitioning. The *fanout* of J is the smallest number k of sets J_i such that $J = \bigcup_{i=1}^{k} J_i$ and for each $i \in [k]$, there are $n, m \in \mathbb{N}$, such that $J_i = span(n, m)$. The *fanout of a recursive partitioning* is the maximal fanout of the labels of its nodes.

2.2.2 Extraction of a recursive partitioning from a hybrid tree

In order to obtain a recursive partitioning π that reflects the structure of a hybrid tree $(\xi, \leq_\xi)$, Gebhardt et al. [5] introduce an extraction algorithm (Algorithm 6.2 in [5]).

Let Γ, Σ be ranked alphabets, such that $\Gamma \subseteq \Sigma^{(0)}$, and $h = (\xi, \leq_\xi)$ a hybrid tree over Γ and Σ. The algorithm replaces the label of each node whose label is in Γ with a number according to the order imposed on these nodes by $\leq_\xi$. The label of each non-leaf node is replaced by the union of the numbers assigned to its children (and to itself, if its label is in Γ as well). Non-leaf nodes whose labels have size one are removed.

Example 2.5. This is the hybrid tree from Example 2.1 and the recursive partitioning extracted from it.



2.2.3 Induction of a hybrid grammar from a hybrid tree and a recursive partitioning

For the induction of hybrid grammars, Gebhardt et al. [5] introduce algorithms for the induction of an LCFRS and an sDCP that each contains indices that couple the two grammars.

Algorithm 6.1 in [5] induces an LCFRS from a string s with length l and a recursive partitioning π where the label of the root of π is $[l]$. The labels of the nodes of π become the nonterminals of the LCFRS. The algorithm creates one rule out of each node of the recursive partitioning. The label of the node is made the nonterminal in the left-hand side, while the labels of the children of that node become the nonterminals in the right-hand side. The variables in the left-hand side argument are ordered in a way that in a derivation step each terminal of s is assigned to the corresponding nonterminal in the left-hand side. Corresponding means here that the nonterminal contains the number corresponding to the position of the terminal in s. For the leaves, the nonterminals in the left-hand side of the rule take one argument which is the terminal in s that corresponds to the number in the nonterminal. The right-hand side of these rules is just ε.

Example 2.6. With the recursive partitioning from Example 2.5 and the string "Das Buch gab sie ihm", Algorithm 6.1 from [5] induces the following LCFRS: $G = (N, \Sigma, S, R)$, where

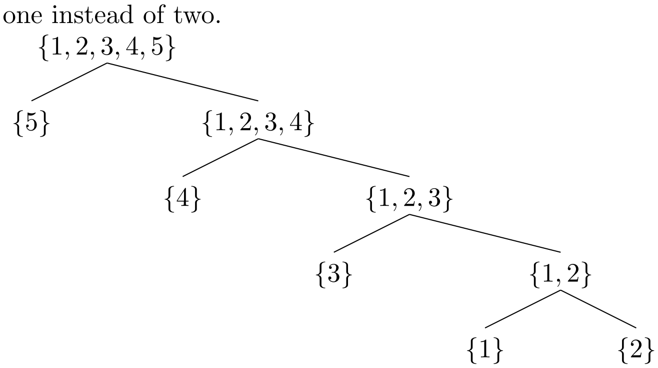
- $N = \{(\!|\{1,2,3,4,5\}|\!), (\!|\{4\}|\!), (\!|\{1,2,3,5\}|\!), (\!|\{3\}|\!), (\!|\{5\}|\!), (\!|\{1,2\}|\!), (\!|\{1\}|\!), (\!|\{2\}|\!)\}$,

- $\Sigma = \{Buch, sie, ihm, gab, Das\}$,
- $S = \llparenthesis\{1,2,3,4,5\}\rrparenthesis$,
- $R = \{\rho_1, ..., \rho_8\}$ as specified in the following.

$$\begin{aligned}
\rho_1 &= \llparenthesis\{1,2,3,4,5\}\rrparenthesis(x_2 x_1 x_3) \to \llparenthesis\{4\}\rrparenthesis(x_1)\llparenthesis\{1,2,3,5\}\rrparenthesis(x_2, x_3)\\
\rho_2 &= \llparenthesis\{1,2,3,5\}\rrparenthesis(x_3 x_1, x_2) \to \llparenthesis\{3\}\rrparenthesis(x_1)\llparenthesis\{5\}\rrparenthesis(x_2)\llparenthesis\{1,2\}\rrparenthesis(x_3)\\
\rho_3 &= \llparenthesis\{1,2\}\rrparenthesis(x_1 x_2) \to Det(x_1)\llparenthesis\{2\}\rrparenthesis(x_2)\\
\rho_4 &= \llparenthesis\{4\}\rrparenthesis(sie) \to \varepsilon\\
\rho_5 &= \llparenthesis\{3\}\rrparenthesis(gab) \to \varepsilon\\
\rho_6 &= \llparenthesis\{1\}\rrparenthesis(Das) \to \varepsilon\\
\rho_7 &= \llparenthesis\{2\}\rrparenthesis(Buch) \to \varepsilon\\
\rho_8 &= \llparenthesis\{5\}\rrparenthesis(ihm) \to \varepsilon
\end{aligned}$$

Notice that the nonterminal $\llparenthesis\{1,2,3,5\}\rrparenthesis$ has a fanout of two. This is because the label of the corresponding node in the recursive partitioning also has fanout two. Therefore, we can induce an LCFRS with a lower fanout by using a different recursive partitioning. This is desirable because a lower fanout leads to a lower parsing complexity.

Take, for example, the following recursive partitioning which has a fanout of one instead of two.



With it, the algorithm returns the following LCFRS which has a fanout of one. (I give only the rules of the LCFRS here):

$$\begin{aligned}
\rho_1 &= \llparenthesis\{1,2,3,4,5\}\rrparenthesis(x_2 x_1) \to \llparenthesis\{5\}\rrparenthesis(x_1)\llparenthesis\{1,2,3,4\}\rrparenthesis(x_2)\\
\rho_2 &= \llparenthesis\{1,2,3,4\}\rrparenthesis(x_2 x_1) \to \llparenthesis\{4\}\rrparenthesis(x_1)\llparenthesis\{1,2,3\}\rrparenthesis(x_2)\\
\rho_3 &= \llparenthesis\{1,2,3\}\rrparenthesis(x_2 x_1) \to \llparenthesis\{3\}\rrparenthesis(x_1)\llparenthesis\{1,2\}\rrparenthesis(x_2)\\
\rho_4 &= \llparenthesis\{1,2\}\rrparenthesis(x_1 x_2) \to \llparenthesis\{1\}\rrparenthesis(x_1)\llparenthesis\{2\}\rrparenthesis(x_2)\\
\rho_5 &= \llparenthesis\{4\}\rrparenthesis(sie) \to \varepsilon\\
\rho_6 &= \llparenthesis\{3\}\rrparenthesis(gab) \to \varepsilon
\end{aligned}$$

$$\rho_7 = \{\!|\{1\}|\!\}(Das) \to \varepsilon$$
$$\rho_8 = \{\!|\{2\}|\!\}(Buch) \to \varepsilon$$
$$\rho_9 = \{\!|\{5\}|\!\}(ihm) \to \varepsilon$$

Algorithm 6.5 in [5] induces an sDCP from a dependency structure $h = (\xi, \leq_\xi)$ and a recursive partitioning π. It uses a construction similar to the algorithm for induction of LCFRS.

When inducing a grammar from a corpus of trees, the nonterminals produced from each tree are renamed in order to make rules and nonterminals less dependent on one specific tree. For this, Nederhof and Vogler [7, page 1376] introduce *strict* and *child labeling* as naming strategies. These can be used in combination with POS, DEPREL, and POS+DEPREL labeling.

3 The algorithm for the transformation of recursive partitionings

Gebhardt et al. [5] introduce an algorithm for the transformation of recursive partitionings, such that an arbitrary recursive partitioning is transformed into a recursive partitioning with a fanout less than or equal to a given value k (Algorithm 6.3 in [5]). The goal is to obtain a recursive partitioning that has the required fanout (to lead to a grammar with lower parsing complexity) but also retains the structure of the original recursive partitioning as far as possible. I slightly adapted this algorithm to fit my definition of recursive partitionings without s-terms in Algorithm 1.

The algorithm takes a recursive partitioning $\pi = J(\pi_1, ..., \pi_n)$, where J is the label of the root and $\pi_1, ..., \pi_n$ are the subtrees of the root. It performs a search among the positions of π until it finds a $p \in pos(\pi)$ such that the sets $\pi(p)$ and $J \setminus \pi(p)$ each have a fanout less than or equal to k (line 5). Such a position always exists ([5, page 29]).

When the algorithm has found such a $p \in pos(\pi)$, it creates a new tree π'' by removing all elements of $\pi(p)$ from π (line 7). For this, the function REMOVE (line 10 - line 23) is called. It then obtains the transformed recursive partitioning by recursively transforming $\pi|_p$ and π'' and using them as subtrees (line 8).

In line 5, Gebhardt et al. [5] choose p according to a breadth-first search. However, it is also possible to use another strategy for choosing p. My task is to implement different strategies for choosing p and to compare the grammars induced with the recursive partitionings obtained with these strategies.

4 Implementation

For their implementation of the induction of hybrid grammars, Gebhardt et al. [5] implemented Algorithm 1 with right-to-left breadth-first search for line 5. Recursive partitionings are implemented as tuples consisting of the label of the

Algorithm 1 Transformation of a recursive partitioning

Require:
a recursive partitioning π
a maximal fanout $k \in \mathbb{N}$
Ensure: a recursive partitioning with fanout $\leq k$

function TRANSFORM($\pi = J(\pi_1, ..., \pi_n)$, k)
  if $|J| = 1$ **then**
    return π
  end if
  $p \leftarrow$ breadth-first search for some $p \in pos(\pi) \setminus \{\varepsilon\}$
    $\hookrightarrow$ such that $\pi(p)$ and $J \setminus \pi(p)$ each have fanout $\leq k$
  $\pi' \leftarrow \pi|_p$
  $\pi'' \leftarrow$ REMOVE(π, $\pi(p)$)
  return J(TRANSFORM(π', k), TRANSFORM(π'', k))
end function

function REMOVE($\pi = J(\pi_1, ..., \pi_k)$, I)
  $\triangleright$ Removes all occurences of members of the set I from π.
  $J' \leftarrow J \backslash I$
  $j \leftarrow 0$
  for $i = 1, ..., k$ **do**
    $\pi' \leftarrow$ REMOVE(π_i, I)
    if $\pi'(\varepsilon) = J'$ **then**
      return π'
    else if $\pi'(\varepsilon) \neq \emptyset$ **then**
      $j++$
      $\pi'_j \leftarrow \pi'$
    end if
  end for
  return $J'(\pi'_1, ..., \pi'_j)$
end function

root and a list of the children of the root, which are recursively implemented in the same way. I adapt this implementation to include transformations

- with left-to-right breadth-first search,
- with choosing p such that $p = argmax_{p \in pos(\pi)} |\pi(p)|$ (meaning such that π at position p has the label with the most elements),
- with choosing p randomly, and
- with choosing p such that no new nonterminals are added if possible. If that is not possible, p is chosen according to one of the other strategies.

The functions implementing the transformations can be found in the file `lcfrs-sdcp-hybrid-grammars/decomposition.pyx`. For running the experiment, one needs to install a number of prerequisites, for which there is a wiki by Gebhardt and Teichmann [4]. Then on needs to execute the file `lcfrs-sdcp-hybrid-grammars/playground/play_recursive_partitioning.py`. Assuming one is in the playground directory, it can be called with `PYTHONPATH=.. python2.7 play_recursive_partitioning.py`.

4.1 Command-line options

I have implemented command line options for `play_recursive_partitioning.py` that determine the transformation strategy and hyperparameters. Some option can be passed multiple arguments separated by whitespace. These are `-s`, `-l`, `-t`, `-f`, `-n`, and `-r`. A call could look like this: `PYTHONPATH=.. python2.7 play_recursive_partitioning.py -s rtl argmax -l child -f 2 3`. Table 1 provides an overview over the command line options.

4.2 Output file

The output is saved in the file `results.txt`. For each combination of transformation strategy, labeling strategy, and maximal fanout, the file contains a number of lines. These lines contain:

- the current strategy, labeling, and fanout,
- the number of nonterminals and rules of the induced grammar, and
- the average number of derivation trees per sentence and the maximal number of derivation trees of any sentence.
- The next three lines contain labeled and unlabeled attachment scores with and without punctuation.
- The last line shows the parse time.

Table 1: Command line options

Option	Meaning	Possible choices	Default
`-s`	strategy for choosing p	• `rtl` for right-to-left breadth-first search • `ltr` for left-to-right breadth-first search • `argmax` for choosing $p = argmax_{p \in pos(\pi)} \lvert\pi(p)\rvert$ • `random` for random choice • `nnont` for no new nonterminals if possible	`rtl`
`-l`	labelling strategy	`strict`, `child`	`strict`
`-t`	also labelling strategy	`pos`, `deprel`, `pos+deprel`	`pos`
`-f`	maximal fanout		1
`-n`	fallback strategy for no-new-nonterminals strategy	`rtl`, `ltr`, `random`, `argmax`	`rtl`
`-r`	random seed for random choice		1
`-c`	language	`polish`, `german`	`german`
`-d`	whether or not to count derivation trees	`yes`, `no`, `y`, `n`	`yes`
`-q`	use shortened version of german dev-corpus	`yes`, `no`	`yes`
`-e`	whether or not to determine attachment scores and parse times	`yes`, `no`	`yes`

The numbers for the derivation trees and/or the attachment scores and the parse time can be missing, if play_recursive_partitioning.py was called with the respective command-line options or if the RAM was insufficient. If the RAM is insufficient for counting the derivation trees, the file also does not show the attachment scores and parse times since these are determined later in play_recursive_partitioning.py. In that case play_recursive_partitioning.py needs to be run again with the option '-d no'.

4.3 Common elements of the transformation functions

The functions implementing the various strategies share common elements which were introduced in the implementation by Gebhardt et al. [5]. I will give an overview over these common elements before describing the individual strategies.

All of the functions take a recursive partitioning (called part) and the maximal fanout (called fanout) as arguments. They search for a subtree of part such that the label of the root of that subtree (saved in the variable subroot) and the label of the root of part, with the elements of subroot removed, have a fanout less than or equal to fanout. I will refer to that condition as the *target condition*.

To do that, the functions first copy the children of the root of part into the list agenda and perform a breadth-first search on agenda. The label of the root of part is held in the variable root. For each element of agenda (child1) they check, whether the label of the root of child1 (subroot) and root fulfill the target condition.

4.4 Right-to-left breadth-first search

This strategy was implemented by Gebhardt et al. [5]. The implementation can be found in the function fanout_limited_partitioning() from lines 76 - 101. Like the functions for the other transformation strategies it performs a breadth-first search over agenda, but before that, it reverses agenda for right-to-left branching (line 85).

If the target condition is fulfilled, the function restrict_part() is called, which deletes all elements of subroot from part (line 94, corresponding to line 7 of Algorithm 1). Then fanout_limited_partitioning_left_to_right() is called recursively on child1 and the result of restrict_part(), yielding transformed trees which are respectively saved in child1_restrict and child2_restrict. The function then returns a recursive partitioning with root as the label of its root and with child1_restrict and child2_restrict as children of its root (lines 95 - 97, corresponding to line 8 of Algorithm 1).

If the target condition is not fulfilled, the children of child1 are, in right-to-left order, added to the end of the queue for the breadth-first search (line 99).

4.5 Left-to-right breadth-first search

This strategy is implemented in the function `fanout_limited_partitioning_left_to_right()` (lines 104 - 130). It works almost the same as the implementation for right-to-left breadth-first search, except that the lists `children` and `subchildren` are not reversed when added to `agenda` (lines 114 and 128).

4.6 Choice of $p = argmax_{p \in pos(\pi)} |\pi(p)|$

This strategy is implemented in the function `fanout_limited_partitioning_argmax()` (lines 133 - 165). Like the other functions it performs a breadth-first search through the subtrees of `part`, however, instead of using the first solution, all positions are checked. If `child1` fulfills the target condition, the function determines, whether one of the previous possible solutions is better than `child1`. For this strategy, better means that `subroot` has more elements than `argroot`, which is the label of the root of the previous best solution. If so, `child1` is copied into the variable `argmax` (lines 144 and 156-157). That way `argmax` holds the subtree with the most elements in the label of its root. After having searched through all subtrees, the function calls itself recursively on `argmax` to create the transformed partitioning (lines 161 - 165).

4.7 Random Choice

This strategy is implemented in the function `fanout_limited_partitioning_random_choice()` (lines 343 - 373). Like the previous functions, it performs a breadth-first search among the subtrees of the recursive partitioning. If it finds one that fulfills the target condition (line 360), the function adds it to the list `possibleChoices` (lines 352 and 361). After all items in `agenda` have been processed, an element from `possibleChoices` is randomly selected (line 366) and then used to create the transformed partitioning (lines 369 - 373).

A random seed can be set through the command line in order to ensure repeatability of the experiments.

4.8 No new nonterminals

In this strategy, $p \in pos(\pi)$ is chosen, such that it does not require a new nonterminal if such a p exists. The idea behind this strategy is that p could be chosen in a way that minimizes the number of nonterminals and/or rules of the resulting grammar. However, since the number of nonterminals/rules also depends on the recursive processing of the children and the processing of the other trees in the corpus, one needs to use a simplified approach.

This strategy is implemented in the function `fanout_limited_partitioning_no_new_nont()` (lines 169 - 213). It takes four extra arguments besides `part` and `fanout`. These are the current tree in the corpus (`tree`), for which the recursive partitioning is to be transformed, the previously generated nonterminals of the hybrid grammar (`nonts`), the nonterminal labelling strategy currently being used (`nont_labelling`), and the strategy to be used as fallback (`fallback`).

It performs a left-to-right breadth-first search on `part`. If it finds a subtree that fulfills the target condition (line 189), it also checks, whether `subroot` requires a new nonterminal (lines 190 - 198). If so, the function continues to search for a subtree which does not require a new nonterminal. If not, the subtree is used to create the transformed recursive partitioning (lines 202 - 205).

It is possible (and in the case of the first tree in the corpus inevitable) that all positions in `part` require a new nonterminal. In that case, the breadth-first search will not return a subtree of `part` to use for the transformation. Therefore the function then calls a fallback function (depending on the `fallback` parameter). These functions (`fallback_rtl()`, `fallback_ltr()`, `fallback_argmax()`, `fallback_random()`) (lines 215 - 339) each work like their counterpart for the respective strategy, except that they call `fanout_limited_partitioning_no_new_nont()` for the recursive transformation of the subtrees instead of themselves.

5 Experiments

In this section, I show experimental results for the training of hybrid grammars with the various transformation strategies.

I ran most of the experiments on a computer with a 2.5 GHz Intel Core i5-7200U CPU and 8GB of RAM. For some experiments, the RAM was not sufficient. I ran those on a server with 32 GB RAM (capped at 15 GB) and two Dual-Core AMD Opteron(tm) Processor 2224 SE CPUs, which had a clock rate of 1 GHz and 3.2 GHz, respectively. Therefore, the parse times for these experiments cannot be compared to the other results. They are marked in yellow in the table for parse times. For some experiments even the higher RAM was not sufficient, therefore, the table entries for these read "insuff. RAM".

For the no-new-nonterminals strategy, I list the results for different fallback strategies in separate columns.

For the random-choice strategy (and no-new-nonterminals with random choice as fallback) I ran three experiments with random seeds 1, 10, and 20. The results given here are the arithmetic mean of these experiments.

As terminal labeling strategy, I used POS for all experiments.

The first three columns in every table show the used nonterminal labeling strategy (strict or child and POS, DEPREL or POS+DEPREL) and the maximal fanout.

I use the same parsers as Gebhardt et al. [5]. For LCFRS parsing with maximal fanout one this is the OpenFST framework ([1]) with python bindings by Gorman [6]. For LCFRS parsing with a higher maximal fanout this is an LCFRS parser by Angelov and Ljunglöf [2], which is part of the Grammatical Framework ([8]).

Since Gebhardt et al. [5] did not find any grammar to have a fanout higher than four in their experiments, I did not include transformations for maximal fanout four or higher in my experiments.

In each table, the best entry is marked in green and in italics and the worst entry in red and bold. For the tables for the parse times, the marked results are the best and worst results not computed on the server.

In the following sections I will be using abbreviations for the strategies.

- `rtl` for right-to-left breadth-first search
- `ltr` for left-to-right breadth-first search
- `argmax` for choosing p such that $p = argmax_{p \in pos(\pi)} |\pi(p)|$
- `random` for choosing p randomly
- `nnont(fallback)` for choosing p sucht that no new nonterminal is created if possible. `fallback` stands for the abbreviation of the respective fallback strategy

5.1 Criteria

In this section, I will give an overview over the criteria according to which I compared the transformation strategies.

The number of nonterminals and rules in the induced grammar influences the parsing complexity. Therefore, a smaller number of nonterminals and rules is better.

The number of derivation trees for a sentence in the corpus indicates the ambiguity of the grammar. If the number is high, there are more ways to syntactically interpret the sentence. This can make semantic analysis of a sentence harder. Therefore, smaller numbers are better as well.

The *unlabeled attachment score* is the percentage of words for which the string parser found the correct HEAD, meaning the parent of that word in a dependency structure. The *labeled attachment score* also takes into account whether the parser found the correct label for that word (according to the terminal labeling strategy). For attachment scores, higher numbers are better since they mean that more words were correctly placed in relation to the grammatical structure of the sentence.

The *parse time* here is the time that was necessary for the string parser to parse the corpus and calculate the attachment scores.

5.2 Corpora

I am using corpora in the conll format from the SPMRL Shared Task [9]. Specifically, I use the German and the Polish Predicted training corpora for training of the grammars and counting of the number of derivation trees per sentence. For calculating the attachment scores and measuring the parse times, I used the development corpora. I shortened the German development corpus to the length of the Polish corpus in order to make parse times comparable. This is done by the script `smallCorpus.py`.

5.3 Number of nonterminals and rules

The results for `random` and `nnont(random)` were rounded to whole numbers.

Table 2: Number of nonterminals - Polish

		fanout	`rtl`	`ltr`	`argmax`	`nnont(rtl)`	`nnont(ltr)`	`nnont(argmax)`	`nnont(random)`	`random`
child	pos	1	539	530	844	591	586	591	2985	2929
		2	538	528	1339	591	589	591	6061	5966
		3	538	528	1429	589	586	588	7767	7661
	deprel	1	284	272	445	310	328	330	2001	1980
		2	281	*268*	695	331	305	306	4802	4778
		3	281	*268*	768	330	335	311	6482	6453
	pos+deprel	1	1622	1623	2196	1902	1931	1916	5669	5399
		2	1621	1624	3063	1916	1912	1906	9491	9064
		3	1621	1624	3158	1934	1941	1917	10992	10519
strict	pos	1	3670	3744	3989	5129	5127	5124	7846	6969
		2	3669	3741	4819	5119	5129	5127	11482	10357
		3	3669	3741	4880	5129	5111	5127	12923	11729
	deprel	1	2209	2290	2476	3241	3184	3268	5382	4908
		2	2206	2286	3047	3187	3192	3189	9009	8407
		3	2206	2286	3119	3186	3251	3253	10585	9940
	pos+deprel	1	5868	6041	6377	8556	8508	8500	11868	10276
		2	5866	6040	7462	8606	8533	8621	15903	13841
		3	5866	6040	7511	8471	8544	8664	**17111**	14936

Table 3: Number of nonterminals - German

		fanout	`rtl`	`ltr`	`argmax`	`nnont(rtl)`	`nnont(ltr)`	`nnont(argmax)`	`nnont(random)`	`random`
child	pos	1	1701	1598	1898	1135	1134	1139	7039	6660
		2	1016	1078	2500	1140	1139	1140	13567	13315
		3	978	1050	2854	1139	1138	1142	17660	17394
	deprel	1	1240	1104	1456	755	755	749	5752	5502
		2	715	687	1997	748	752	753	11806	11659
		3	682	*656*	2245	737	739	753	15781	15642
	pos+deprel	1	3730	3534	4762	3282	3280	3286	12047	11007
		2	2873	2941	5947	3294	3274	3286	19658	18596
		3	2830	2903	6230	3285	3288	3277	23114	21966
strict	pos	1	9090	8365	8660	9700	9694	9611	18213	15597
		2	8082	7593	8740	9640	9696	9617	24871	22011
		3	8000	7496	8768	9606	9652	9615	27958	24951
	deprel	1	6275	5779	6143	6853	6891	6997	13972	12302
		2	5503	5200	6637	7122	6866	6871	20578	18774
		3	5439	5142	6722	6977	6921	6928	23856	21948
	pos+deprel	1	12749	11982	12964	15075	15050	15148	24725	14257
		2	11703	11232	13309	15253	15119	14995	31806	27139
		3	11620	11134	13216	15150	15212	15214	**34293**	29424

In general, the strategies `rtl` and `ltr` produce the lowest number of nonterminals. For the German corpus, `ltr` is often slightly better than `rtl`, especially for strict labeling. The strategies `random` and `nnont(random)` lead in all but one case to the highest numbers. The reason for this is that with `rtl` and `ltr`, the recursive partitionings of similar hybrid trees are transformed in a similar way, whereas with `random` they can result in very different transformed partitionings which in turn lead to more different nonterminals.

The `nnont` strategies lead to higher numbers of nonterminals than the corresponding fallback strategies, with the exception of child labeling with fanout one for fallback `rtl`, `ltr` and with any fanout for fallback `argmax`. In these cases the numbers for `nnont` are between 7% and 66% lower than for the fallback strategies.For the German corpus, the strategies `nnont(rtl)`, `nnont(ltr)` and `nnont(argmax)` lead to the overall lowest numbers for child labeling with maximal fanout one. For `nnont(random)`, the numbers with child labeling are also better than with strict labeling, although they are higher than the numbers for `random` in all but one case. The reason for these observations is presumably that with child labeling, different nodes in recursive partitionings are assigned the same label. This makes it more likely that a node results in a nonterminal that already exists.

For the Polish corpus, the different maximal fanouts do not lead to different results for the strategies `ltr`, `rtl` and their corresponding `nnont` strategies. For `argmax`, `random` and `nnont(random)` a higher maximal fanout leads to more nonterminals. This is due to the fact that, with a higher fanout, there are more nodes in a recursive partitioning that fulfill the target condition and could be randomly chosen or have the longest label. Therefore, there are more possibilities to transform recursive partitionings, which leads to more nonterminals. Interestingly, for `nnont(argmax)` the maximal fanout has no influence on the number of nonterminals. For the German corpus, the results are similar, except that the number of nonterminals is lower for fanouts two and three than for fanout one for the strategies `rtl` and `ltr`.

The German corpus leads to a higher number of nonterminals than the Polish corpus, depending on the strategy and the hyperparameters between 39% and 337% higher.

Similar observations can be made by considering the number of rules.

Table 4: Number of rules - Polish

		fanout	`rtl`	`ltr`	`argmax`	`nnont (rtl)`	`nnont (ltr)`	`nnont(argmax)`	`nnont(random)`	`random`
child	pos	1	2981	2912	3647	2983	2982	2983	10366	9069
		2	2983	2905	5134	2983	2984	2983	14696	12918
		3	2983	2905	5237	2982	2982	2982	15580	13740
	deprel	1	1949	1885	2491	1936	1958	1962	8105	7321
		2	1950	1879	3557	1960	*1927*	1932	12633	11603
		3	1950	1879	3685	1962	1963	1939	13716	12644
	pos+deprel	1	5171	5083	5830	5376	5411	5383	14593	11750
		2	5171	5078	7548	5388	5386	5383	18497	14901
		3	5171	5078	7626	5404	5416	5397	19105	15422
strict	pos	1	7093	7127	7481	8819	8820	8817	15691	12655
		2	7095	7121	8593	8814	8820	8820	19818	16023
		3	7095	7121	8654	8820	8801	8820	20530	16664
	deprel	1	4938	4971	5306	6258	6211	6282	12406	10439
		2	4939	4967	6247	6207	6216	6217	16801	14335
		3	4939	4967	6341	6213	6261	6266	17691	15169
	pos+deprel	1	9620	9781	10167	12164	12131	12118	20007	15446
		2	9620	9776	11384	12222	12151	12231	23839	18217
		3	9620	9776	11434	12102	12152	12266	**24359**	18638

Table 5: Number of rules - German

		fanout	rtl	ltr	argmax	nnont(rtl)	nnont(ltr)	nnont(argmax)	nnont(random)	random
child	pos	1	6243	5940	7129	5546	5546	5551	20434	17260
		2	5354	5434	9238	5551	5553	5552	29021	25287
		3	5302	5392	9554	5551	5553	5554	26516	27063
	deprel	1	5069	4694	5915	4296	4296	4292	17998	15521
		2	4278	4185	7495	4291	4296	4295	26511	23672
		3	4219	*4130*	7859	4275	4277	4296	28640	25722
	pos+deprel	1	9902	9610	11340	9634	9630	9634	27930	21731
		2	9048	9176	13536	9636	9625	9629	35468	28307
		3	8988	9136	13680	9632	9635	9633	36703	29405
strict	pos	1	14887	14206	14717	15951	15952	15863	31240	24521
		2	14006	13551	15268	15900	15949	15868	38670	31030
		3	13949	13467	15353	15864	15913	15867	40153	32393
	deprel	1	11044	10443	11126	11909	11944	12030	26106	21127
		2	10257	9856	12084	12154	11920	11926	33719	28072
		3	10194	9790	12217	12012	11966	11977	35427	29699
	pos+deprel	1	18378	17717	18823	20666	20632	20767	37572	28271
		2	17581	17172	19629	20846	20723	20620	44379	33786
		3	17526	17093	19610	20772	20810	20840	**45356**	34628

5.4 Number of derivation trees

For `random` and `nnont(random)` both the average and the maximal number of derivation trees varied widely for the different random seeds. I, therefore, include both the arithmetic mean and the median for the results of these strategies.

5.4.1 Average number of derivation trees

Most of the numbers in these tables were rounded to two decimal places. In the polish table, I did not round the numbers in the columns `ltr` and `rtl`, because for some of these their difference to one would otherwise not have been visible.

5.4.2 Maximal number of derivation trees

The results for `random` and `nnont(random)` were rounded to whole numbers, since they are the average of three different runs.

Table 6: Average number of derivation trees - Polish

		fanout	`rtl`	`ltr`	`argmax`	nnont (`rtl`)	nnont (`ltr`)	`nnont(argmax)`	`nnont(random) mean`	`nnont(random) median`	`random mean`	`random median`
child	pos	1	1.0005	1.0155	1.24	1.02	1.03	1.02	92443.13	119773.41	9210.38	8396.92
		2	*1*	*1*	2.05	1.02	1.02	1.02	87414.26	28476.51	2493.47	2789.04
		3	*1*	*1*	2.03	1.15	1.06	1.03	11324.42	3779.00	241.69	216.86
	deprel	1	1.0440	1.0095	1.36	1.17	1.06	1.08	1194461.30	**1222231.00**	736366.44	856458.71
		2	*1*	*1*	2.92	1.05	1.09	1.27	901035.75	301502.78	128481.06	42388.17
		3	*1*	*1*	3.00	1.20	1.04	1.10	92674.88	11937.39	7933.31	4100.99
	pos+deprel	1	1.0005	1.0045	1.07	1.07	1.10	1.07	310.26	242.93	44.53	38.34
		2	*1*	*1*	1.50	1.09	1.07	1.20	72.81	79.20	14.04	15.06
		3	*1*	*1*	1.49	1.07	1.16	1.08	26.03	25.40	5.05	5.13
strict	pos	1	*1*	1.0005	1.13	1.65	1.65	1.88	7499.63	6698.63	1209.11	1044.75
		2	*1*	*1*	1.72	1.67	1.65	1.65	1744.10	1679.41	301.60	244.22
		3	*1*	*1*	1.75	1.65	1.66	1.88	475.60	442.71	106.74	45.59
	deprel	1	*1*	1.0005	1.28	1.66	1.82	1.62	131530.02	123501.14	48591.54	42059.12
		2	*1*	*1*	2.52	1.73	1.92	2.04	40766.86	44183.18	5835.10	2030.89
		3	*1*	*1*	2.53	1.84	1.63	1.62	4100.20	2406.26	659.80	647.52
	pos+deprel	1	*1*	*1*	1.07	1.40	1.55	1.43	118.53	123.24	13.70	13.40
		2	*1*	*1*	1.39	1.37	1.45	1.32	38.28	40.66	6.10	5.94
		3	*1*	*1*	1.40	1.44	1.39	1.32	17.93	18.65	3.07	3.15

Table 7: Average number of derivation trees - German

		fanout	rtl	ltr	argmax	nnont(rtl)	nnont(ltr)	nnont(argmax)	nnont(random) mean	nnont(random) median	random mean	random median
child	pos	1	1.58	2.61	3.41	2.48	1.84	1.84	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
		2	1.03	1.19	7.43	1.84	1.84	1.84	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
		3	1.02	1.004	5.81	1.84	1.84	1.84	insuff. RAM	insuff. RAM	337260.28	505890.41
	deprel	1	2.22	3.84	6.53	1.50	1.50	1.53	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
		2	1.10	1.42	15.84	1.50	1.55	1.51	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
		3	1.03	1.01	9.94	1.50	1.50	1.55	**661724834.40**	521308190.84	123475.56	149646.36
	pos+deprel	1	1.06	1.29	1.46	1.18	1.18	1.21	1088516.36	912811.31	127818.13	141051.60
		2	1.0015	1.05	2.44	1.16	1.20	1.22	152214.33	94285.33	2124.19	1927.85
		3	1.001	1.0005	2.17	1.23	1.22	1.18	11421.62	9648.00	81.48	82.51
strict	pos	1	1.04	1.11	2.01	2.45	2.59	2.03	9598344.87	7735129	544989807.40	1414107.06
		2	1.04	1.07	4.42	3.49	2.45	2.03	24788445.30	23738709.53	854248.55	990897.88
		3	1.04	*1*	4.26	2.99	3.48	2.13	8792592.15	4578138.24	18460.01	15446.19
	deprel	1	1.17	1.35	2.46	2.12	2.45	1.90	61513702.28	59895422.15	4332236.56	5200939.72
		2	1.03	1.02	7.31	1.95	2.12	2.12	35032.72	33916.61	470284.55	395051.03
		3	1.0045	*1*	6.6565	1.91	2.14	2.37	12201986.90	6921753.43	29561.55	23470.91
	pos+deprel	1	1.03	1.06	1.14	1.68	1.63	1.74	35032.72	33916.61	2195.17	2063.28
		2	*1*	1.0005	1.89	1.63	1.66	1.77	17107.59	21289.83	583.50	711.64
		3	*1*	*1*	1.90	1.59	1.58	1.82	3539.31	3410.90	29.73	29.94

Table 8: Maximal number of derivation trees - Polish

		fanout	`rtl`	`ltr`	`argmax`	nnont (`rtl`)	nnont (`ltr`)	`nnont(argmax)`	`nnont(random) mean`	`nnont(random) median`	`random mean`	`random median`
child	pos	1	2	2	12	4	4	4	171058222	222038544	10234606	8941408
		2	*1*	*1*	28	4	4	4	160516985	40486188	2866505	3416906
		3	*1*	*1*	28	4	4	4	20838414	5342112	267438	224224
	deprel	1	7	2	16	4	8	16	2328278372	**2385151572**	1439972013	1678197470
		2	*1*	*1*	52	8	2	8	1697133824	526215906	220724418	44028390
		3	*1*	*1*	56	16	8	4	173200828	6674920	12545566	3433662
	pos+deprel	1	2	2	4	6	8	8	386199	270764	25206	18102
		2	*1*	*1*	10	8	4	8	28854	26460	3233	3116
		3	*1*	*1*	11	8	8	8	6011	5424	413	429
strict	pos	1	*1*	2	8	16	16	16	10347847	7220850	1125529	1111844
		2	*1*	*1*	16	16	16	16	1453235	1593900	294937	166980
		3	*1*	*1*	16	16	16	16	408524	438834	154408	23132
	deprel	1	*1*	2	16	16	16	16	240316324	222886899	87601659	76145664
		2	*1*	*1*	32	16	32	32	65975673	73604075	6708850	1201060
		3	*1*	*1*	36	16	16	16	5458668	1111831	839704	881460
	pos+deprel	1	*1*	*1*	4	16	16	16	105806	108810	1901	1958
		2	*1*	*1*	8	8	16	16	12335	10500	675	606
		3	*1*	*1*	8	16	16	8	5652	4392	294	260

Table 9: Maximal number of derivation trees - German

		fanout	rtl	ltr	argmax	nnont(rtl)	nnont(ltr)	nnont(argmax)	nnont(random) mean	nnont(random) median	random mean	random median
child	pos	1	60	88	364	64	64	64	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
		2	2	12	336	64	64	64	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
		3	2	2	576	64	64	64	insuff. RAM	insuff. RAM	391334373	587001560
	deprel	1	138	133	496	21	21	21	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
		2	8	31	1272	1.5035	21	21	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
		3	4	2	944	21	21	21	**894353667432**	495697164512	251099920	155930884
	pos+deprel	1	10	16	70	24	8	16	835987851	584755200	168341255	145207404
		2	2	4	76	6	16	16	136920522	71933616	1758899	2198811
		3	2	2	32	24	24	24	9345373	5895570	45737	55722
strict	pos	1	4	4	72	64	80	32	8718654577	6308229760	808025672	900256415
		2	4	4	192	64	64	32	16467070070	14426310989	1055447456	1009847668
		3	4	1	384	96	72	32	14698374308	5362603124	26992833	18115469
	deprel	1	12	8	90	16	32	16	90950934602	73553514720	7040325268	9236731212
		2	8	4	288	16	16	16	insuff. RAM	insuff. RAM	246641067	231049692
		3	4	*1*	384	16	32	64	22541048194	11330051214	41410861	19972632
	pos+deprel	1	4	3	8	32	24	48	19173006	19461906	2170479	2233374
		2	*1*	2	18	24	24	48	9695834	11618061	677683	589056
		3	*1*	*1*	16	24	24	32	2468826	1806528	8403	6696

The strategies `rtl` and `ltr` lead to the grammars with the lowest ambiguity. For the Polish corpus, the average number of derivation trees is only in five cases for `ltr` and three cases for `rtl` higher than one. All of these eight cases are for maximal fanout one. The strategies `random` and `nnont(random)` lead to the highest numbers of parse trees. For these two strategies, there are large differences between the labeling strategies with POS+DEPREL leading to results that are orders of magnitude lower than the results for either POS or DEPREL labeling.

The average number of derivation trees for the `nnont` strategies is often higher than that of the corresponding fallback strategies. Exceptions from this are the `argmax` and `nnont(argmax)` strategies where `nnont(argmax)` results in a lower average number in most cases (in 14 out of 18 cases for the Polish corpus and in 16 out of 18 cases for the German corpus), especially with child labeling.

For `argmax`, a maximal fanout higher than one leads in most cases to more derivation trees (for the German corpus, fanout two always leads to a higher number than fanout three). For `nnont(argmax)`, `ltr`, `rtl`, and their respective `nnont` strategies, the maximal fanout has no influence on the average number of derivation trees, for the Polish corpus. For the German corpus, a higher fanout leads to a higher ambiguity for `ltr` and `rtl`. For `random` and `nnont(random)` a higher maximal fanout leads to a lower average number of derivation trees.

The numbers for the German corpus are in all but four cases higher than the numbers for the Polish corpus.

Similar observations can be made for the maximum number of derivation trees, except that `argmax` and `nnont(argmax)` do not show as much of a difference with strict labeling.

For the Polish corpus, almost all the maximal numbers of derivation trees for `nnont` with fallback strategies `rtl`, `ltr` and `argmax` are powers of two, with only one exception.

5.5 Labelled attachment score

The results for `random` and `nnont(random)` were rounded to two decimal places, since the individual results for the different random seeds had the same accuracy.

Since the observations for the Labelled Attachment Score and the Unlabelled Attachment Score are similar with and without punctuation, I put the tables for the LAS with punctuation and all tables for the UAS in Appendix appendix A. The difference is that the UAS is higher than the LAS.

Table 10: Labelled Attachment Score (no punctuation) - Polish

		fanout	`rtl`	`ltr`	`argmax`	`nnont(rtl)`	`nnont(ltr)`	`nnont(argmax)`	`nnont(random)`	`random`
child	pos	1	44.19	45.13	44.59	44.59	44.58	44.58	45.45	41.91
		2	44.08	44.72	41.84	44.69	44.68	44.69	45.26	36.78
		3	44.08	44.72	41.89	44.69	44.70	44.73	45.36	33.33
	deprel	1	58.05	57.81	56.73	58.01	57.84	57.81	54.21	51.72
		2	57.95	58.29	56.71	58.35	58.60	58.56	52.50	49.31
		3	57.95	58.29	56.60	58.40	58.36	58.57	52.40	48.70
	pos+deprel	1	56.12	57.91	55.14	56.80	56.79	56.86	*61.02*	47.84
		2	56.18	57.79	45.61	57.23	57.21	57.21	60.53	32.80
		3	56.18	57.79	44.90	56.98	57.16	57.10	59.86	26.69
strict	pos	1	34.50	35.39	35.80	31.47	31.47	31.50	37.34	31.92
		2	34.61	35.52	34.63	34.26	34.24	34.24	36.75	22.92
		3	34.61	35.52	34.54	34.23	34.27	34.27	36.72	19.38
	deprel	1	59.27	59.07	57.82	58.43	58.42	58.50	56.96	54.13
		2	59.23	59.41	57.20	58.75	58.73	58.63	57.05	52.63
		3	59.23	59.41	57.16	58.69	58.76	58.75	57.37	51.25
	pos+deprel	1	34.26	34.57	34.54	26.02	25.89	25.95	37.68	25.34
		2	34.34	34.55	31.27	31.64	31.54	31.33	35.37	17.19
		3	34.34	34.55	31.29	32.18	31.47	31.40	35.37	**14.28**

`random` almost always results in the lowest labeled attachment score, however, `nnont(random)` results in much higher labeled attachment scores and the highest of all strategies with POS and POS+DEPREL labeling.

The other results are close together with the exception of the `nnont` strategies with fallback `rtl`, `ltr` and `argmax` for strict/POS and strict/POS+DEPREL labeling. For these combinations, the values for fanout one are lower than the corresponding values of the fallback strategies and also lower than the values for maximal fanouts two and three. `nnont(argmax)` results in higher values than `argmax` for child/POS and child/POS+DEPREL labeling and maximal fanouts two and three.

For `random` a higher maximal fanout leads to a lower score. `random` with strict/POS+DEPREL labeling and maximal fanout three leads to the lowest score overall.

Both corpora lead to these observations, however, they sometimes differ in their scores depending on the labeling strategy.

Table 11: Labelled Attachment Score (no punctuation) - German

		fanout	rtl	ltr	argmax	nnont(rtl)	nnont(ltr)	nnont(argmax)	nnont(random)	random
child	pos	1	55.83	55.55	54.55	54.07	54.00	54.09	56.82	52.43
		2	55.36	55.40	51.54	55.42	55.42	55.44	57.14	42.50
		3	55.12	55.43	50.69	55.46	55.45	55.47	57.20	32.51
	deprel	1	68.03	67.35	67.74	67.24	67.20	67.15	65.20	63.08
		2	insuff. RAM	69.42	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
		3	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
	pos+deprel	1	59.52	57.90	54.04	55.31	55.25	55.33	62.51	44.95
		2	59.82	59.00	44.22	59.17	59.16	59.20	63.31	27.58
		3	59.85	58.74	39.33	59.20	59.19	59.23	62.86	18.83
strict	pos	1	31.00	32.31	34.65	25.31	25.23	25.20	36.61	27.62
		2	32.43	34.37	36.68	33.91	33.96	33.96	39.15	18.85
		3	32.26	34.37	37.67	33.95	33.98	33.96	38.24	15.35
	deprel	1	70.08	70.91	70.58	68.13	68.03	68.12	69.39	66.22
		2	70.90	72.22	70.86	*72.25*	72.03	72.13	70.00	62.74
		3	70.80	*72.25*	70.89	72.19	72.13	72.23	insuff. RAM	insuff. RAM
	pos+deprel	1	23.73	24.33	24.84	16.85	16.70	16.99	25.77	17.39
		2	25.68	26.03	25.53	24.96	24.95	24.90	27.70	11.20
		3	25.61	26.05	26.35	24.81	24.93	24.71	27.40	**8.37**

5.6 Parse time

The values in these tables are in seconds and all numbers are rounded to two decimal places.

Table 12: Parse time in seconds - Polish

		fanout	`rtl`	`ltr`	`argmax`	nnont(`rtl`)	nnont(`ltr`)	`nnont(argmax)`	`nnont(random)`	`random`
child	pos	1	1.49	1.76	1.49	1.70	1.70	1.69	10.45	6.09
		2	12.58	4.65	23.07	4.95	4.89	4.81	122.43	117.34
		3	12.63	4.63	24.85	4.88	4.89	4.90	118.25	114.92
	deprel	1	32.61	30.75	42.77	30.21	30.49	30.53	305.82	258.23
		2	23.92	15.30	95.17	16.06	15.44	15.62	617.71	511.81
		3	23.93	15.31	102.87	15.84	15.64	15.61	630.02	**643.21**
	pos+deprel	1	1.30	1.43	*1.21*	1.40	1.42	1.40	4.72	2.19
		2	12.17	6.21	16.08	8.31	7.50	7.57	38.17	18.61
		3	12.12	6.19	13.29	7.51	7.56	7.54	32.48	15.11
strict	pos	1	1.85	1.88	1.78	2.13	2.13	2.14	4.01	2.88
		2	27.23	8.89	24.19	15.20	15.31	15.30	85.35	77.58
		3	27.12	8.90	25.71	15.25	15.31	14.84	86.36	77.27
	deprel	1	10.78	9.93	7.64	10.85	9.50	9.70	67.76	50.58
		2	16.88	9.33	22.45	13.94	13.91	14.10	125.33	112.76
		3	16.98	9.26	23.56	13.85	14.17	14.04	115.93	110.17
	pos+deprel	1	2.25	2.22	2.16	2.33	2.40	2.39	3.30	2.47
		2	18.52	8.95	15.55	12.28	12.28	12.07	26.45	9.76
		3	18.59	8.93	15.56	12.35	12.12	11.94	24.63	9.51

`ltr` leads to the shortest parse times for maximal fanouts two and three. For maximal fanout one, the results for the strategies are close together, except for `random` and `nnont(random)`, which lead to longer parse times. `nnont(random)` results always in the longest parse time. `random` usually leads to parse times only slightly shorter than `nnont(random)`. The exception from this is strict/-POS+DEPREL labeling where `random` leads to parse times that are almost as short as the times for `ltr` (up to 43% longer compared to up to 4003% longer for other labeling strategies).

`nnont(rtl)` and `nnont(argmax)` result in shorter parser than times `rtl` and `argmax` for maximal fanouts two and three. `nnont(ltr)` results in longer parse times than `ltr` for child labeling with maximal fanouts two and three.

The parse times for fanout two and three are longer than for fanout one,

Table 13: Parse time in seconds - German

		fanout	`rtl`	`ltr`	`argmax`	`nnont(rtl)`	`nnont(ltr)`	`nnont(argmax)`	`nnont(random)`	`random`
child	pos	1	12.53	6.64	7.95	6.59	6.58	6.59	99.24	47.27
		2	240.46	35.11	300.59	42.50	40.79	40.67	5278.70	1440.56
		3	244.75	31.91	390.04	40.51	40.39	41.27	3973.60	883.09
	deprel	1	802.95	567.39	762.77	505.93	507.19	506.80	**3378.67**	2808.61
		2	insuff. RAM	6217.51	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
		3	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
	pos+deprel	1	4.62	3.35	*2.71*	3.33	3.35	3.33	19.92	7.11
		2	79.30	22.23	64.80	28.88	27.94	27.61	219.92	75.28
		3	73.48	22.11	69.03	27.66	30.24	27.66	171.27	42.48
strict	pos	1	9.36	5.56	5.83	6.94	6.97	6.98	14.53	9.43
		2	137.98	30.17	108.20	51.45	48.90	49.05	481.84	287.85
		3	133.04	28.73	133.16	50.04	49.07	50.26	350.91	212.78
	deprel	1	145.15	103.98	95.96	82.87	82.93	83.83	692.53	445.21
		2	946.07	227.22	756.79	319.90	324.33	325.37	9553.08	6216.61
		3	3406.73	266.82	4368.52	302.27	326.20	331.04	insuff. RAM	insuff. RAM
	pos+deprel	1	6.09	5.88	6.14	6.26	6.25	6.27	9.03	5.26
		2	76.13	21.03	59.01	30.83	30.73	30.43	105.26	25.43
		3	74.92	20.48	68.65	30.06	31.00	30.61	88.87	29.41

wich is at least partially due to the different parsers. The only exceptions from this occur for DEPREL labeling in the Polish corpus. This is unclear for the German corpus since for child/DEPREL labeling and fanouts two and three, the RAM was insufficient in all but one case.

Induction using the German corpus leads to longer parse times than using the Polish corpus.

5.7 Conclusions

Considering all criteria, `ltr` produces the overall best results. While `nnont(random)` often leads to a higher LAS, it also leads to considerably longer parse times and a higher ambiguity. `rtl` leads to results similar to the ones of `ltr` with the exception of parse time for maximal fanouts two and three. `random` leads generally to worse results than most other strategies. The `nnont` strategies only minimize the number of nonterminals under specific circumstances and seem to work better for German than for Polish.

Some further questions that remain are

- Is there a better strategy for minimizing the number of nonterminals?
- How do the various transformation strategies compare to directly extracted recursive partitionings?
- Since both Polish and German have a relatively free word order, it would be interesting to see to what results the transformation strategies lead in languages with strict word order, such as English.

Acknowledgement

I would like to thank Markus Teichmann for guiding me through this process, proofreading my drafts, and giving me advice.

References

[1] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. Openfst: A general and efficient weighted finite-state transducer library. In *Proceedings of the 12th International Conference on Implementation and Application of Automata*, CIAA'07, pages 11–23, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-76335-X, 978-3-540-76335-2. URL http://dl.acm.org/citation.cfm?id=1775283.1775287.

[2] Krasimir Angelov and Peter Ljunglöf. Fast statistical parsing with parallel multiple context-free grammars. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 368–376. Association for Computational Linguistics, Gothenburg, Sweden, 2014.

[3] Pierre Deransart and Jan Małuszynski. Relating logic programs and attribute grammars. *The Journal of Logic Programming*, 2:119–155, July 1985.

[4] Kilian Gebhardt and Markus Teichmann. Install prerequesites. URL https://gitlab.tcs.inf.tu-dresden.de/hybrid-grammars/lcfrs-sdcp-hybrid-grammars/wikis/install-prerequesites.

[5] Kilian Gebhardt, Mark-Jan Nederhof, and Heiko Vogler. Hybrid grammars for parsing of discontinuous phrase structures and non-projective dependency structures. *Computational Linguistics, accepted for publication*, 2017.

[6] Kyle Gorman. Pynini: A python library for weighted finite-state grammar compilation. In *Proceedings of the ACL Workshop on Statistical NLP and Weighted Automata*, pages 75–80, 2016.

[7] Mark-Jan Nederhof and Heiko Vogler. Hybrid grammars for discontinuous parsing. 2014. doi: http://www.aclweb.org/anthology/C14-1130.

[8] Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford, 2011. ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).

[9] Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. Introducing the spmrl 2014 shared task on parsing morphologically-rich languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 103–109, Dublin, Ireland, August 2014. Dublin City University. URL http://www.aclweb.org/anthology/W14-6111.

[10] K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Meeting on Association for Computational Linguistics*, ACL '87, pages 104–111, Stroudsburg, PA, USA, 1987. Association for Computational Linguistics. doi: 10.3115/981175.981190. URL http://dx.doi.org/10.3115/981175.981190.

A Attachment Scores

Table 14: Labelled Attachment Score (punctuation) - Polish

		fanout	rtl	ltr	argmax	mont(rtl)	mont(ltr)	mont(argmax)	mont(random)	random
child	pos	1	47.07	47.95	47.24	47.45	47.45	47.45	47.92	44.47
		2	46.87	47.58	44.38	47.57	47.57	47.57	48.06	39.34
		3	46.87	47.58	44.50	47.55	47.60	47.55	48.19	35.75
	deprel	1	60.28	60.13	59.04	60.39	60.22	60.19	56.77	54.21
		2	60.28	60.68	59.28	60.79	60.99	61.05	55.14	52.07
		3	60.28	60.68	59.20	60.83	61.02	60.79	55.29	51.61
	pos+deprel	1	57.30	58.96	56.17	57.91	57.87	57.95	62.26	49.24
		2	57.34	58.88	46.62	58.26	58.24	58.24	*61.74*	34.15
		3	57.34	58.88	45.86	58.03	58.12	58.17	61.10	28.08
strict	pos	1	36.60	37.59	37.72	33.24	33.24	33.26	39.35	33.89
		2	36.73	37.71	36.23	36.36	36.34	36.34	38.82	24.63
		3	36.73	37.71	36.14	36.33	36.36	36.38	38.85	21.05
	deprel	1	61.45	61.02	59.72	60.06	60.07	60.12	59.19	56.52
		2	61.38	*61.47*	59.33	60.81	60.73	60.80	59.4	55.27
		3	61.38	*61.47*	59.28	60.74	60.82	60.81	59.64	53.58
	pos+deprel	1	35.12	35.47	35.37	26.57	26.44	26.49	38.62	26.40
		2	35.19	35.47	32.06	32.39	32.10	32.31	36.32	18.09
		3	35.19	35.47	32.04	32.93	32.16	32.22	36.33	**15.22**

Table 15: Unlabelled Attachment Score (punctuation) - Polish

		fanout	rtl	ltr	argmax	mont(rtl)	mont(ltr)	mont(argmax)	mont(random)	random
child	pos	1	75.81	76.53	75.08	75.96	76.02	75.97	75.39	70.69
		2	75.83	*76.27*	72.88	76.21	*76.27*	76.21	75.91	66.73
		3	75.83	*76.27*	73.06	76.19	76.19	76.18	76.19	63.37
	deprel	1	74.15	74.09	72.83	74.30	74.19	74.14	70.49	68.54
		2	74.14	74.54	72.57	74.66	74.90	74.80	69.53	66.50
		3	74.14	74.54	72.61	74.71	74.72	74.89	69.61	66.51
	pos+deprel	1	73.07	74.06	72.00	73.51	73.47	73.54	76.10	67.45
		2	73.10	74.02	64.96	73.82	73.86	73.86	76.08	57.48
		3	73.10	74.02	64.20	73.75	73.82	73.78	75.69	53.57
strict	pos	1	63.92	64.59	64.50	60.86	60.86	60.88	65.88	60.79
		2	64.11	64.57	63.42	64.15	64.08	64.08	65.30	53.90
		3	64.11	64.57	63.30	64.08	64.16	64.08	65.35	51.09
	deprel	1	73.50	72.87	71.78	71.84	71.86	71.95	71.36	69.46
		2	73.55	73.26	71.73	72.76	72.74	72.66	71.63	68.10
		3	73.55	73.26	71.65	72.70	72.76	72.79	71.78	66.09
	pos+deprel	1	57.25	57.77	57.52	51.15	51.03	51.07	59.27	51.68
		2	57.34	57.85	55.17	55.36	55.32	55.18	58.31	46.35
		3	57.34	57.85	55.18	55.70	55.17	55.19	58.18	**44.58**

Table 16: Unlabelled Attachment Score (no punctuation) - Polish

		fanout	rtl	ltr	argmax	mont(rtl)	mont(ltr)	mont(argmax)	mont(random)	random
child	pos	1	77.73	*78.52*	77.09	77.84	77.90	77.84	77.58	72.47
		2	77.88	78.24	74.89	78.14	78.19	78.14	77.86	68.60
		3	77.88	78.24	75.00	78.12	78.14	78.09	78.07	65.31
	deprel	1	74.23	74.12	72.87	74.29	74.18	74.12	70.20	68.46
		2	74.11	74.47	72.20	74.53	74.76	74.66	69.28	66.22
		3	74.11	74.47	72.23	74.59	74.60	74.73	69.07	66.15
	pos+deprel	1	74.11	75.04	73.21	74.52	74.49	74.55	76.93	68.80
		2	74.13	74.96	66.52	74.87	74.91	74.91	76.97	59.60
		3	74.13	74.96	65.80	74.82	74.90	74.84	76.60	55.93
strict	pos	1	65.91	66.51	66.57	63.30	63.30	63.33	67.92	62.72
		2	66.10	66.45	65.83	66.21	66.14	66.14	67.25	56.41
		3	66.10	66.45	65.72	66.14	66.23	66.14	67.24	53.70
	deprel	1	73.35	72.91	71.92	72.13	72.13	72.26	71.16	69.25
		2	73.51	73.20	71.65	72.77	72.74	72.63	71.32	67.59
		3	73.51	73.20	71.62	72.73	72.79	72.79	71.51	65.80
	pos+deprel	1	59.37	59.87	59.58	53.93	53.82	53.88	61.12	54.08
		2	59.47	59.95	57.45	57.69	57.65	57.51	60.35	49.31
		3	59.47	59.95	57.50	58.01	57.50	57.52	60.17	**47.70**

Table 17: Unlabelled Attachment Score (no punctuation) - German

		fanout	`rtl`	`ltr`	`argmax`	`nnont(rtl)`	`nnont(ltr)`	`nnont(argmax)`	`nnont(random)`	`random`
child	pos	1	77.00	76.95	75.53	75.19	75.10	75.21	77.86	72.30
		2	77.09	76.98	72.11	76.89	76.90	76.89	78.27	60.99
		3	76.79	77.01	70.82	76.91	76.91	76.94	*78.54*	50.60
	deprel	1	75.19	74.52	74.96	74.39	74.33	74.28	72.34	63.08
		2	insuff. RAM	76.54	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
		3	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
	pos+deprel	1	68.02	66.68	63.37	64.26	64.27	64.29	70.66	44.95
		2	68.34	67.66	54.07	67.74	67.75	67.77	71.47	27.58
		3	68.45	67.44	49.99	67.78	67.78	67.80	71.00	18.83
strict	pos	1	49.75	51.36	53.51	43.38	43.29	43.27	55.61	46.04
		2	51.35	53.63	55.83	53.49	53.53	53.50	58.25	36.91
		3	51.23	53.58	56.75	53.49	53.55	53.53	57.44	33.14
	deprel	1	76.22	76.98	76.62	74.70	74.60	74.71	75.70	72.97
		2	77.11	78.19	77.07	78.17	77.90	78.00	76.21	69.63
		3	76.99	78.23	77.03	78.10	77.99	78.08	insuff. RAM	insuff. RAM
	pos+deprel	1	36.76	37.31	37.62	30.61	30.44	30.71	38.63	31.50
		2	38.38	38.82	38.20	37.85	37.80	37.78	40.23	25.96
		3	38.31	38.83	38.91	37.71	37.79	37.61	39.94	**23.57**

Table 18: Unlabelled Attachment Score (punctuation) - German

		fanout	`rtl`	`ltr`	`argmax`	`nnont(rtl)`	`nnont(ltr)`	`nnont(argmax)`	`nnont(random)`	`random`
child	pos	1	76.76	76.66	75.21	74.83	74.74	74.85	77.57	72.11
		2	76.86	76.75	71.56	76.63	76.63	76.63	78.03	61.09
		3	76.57	76.73	70.31	76.65	76.65	76.68	78.32	50.72
	deprel	1	75.04	74.59	75.13	74.56	74.50	74.45	72.70	71.19
		2	insuff. RAM	76.67	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
		3	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
	pos+deprel	1	68.13	66.62	63.17	64.28	64.28	64.31	70.52	55.57
		2	68.51	67.58	54.05	67.67	67.67	67.69	71.37	40.16
		3	68.61	67.37	50.11	67.70	67.70	67.72	70.87	32.91
strict	pos	1	49.74	51.30	53.47	43.24	43.14	43.12	55.47	46.15
		2	51.36	53.51	55.74	53.39	53.42	53.40	58.17	37.10
		3	51.25	53.46	56.63	53.39	53.44	53.42	57.33	33.30
	deprel	1	76.37	76.80	76.63	74.46	74.35	74.47	75.73	73.19
		2	77.34	78.13	77.16	78.15	77.88	77.99	76.28	69.99
		3	77.19	*78.19*	77.08	78.07	77.97	78.04	insuff. RAM	insuff. RAM
	pos+deprel	1	36.96	37.47	37.82	30.74	30.57	30.83	38.81	31.68
		2	38.57	38.98	38.37	37.97	37.92	37.89	40.36	26.17
		3	38.50	38.99	39.07	37.82	37.90	37.72	40.08	**23.78**

Table 19: Labelled Attachment Score (punctuation) - German

		fanout	rtl	ltr	argmax	nnont(rtl)	nnont(ltr)	nnont(argmax)	nnont(random)	random
child	pos	1	57.56	57.26	56.17	55.62	55.54	55.63	58.52	53.93
		2	57.17	57.17	52.76	57.13	57.13	57.15	58.91	43.75
		3	56.92	57.14	51.86	57.17	57.16	57.17	58.99	33.28
	deprel	1	68.59	68.12	68.61	68.11	68.06	68.01	66.25	64.19
		2	insuff. RAM	70.23	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
		3	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM	insuff. RAM
	pos+deprel	1	59.11	58.12	53.99	55.54	55.48	55.56	62.80	45.05
		2	60.32	59.25	44.11	59.42	59.40	59.44	63.67	27.46
		3	60.35	58.99	39.24	59.44	59.43	59.47	63.17	18.63
strict	pos	1	31.67	33.00	35.50	25.61	25.51	25.49	37.46	28.22
		2	33.24	35.14	37.62	34.71	34.75	34.75	40.17	19.23
		3	33.07	35.12	38.60	34.75	34.77	34.75	39.21	15.59
	deprel	1	70.81	71.30	71.17	68.51	68.40	68.50	70.02	67.07
		2	71.73	72.73	71.55	72.80	72.57	72.68	70.67	63.76
		3	71.60	*72.79*	71.53	72.73	72.67	72.75	insuff. RAM	insuff. RAM
	pos+deprel	1	23.54	24.11	24.65	16.57	16.43	16.70	25.59	17.25
		2	25.51	25.83	25.32	24.71	24.70	24.65	27.48	11.04
		3	25.42	25.85	26.15	24.56	24.68	24.46	27.17	**8.23**