Implementation of a Chomsky-Schützenberger n-Best Parser for Weighted Multiple Context-Free Grammars

Thomas Ruprecht and **Tobias Denkinger**
Faculty of Computer Science
Technische Universität Dresden
01062 Dresden, Germany
{thomas.ruprecht,tobias.denkinger}@tu-dresden.de

Abstract

Constituent parsing has been studied extensively in the last decades. Chomsky-Schützenberger parsing as an approach to constituent parsing has only been investigated theoretically, yet. It uses the decomposition of a language into a regular language, a homomorphism, and a bracket language to divide the parsing problem into simpler subproblems. We provide the first implementation of Chomsky-Schützenberger parsing. It employs multiple context-free grammars and incorporates many refinements to achieve feasibility. We compare its performance to state-of-the-art grammar-based parsers.

1 Introduction

The description of the syntax of natural languages (such as Danish, English, and German) with the help of formal grammars has been studied since Chomsky (1956). With a formal grammar, computers can calculate a syntactic representation (called *parse*) of a sentence in a natural language. Of the grammar classes in the Chomsky hierarchy (Chomsky, 1959), context-free grammars (short: CFGs) lack the expressive power necessary to model natural languages (Shieber, 1985) and parsing with context-sensitive grammars cannot be done efficiently (i.e. in polynomial time). This led to the introduction of a series of classes of *mildly context-sensitive grammars* (Joshi, 1985) that allow parsing in polynomial time but also capture an increasing amount of phenomena present in natural languages. Tree adjoining grammars (Joshi et al., 1975), linear context-free string-rewriting systems (short: LCFRSs, Vijay-Shanker et al., 1987), and multiple CFGs (short: MCFGs, Seki et al., 1991) are among those classes.

Chomsky-Schützenberger (short: CS) parsing was introduced by Hulden (2011) for CFGs and extended to MCFGs by Denkinger (2017). It uses a classical theorem by Chomsky and Schützenberger (1963, or the generalisation by Yoshinaka et al., 2010), which states that the language $\mathcal{L}(G)$ of a CFG (or an MCFG) G can be represented by a regular language R, a homomorphism h, and a Dyck language (resp. multiple Dyck language) D such that $\mathcal{L}(G) = h(R \cap D)$. The elements of $R \cap D$ correspond to parses in G. For a sentence w, a CS parser calculates the elements of $h^{-1}(w) \cap R \cap D$ and transforms them into parses. CS parsing can be viewed as a coarse-to-fine mechanism where R corresponds to the coarse grammar and $R \cap D$ to the fine grammar. The respective coarse-to-fine pipeline consists of (conceptually) simple operations such as h^{-1} or the intersection with R, which provides great flexibility. The flexibility is used to provide a fallback mechanism in case a finer stage of the pipeline rejects all proposals of a coarser stage. It also permits CS parsing in a broader setting than usual (for parsing) with minimal modification (see sec. 6).

We suspected that the coarse-to-fine view on CS parsing leads to an efficient implementation. Since initial tests revealed that the original algorithm for MCFGs (Denkinger, 2017, alg. 3, recalled in sec. 2) is not feasible in practice, we explore numerous optimisations (sec. 4), one of which is the use of a context-free approximation of the multiple Dyck language D. We introduce *component-wise derivations* (sec. 3) to relate this context-free approximation to D. Employing the optimisations, we provide the first implementation of a CS parser. In sec. 5, we compare our parser's performance to Grammatical Framework (Angelov and Ljunglöf, 2014), rparse (Kallmeyer and Maier, 2013), and disco-dop (van Cranenburgh et al., 2016). We restrict our comparison to (discontinuous) grammar-based parsers (excluding e.g. transition systems, Maier, 2015, Coavoux and Crabbé, 2017) since the principle of CS parsing requires a grammar.

2 Preliminaries

The sets of *non-negative integers* and *positive integers* are denoted by $\mathbb{N}$ and $\mathbb{N}_+$, respectively. We abbreviate $\{1, \dots, n\}$ by $[n]$ for each $n \in \mathbb{N}$.

Let A and B be sets. The *powerset of* A and the *set of (finite) strings over* A are denoted by $\mathcal{P}(A)$ and A^*, respectively. The *set of possibly infinite sequences of elements of* A is denoted by A^ω. A *partition of* A is a set $\mathfrak{P} \subseteq \mathcal{P}(A)$ whose elements (called *cells*) are non-empty, pairwise disjoint, and cover A (i.e. $\bigcup_{\mathfrak{p} \in \mathfrak{P}} \mathfrak{p} = A$). For each $a \in A$ and each equivalence relation $\approx$ on A, we denoted the *equivalence class of* a *w.r.t.* $\approx$ by $[a]_\approx$. The set of *functions from* A *to* B is denoted by $A \to B$. Note that $(A \to B) \subseteq (A \times B)$. The *composition* of two binary relations R_1 and R_2 is $R_2 \circ R_1 = \{(a, c) \mid \exists b\colon (a, b) \in R_1, (b, c) \in R_2\}$.

Finite state automata. We assume that the reader is familiar with finite state automata. For details, we refer to Hopcroft and Ullman (1979).

A *finite state automaton* (short: FSA) is a tuple $\mathcal{A} = (Q, \Delta, q_\mathrm{i}, q_\mathrm{f}, T)$ where Q and Δ are finite sets (*states* and *terminals*, respectively), $q_\mathrm{i}, q_\mathrm{f} \in Q$ (*initial* and *final state*, respectively), and $T \subseteq Q \times \Delta^* \times Q$ is finite (*transitions*). We call q the *source* and q' the *target* of a transition (q, u, q'). A *run* is a string θ of transitions such that the target of a transition is the source of the next transition in θ. The *language of* $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$.

Sorts. Sorts are a widespread concept in computer science: one can think of sorts as data types in a programming language. Let S be a set (of *sorts*). An S*-sorted set* is a tuple (Ω, sort) where Ω is a set and $\mathit{sort}\colon \Omega \to S$. We abbreviate (Ω, sort) by Ω and $\mathit{sort}^{-1}(s)$ by Ω_s for $s \in S$.

Now let Ω be an $(S^* \times S)$-sorted set. The *set of trees over* Ω is the S-sorted set T_Ω where $(\mathrm{T}_\Omega)_s = \{\omega(t_1, \dots, t_k) \mid s_1, \dots, s_k \in S, \omega \in \Omega_{(s_1 \cdots s_k, s)}, t_1 \in (\mathrm{T}_\Omega)_{s_1}, \dots, t_k \in (\mathrm{T}_\Omega)_{s_k}\}$ for each $s \in S$.

Multiple context-free grammars. A rule of a context-free grammar has the ability to concatenate the strings generated by its right-hand side non-terminals. *Multiple* context-free grammars extend this ability to concatenating string-*tuples*. This is done with the help of composition functions. Let Σ be a finite set. A *composition function w.r.t.* Σ is a function c that takes tuples of strings over Σ as arguments and returns a tuple of strings over Σ (i.e. there are $k \in \mathbb{N}$ and $s_1, \dots, s_k, s \in \mathbb{N}_+$ such that $c\colon (\Sigma^*)^{s_1} \times \dots \times (\Sigma^*)^{s_k} \to (\Sigma^*)^s$), and is defined by an equation $c((x_1^1, \dots, x_1^{s_1}), \dots, (x_k^1, \dots, x_k^{s_k})) = (u_1, \dots, u_s)$ where $u_1, \dots, u_s$ are strings of x_i^j's and symbols from Σ. We call c *linear* if each x_i^j occurs at most once in $u_1 \cdots u_s$. We sometimes write $[u_1, \dots, u_s]$ instead of c. Furthermore, setting $\mathit{sort}(c) = (s_1 \cdots s_k, s)$, the composition functions w.r.t. Σ form a sorted set.

The following example shows how linear composition functions are used in the rules of a multiple context-free grammar.

Example 1. Consider $G = (N, \Sigma, S, P)$ where $N = \{S, A, B\}$ and $\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}\}$ are finite sets (*non-terminals* and *terminals*, respectively), $S \in N$ (*initial non-terminal*) and P is a finite set (*rules*) that contains the following five objects:

$$\begin{aligned} &\rho_1 = S \to [x_1^1 x_2^1 x_1^2 x_2^2](A, B) \\ &\rho_2 = A \to [\mathrm{a}x_1^1, \mathrm{c}x_1^2](A) \qquad \rho_4 = A \to [\varepsilon, \varepsilon]() \\ &\rho_3 = B \to [\mathrm{b}x_1^1, \mathrm{d}x_1^2](B) \qquad \rho_5 = B \to [\varepsilon, \varepsilon](). \end{aligned}$$

We call G a multiple context-free grammar. Consider the rule ρ_1. Similar to a rule of a context-free grammar, ρ_1 rule has one left-hand side non-terminal (S) and zero or more right-hand side non-terminals (A and B). A *derivation of* G can be build by combining rules in P to form a tree according to their left- and right-hand side non-terminals. If a derivation starts with the initial non-terminal (here S), then it is called *complete*. Hence, each complete derivation in G has the form $d_{m,n} = \rho_1(\rho_2^m(\rho_4), \rho_3^n(\rho_5))$ for some $m, n \in \mathbb{N}$. If we replace each rule in a derivation by its composition function, we obtain a term of composition functions which can be evaluated. We call the resulting value the *yield* of a derivation. A derivation $d_{m,n}$ has yield $\mathrm{yd}(d_{m,n}) = \mathrm{a}^m \mathrm{b}^n \mathrm{c}^m \mathrm{d}^n$. The set of yields of all complete derivations is the *language of* G: $\mathcal{L}(G) = \{\mathrm{a}^m \mathrm{b}^n \mathrm{c}^m \mathrm{d}^n \mid m, n \in \mathbb{N}\}$. □

The following definition formalises the notions of ex. 1 and introduces some additional concepts.

Definition 2 (Seki et al., 1991)**.** A *multiple context-free grammar* (short: MCFG) is a tuple $G = (N, \Sigma, S, P)$ where N is a finite $\mathbb{N}_+$-sorted set (*non-terminals*), Σ is a finite set (*terminals*), $S \in N_1$ (*initial non-terminal*), P is a finite $(N^* \times N)$-sorted set of strings ρ of the form $A \to c(B_1, \dots, B_k)$ such that $A, B_1, \dots, B_k \in N$, c is a linear composition function, and $\mathit{sort}(c) =$

$(sort(B_1)\cdots sort(B_k), sort(A))$. The sort of ρ is $(B_1\cdots B_k, A)$. The *left-hand side* (short: lhs) of ρ is A. The *fanout of* ρ is $\mathrm{fanout}(\rho) = sort(A)$ and the *rank of* ρ is $rank(\rho) = k$. The elements of P are called *rules*.

The *set of derivations (resp. complete derivations) of* G is $\mathrm{D}_G = \mathrm{T}_P$ (resp. $\mathrm{D}^{\mathrm{c}}_G = (\mathrm{T}_P)_S$). Let $w \in \Sigma^*$ and $d = \rho(d_1, \ldots, d_k) \in \mathrm{D}_G$ with $\rho = A \to c(B_1, \ldots, B_k)$. The *yield of* d is $\mathrm{yd}(d) = c(\mathrm{yd}(d_1), \ldots, \mathrm{yd}(d_k))$. The *set of derivations of* w *in* G is $\mathrm{D}^{\mathrm{c}}_G(w) = \mathrm{yd}^{-1}(w) \cap \mathrm{D}^{\mathrm{c}}_G$. The *language of* A *in* G is $\mathcal{L}(G, A) = \{\mathrm{yd}(d) \mid d \in (\mathrm{T}_P)_A\}$. The *language of* G is $\mathcal{L}(G) = \mathcal{L}(G, S)$. Any language generated by an MCFG is called *multiple context-free* (short: mcf). □

A *context-free grammar* (short: CFG) is an MCFG where each non-terminal has sort 1. Each rule of a CFG has the form $A \to [u_0 x^1_{i(1)} u_1 \cdots x^1_{i(n)} u_n](B_1, \ldots, B_k)$. We abbreviate this rule by $A \to u_0 B_{i(1)} u_1 \cdots B_{i(n)} u_n$.

Weighted multiple context-free grammars. A weighted MCFG is obtained by assigning a weight to each rule of an (unweighted) MCFG. In this paper, the weights will be taken from a **p**artially **o**rdered **c**ommutative **mo**noid with **z**ero (short: POCMOZ). A *POCMOZ* is an algebra $(M, \odot, \mathbb{1}, \mathbb{0}, \trianglelefteq)$ where $\trianglelefteq$ is a partial order on M; $\odot$ is associative, commutative, decreasing (i.e. $m \odot m' \trianglelefteq m$), and monotonous (i.e. $m_1 \trianglelefteq m_2$ implies $m_1 \odot m \trianglelefteq m_2 \odot m$); $\mathbb{1}$ is neutral w.r.t. $\odot$; and $\mathbb{0}$ is absorbing w.r.t. $\odot$. We call M *factorisable* if for each $m \in M \setminus \{\mathbb{1}\}$, there are $m_1, m_2 \in M \setminus \{\mathbb{1}\}$ with $m = m_1 \odot m_2$. The *probability algebra* $\mathrm{Pr} = ([0, 1], \cdot, 1, 0, \leq)$ is a factorisable POCMOZ where $r = \sqrt{r} \cdot \sqrt{r}$ for each $r \in [0, 1)$.

Example 3 (continues ex. 1)**.** Consider the tuple (G, μ) where $\mu: P \to \mathrm{Pr}$ is a function where $\mu(\rho_1) = 1$, $\mu(\rho_2) = \mu(\rho_4) = 1/2$, $\mu(\rho_3) = 1/3$, and $\mu(\rho_5) = 2/3$. We call (G, μ) a weighted MCFG. The *weight* of the a derivation $d_{m,n}$ is obtained by multiplying the weights of all rule occurrences in it: $\mathrm{wt}(d_{m,n}) = 1/2^{m+1} \cdot 2/3^{n+1}$. □

Definition 4. A *weighted MCFG* (short: wMCFG) is a tuple (G, μ) where $G = (N, \Sigma, S, P)$ is an MCFG (*underlying MCFG*), $\mu: P \to M \setminus \{\mathbb{0}\}$ (*weight assignment*), and $(M, \odot, \mathbb{1}, \mathbb{0}, \trianglelefteq)$ is a factorisable POCMOZ.

(G, μ) inherits all objects associated with G. Let $d = \rho(d_1, \ldots, d_k) \in \mathrm{D}_G$. The *weight of* d is $\mathrm{wt}(d) = \mu(\rho) \odot \bigodot_{i=1}^{k} \mathrm{wt}(d_i)$. □

For the rest of this paper, we fix a wMCFG (G, μ) *with underlying MCFG* $G = (N, \Sigma, S, P)$ *and weight assignment* $\mu: P \to M$.

Chomsky-Schützenberger theorem. In the Chomsky-Schützenberger theorem for CFGs (cf. sec. 1), D contains strings of brackets where each opening bracket is matched by the corresponding closing bracket. This property can be described with an equivalence relation. Let Δ be a set (of opening brackets) and $\bar{\Delta}$ be the set (of closing brackets) that contains $\bar{\delta}$ for each $\delta \in \Delta$. We define $\equiv_\Delta$ as the smallest equivalence relation where $u\, \delta\bar{\delta}\, v \equiv_\Delta uv$ for each $\delta \in \Delta$ and $u, v \in \Delta^*$. The *Dyck language w.r.t.* Δ is $\mathrm{D}_\Delta = [\varepsilon]_{\equiv_\Delta}$.

In the Chomsky-Schützenberger representation for MCFGs, the brackets fulfil three functions: (i) *terminal brackets* $[\![_\sigma]\!]_\sigma$ stand for a terminal symbol σ, (ii) *component brackets* $[\![^\ell_\rho$ and $]\!]^\ell_\rho$ denote beginning and end of substrings produced by the ℓ-th component of a rule ρ, and (iii) *variable brackets* $[\![^j_{\rho,i}$ and $]\!]^j_{\rho,i}$ denote beginning and end of substrings produced by variable x^j_i in a rule ρ. As for CFGs, each opening bracket must be matched by the corresponding closing bracket. Furthermore, because applying a rule of an MCFG produces multiple strings simultaneously, we need to ensure that the brackets corresponding to the same application of a rule occur simultaneously. This is described with another equivalence relation. Let $\mathfrak{P}$ be a partition of Δ. Intuitively, each cell of $\mathfrak{P}$ is a set of (opening) brackets that occur simultaneously. We define $\equiv_{\mathfrak{P}}$ as the smallest equivalence relation on $\mathcal{P}\big((\Delta \cup \bar{\Delta})^*\big)$ where for each $\{\delta_1, \ldots, \delta_s\} \in \mathfrak{P}$ with $|\{\delta_1, \ldots, \delta_s\}| = s$, $u_0, \ldots, u_s, v_1, \ldots, v_s \in \mathrm{D}_\Delta$, and $L \subseteq (\Delta \cup \bar{\Delta})^*$:

$$\begin{aligned} &\{u_0\ \delta_1 v_1 \bar{\delta}_1\ u_1 \cdots\ \delta_s v_s \bar{\delta}_s\ u_s\} \cup L \\ \equiv_{\mathfrak{P}}\quad &\{u_0 \cdots u_s,\ \ v_1 \cdots v_s\} \qquad\quad \cup L. \end{aligned}$$

The *multiple Dyck language w.r.t.* $\mathfrak{P}$ is $\mathrm{mD}_{\mathfrak{P}} = \bigcup (L \mid L \in [\{\varepsilon\}]_{\equiv_{\mathfrak{P}}})$. Note that $\mathrm{mD}_{\mathfrak{P}} \subseteq \mathrm{D}_\Delta$.

Theorem 5 provides a representation of each mcf language by a multiple Dyck language (see above), a recognisable language (to ensure local consistency), and a homomorphism (to decode the bracket sequences into terminal strings). The corresponding construction is recalled in def. 6.

Theorem 5 (cf. Yoshinaka et al., 2010, thm. 3)**.** For every mcf language $L \subseteq \Sigma^*$ there are a homomorphism $h: (\Delta \cup \bar{\Delta})^* \to \Sigma^*$, a regular language $R \subseteq (\Delta \cup \bar{\Delta})^*$, and a multiple Dyck language $mD \subseteq (\Delta \cup \bar{\Delta})^*$ such that $L = h(R \cap mD)$. ■

Definition 6 (Denkinger, 2017, def. 3.6, 4.9, 5.15)**.** The *multiple Dyck language w.r.t. G* is $\mathrm{mD}_G = \mathrm{mD}_{\mathfrak{P}_G}$ where $\mathfrak{P}_G$ is the smallest set that contains the cell $\{\llbracket_\sigma\}$ for each $\sigma \in \Sigma$ and the cells $\{\llbracket^\ell_\rho \mid \ell \in [sort(A)]\}$ and $\{\llbracket^j_{\rho,i} \mid j \in [sort(B_i)]\}$ for each $\rho = A \to c(B_1, \dots, B_k) \in P$ and $i \in [k]$. Let $\Delta_G = \bigcup_{\mathfrak{p} \in \mathfrak{P}_G} \mathfrak{p}$. We denote the elements of $\overline{\Delta_G}$ by closing brackets, e.g. $\overline{\llbracket_\sigma} = \rrbracket_\sigma$, and let $\Omega_G = \Delta_G \cup \overline{\Delta_G}$.

The *homomorphism w.r.t. G*, denoted by hom_G, is the unique extension of $h\colon \Omega_G \to \Sigma \cup \{\varepsilon\}$ to strings where $h(\delta) = \sigma$ if δ is of the form $\llbracket_\sigma$ and $h(\delta) = \varepsilon$ otherwise.

The *automaton w.r.t. G*, denoted by $\mathcal{A}_G$, is the FSA $(Q, \Omega_G, S^1, \overline{S^1}, T)$ where $Q = \{A^\ell, \overline{A^\ell} \mid A \in N, \ell \in [sort(A)]\}$ and T is the smallest set such that for each rule $\rho \in P$ of the form

$$A \to [u_{1,0}y_{1,1}u_{1,1} \cdots y_{1,n_1}u_{1,n_1},\ \dots,\\ u_{s,0}y_{s,1}u_{s,1} \cdots y_{s,n_s}u_{s,n_s}](B_1, \dots, B_k)$$

where the ys are elements of X and the us are elements of Σ^*, we have (abbreviating $\llbracket_{\sigma_1}\rrbracket_{\sigma_2} \cdots \llbracket_{\sigma_k}\rrbracket_{\sigma_k}$ by $\widetilde{\sigma_1 \cdots \sigma_k}$) the following transitions in T:

(i) $\big(A^\ell,\ \llbracket^\ell_\rho \widetilde{u_{\ell,0}} \rrbracket^\ell_\rho,\ \bar{A}^\ell\big) \in T$ for every $\ell \in [s]$ with $n_\ell = 0$,

(ii) $\big(A^\ell,\ \llbracket^\ell_\rho \widetilde{u_{\ell,0}} \llbracket^j_{\rho,i},\ B^j_i\big) \in T$ for every $\ell \in [s]$ where $n_\ell \neq 0$ and $y_{\ell,1}$ is of the form x^j_i,

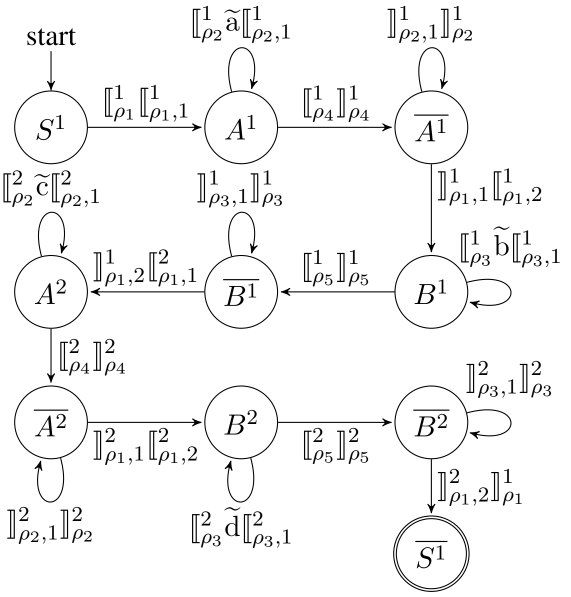
(iii) $\big(\bar{B}^j_i,\ \rrbracket^j_{\rho,i} \widetilde{u_{\ell,\kappa}} \llbracket^{j'}_{\rho,i'},\ B^{j'}_{i'}\big) \in T$ for every $\ell \in [s]$ and $\kappa \in [n_\ell - 1]$ where $y_{\ell,\kappa}$ is of the form x^j_i and $y_{\ell,\kappa+1}$ is of the form $x^{j'}_{i'}$, and

(iv) $\big(\bar{B}^j_i,\ \rrbracket^j_{\rho,i} \widetilde{u_{\ell,n_\ell}} \rrbracket^\ell_\rho,\ \bar{A}^\ell\big) \in T$ for every $\ell \in [s]$ where $n_\ell \neq 0$ and y_{ℓ,n_ℓ} is of the form x^j_i.

We abbreviate $\mathcal{L}(\mathcal{A}_G)$ by $\mathcal{R}_G$. □

Example 7 (continues ex. 1)**.** The automaton w.r.t. G is shown in fig. 1. An illustration of the application of $\equiv_{\mathfrak{P}_G}$ is given in the appendix (p. 11). □

The vanilla parser. The *vanilla parser* (i.e. alg. 3 from Denkinger, 2017), is shown in fig. 2 (top). Similar to the parser proposed by Hulden (2011), we divide it in three essential phases: (i) FSA constructions for the intersection of $\mathrm{hom}_G^{-1}(w)$ and $\mathcal{R}_G$, (ii) an extraction of (in our case *multiple*) Dyck words from the intersection, and (iii) the conversion of words into derivations.



$$\begin{tikzcd}
\text{start} \arrow[d] & & \\
S^1 \arrow[r, "\llbracket^1_{\rho_1}\llbracket^1_{\rho_1,1}"] & A^1 \arrow[loop above, "\llbracket^1_{\rho_2}\widetilde{\mathrm{a}}\llbracket^1_{\rho_2,1}"] \arrow[r, "\llbracket^1_{\rho_4}\rrbracket^1_{\rho_4}"] & \overline{A^1} \arrow[loop above, "\rrbracket^1_{\rho_2,1}\rrbracket^1_{\rho_2}"] \arrow[d, "\rrbracket^1_{\rho_1,1}\llbracket^1_{\rho_1,2}"] \\
A^2 \arrow[loop above, "\llbracket^2_{\rho_2}\widetilde{\mathrm{c}}\llbracket^2_{\rho_2,1}"] \arrow[d, "\llbracket^2_{\rho_4}\rrbracket^2_{\rho_4}"] & \overline{B^1} \arrow[loop above, "\rrbracket^1_{\rho_3,1}\rrbracket^1_{\rho_3}"] \arrow[l, "\rrbracket^1_{\rho_1,2}\llbracket^2_{\rho_1,1}"'] & B^1 \arrow[loop right, "\llbracket^1_{\rho_3}\widetilde{\mathrm{b}}\llbracket^1_{\rho_3,1}"] \arrow[l, "\llbracket^1_{\rho_5}\rrbracket^1_{\rho_5}"'] \\
\overline{A^2} \arrow[loop below, "\rrbracket^2_{\rho_2,1}\rrbracket^2_{\rho_2}"] \arrow[r, "\rrbracket^2_{\rho_1,1}\llbracket^2_{\rho_1,2}"'] & B^2 \arrow[loop below, "\llbracket^2_{\rho_3}\widetilde{\mathrm{d}}\llbracket^2_{\rho_3,1}"] \arrow[r, "\llbracket^2_{\rho_5}\rrbracket^2_{\rho_5}"'] & \overline{B^2} \arrow[loop right, "\rrbracket^2_{\rho_3,1}\rrbracket^2_{\rho_3}"] \arrow[d, "\rrbracket^2_{\rho_1,2}\rrbracket^1_{\rho_1}"] \\
 & & \overline{S^1}
\end{tikzcd}$$

Figure 1: Automaton w.r.t. G, cf. ex. 7.

Formally, the vanilla parser is the function $V\colon \Sigma^* \to (\mathrm{D}^{\mathrm{c}}_G)^\omega$ defined as

$$V = \textsc{map}(\textsc{toDeriv}) \circ \textsc{filter}(\mathrm{mD}_G)\\ \circ \textsc{sort}(\mu') \circ (\cap\, \mathcal{R}_G) \circ \mathrm{hom}_G^{-1}$$

where $\mathrm{hom}_G^{-1}(w) \cap \mathcal{R}_G$ is represented by an FSA for each $w \in \Sigma^*$ (phase (i)). $\mu'(u)$ is the product of the weights of each occurrence of a bracket of the form $\llbracket^\ell_\rho$ or $\rrbracket^\ell_\rho$ in u. These weights are fixed such that $\mu'(\llbracket^1_\rho\rrbracket^1_\rho \cdots \llbracket^\ell_\rho\rrbracket^\ell_\rho) = \mu(\rho)$ for each $\rho \in P$ with fanout ℓ. $\textsc{sort}(\mu')$ brings the elements of its argument, which is a subset of ${\Omega_G}^*$, in some descending order w.r.t. μ' and $\trianglelefteq$, returning a (possibly infinite) sequence of elements of ${\Omega_G}^*$, which we call *candidates*. Sequences are implemented as iterators. $\textsc{filter}(\mathrm{mD}_G)$ removes the candidates from its argument sequence that are not in mD_G while preserving the order (cf. Denkinger, 2017, alg. 2). (Both steps, $\textsc{sort}(\mu')$ and $\textsc{filter}(\mathrm{mD}_G)$, are phase (ii).) $\textsc{toDeriv}$ returns the derivation in G that corresponds to its argument (which is from the set $\mathcal{R}_G \cap \mathrm{mD}_G$), cf. Denkinger (2017, function fromBrackets, p. 20). $\textsc{map}(\textsc{toDeriv})$ applies $\textsc{toDeriv}$ to each candidate in its argument while preserving the order (phase (iii)).

Denkinger (2017, thm. 5.22) showed that $\textsc{take}(n) \circ V$ solves the n-best parsing problem.[1] We omit the additional restrictions that he imposed on the given wMCFG because they are only necessary to show the termination of his algorithm.

[1] In the following, we will gloss over the distinction between derivations and parses.

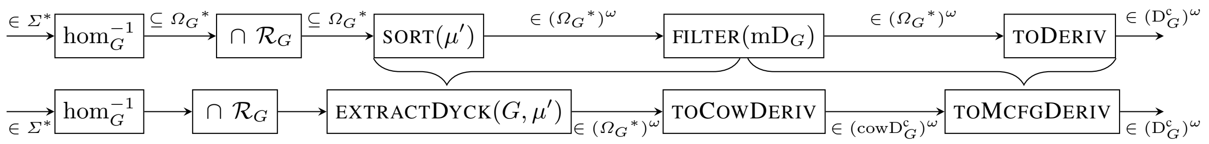


Figure 2: Visualisation of the vanilla parser (top) and the parser with the optimisations from sec. 4 (bottom).

3 Component-Wise Derivations

In sec. 4, we will outline modifications to the vanilla parser that make the extraction of the elements of mD_G from $\hom_G^{-1}(w) \cap \mathcal{R}_G$ efficient (items 2–4). To facilitate this, we first decompose FILTER(mD_G) into FILTER$(\mathrm{mD}_G) \circ$ FILTER(D_{Δ_G}), which is possible because $\mathrm{D}_{\Delta_G} \supseteq \mathrm{mD}_G$. Secondly, we implement FILTER$(\mathrm{D}_{\Delta_G}) \circ$ SORT(μ') with a dynamic programming algorithm (cf. Hulden, 2011, alg. 1, similar to Bar-Hillel et al., 1961, sec. 8). And lastly, we replace FILTER(mD_G) by steps that exploit the well-bracketing of the elements of D_{Δ_G}.

The elements of $\mathcal{R}_G \cap \mathrm{D}_{\Delta_G}$ can be represented as trees over rules of G.[2] We label the edges of those trees to allow us to check if vertices that correspond to the same application of a rule of the MCFG G match. The resulting objects are called *component-wise derivations*. The set $\mathcal{R}_G \cap \mathrm{D}_{\Delta_G}$ is characterised in terms a CFG G_{cf}.

Definition 8. Let $\rho \in P$ be a rule of the form $A \to [u_1, \dots, u_s](B_1, \dots, B_k)$, $\ell \in [s]$, and u_ℓ be of the form $w_0 x_{i(1)}^{j(1)} w_1 \cdots x_{i(n)}^{j(n)} w_n$ for some $w_0, \dots, w_n \in \Sigma^*$. We define the rule

$$\rho^{(\ell)} = A^\ell \to \big[\!\big[_\rho^\ell \widetilde{w_0} v_1 \widetilde{w_1} \cdots v_n \widetilde{w_n} \big]\!\big]_\rho^\ell$$

where each $v_\kappa = [\![_{\rho,i(\kappa)}^{j(\kappa)} B_{i(\kappa)}^{j(\kappa)}]\!]_{\rho,i(\kappa)}^{j(\kappa)}$. The *context-free CS approximation of* G (short: *CFA*), denoted by G_{cf}, is the CFG $(N_{\mathrm{cf}}, \Omega_G, S^1, P_{\mathrm{cf}})$ where $N_{\mathrm{cf}} = \{A^\ell \mid A \in N, \ell \in [\mathit{sort}(A)]\}$ and $P_{\mathrm{cf}} = \{\rho^{(\ell)} \mid \rho \in P, \ell \in [\mathrm{fanout}(\rho)]\}$. □

Observation 9. $\mathrm{D}_{\Delta_G} \cap \mathcal{R}_G = \mathcal{L}(G_{\mathrm{cf}})$. ■

We introduce *component-wise derivations* to relate the derivations of G_{cf} with those of G.

Definition 10. Let $\ell \in \mathbb{N}_+$ and t be a tree whose vertices are labelled with elements of P and whose edges are labelled with elements of $\mathbb{N}_+ \times \mathbb{N}_+$. The *label at the root of* t is denoted by $\mathrm{root}(t)$. The *set of labels of the outgoing edges from the root of* t is denoted by $\mathrm{out}(t)$. A (i,j)-*subtree of* t, is a sub-graph of t consisting of all the vertices (and their edges) reachable from some target vertex of the outgoing edge from the root that is labelled with (i,j). If there is a unique (i,j)-subtree of t, then we denote it by $\mathrm{sub}_{(i,j)}(t)$. Now let $\mathrm{root}(t) = A \to [u_1, \dots, u_s](B_1, \dots, B_k)$. We call t an *(ℓ-)component-wise derivation, short: (ℓ-)cow derivation, of* G if the following four requirements are met: (i) $\mathrm{out}(t)$ contains exactly the pairs (i,j) such that x_i^j occurs in u_ℓ, (ii) a unique (i,j)-subtree of t exists, (iii) $\mathrm{root}(\mathrm{sub}_{(i,j)}(t))$ has lhs B_i, and (iv) $\mathrm{sub}_{(i,j)}(t)$ is a j-cow derivation for each $(i,j) \in \mathrm{out}(t)$. We denote the set of cow derivations of G whose root's lhs is S by $\mathrm{cowD}^{\mathrm{c}}_G$. The set of ℓ-cow derivations whose root's label has lhs A is denoted by $\ell\text{-}\mathrm{cowD}^A_G$. □

An example of a cow derivation is shown in fig. 3a. The root is the top-most vertex.

Definition 11. Let $\rho = A \to c(B_1, \dots, B_k) \in P$, $\ell \in [\mathrm{fanout}(\rho)]$, and the ℓ-th component of c be $u_0 x_{i(1)}^{j(1)} u_1 \cdots x_{i(n)}^{j(n)} u_n$ with $u_1, \dots, u_n \in \Sigma^*$. Furthermore, for each $\kappa \in [n]$, let $t_\kappa \in j(\kappa)\text{-}\mathrm{cowD}_G^{B_{i(\kappa)}}$. By $\rho\big\langle (i(\kappa), j(\kappa))/t_\kappa \mid \kappa \in [n] \big\rangle$, we denote the cow derivation t such that $\mathrm{root}(t) = \rho$, $\mathrm{out}(t) = \{(i(\kappa), j(\kappa)) \mid \kappa \in [n]\}$, and for each $\kappa \in [n]$: $\mathrm{sub}_{(i(\kappa), j(\kappa))}(t) = t_\kappa$. □

Lemma 12. There is a bijection toCowD between $\mathcal{L}(G_{\mathrm{cf}})$ and $\mathrm{cowD}^{\mathrm{c}}_G$. ■

Proof sketch. We define the partial function toCowD from ${\Omega_G}^*$ to cow derivations of G as follows: $\mathrm{toCowD}(u) = \rho\big\langle (i(\kappa), j(\kappa))/\mathrm{toCowD}(v_\kappa) \mid \kappa \in [n] \big\rangle$ if u is of the form

$$\Big[\!\Big[_\rho^\ell \widetilde{u_0} [\![_{\rho,i(1)}^{j(1)} v_1]\!]_{\rho,i(1)}^{j(1)} \widetilde{u_1} \cdots [\![_{\rho,i(n)}^{j(n)} v_n]\!]_{\rho,i(n)}^{j(n)} \widetilde{u_n} \Big]\!\Big]_\rho^\ell$$

for some rule $\rho = A \to c(B_1, \dots, B_k)$ where the ℓ-th component of c is $u_0 x_{i(1)}^{j(1)} u_1 \cdots x_{i(n)}^{j(n)} u_n$ with $u_1, \dots, u_n \in \Sigma^*$; otherwise, $\mathrm{toCowD}(u)$ is undefined. The partial function toCowD is a bijection between $\mathcal{L}(G_{\mathrm{cf}})$ and $\mathrm{cowD}^{\mathrm{c}}_G$ (proven in appendix A.2). ■

[2]Those trees correspond to the derivations of the *guiding grammar* in the coarse-to-fine parsing approach of Barthélemy et al. (2001, sec. 3).

Example 13 (continues ex. 1)**.** We construct $G_{\mathrm{cf}} = (\{S^1, A^1, A^2, B^1, B^2\}, \Omega_G, S^1, P_{\mathrm{cf}})$ where P_{cf} contains, among others, the following rules:

$$\begin{aligned}
\rho_1^{(1)} &= S^1 \to \llbracket^{1}_{\rho_1} \llbracket^{1}_{\rho_{1,1}} A^1 \rrbracket^{1}_{\rho_{1,1}} \llbracket^{1}_{\rho_{1,2}} B^1 \rrbracket^{1}_{\rho_{1,2}} \\
&\qquad\qquad \llbracket^{2}_{\rho_{1,1}} A^2 \rrbracket^{2}_{\rho_{1,1}} \llbracket^{2}_{\rho_{1,2}} B^2 \rrbracket^{2}_{\rho_{1,2}} \rrbracket^{1}_{\rho_1}, \\
\rho_3^{(1)} &= B^1 \to \llbracket^{1}_{\rho_3} \widetilde{\mathrm{b}} \llbracket^{1}_{\rho_{3,1}} B^1 \rrbracket^{1}_{\rho_{3,1}} \rrbracket^{1}_{\rho_3}, \\
\rho_4^{(1)} &= A^1 \to \llbracket^{1}_{\rho_4} \rrbracket^{1}_{\rho_4}, \quad \rho_4^{(2)} = A^2 \to \llbracket^{2}_{\rho_4} \rrbracket^{2}_{\rho_4}, \\
\rho_5^{(1)} &= B^1 \to \llbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_5}, \quad \ldots .
\end{aligned}$$

Figure 3a shows the image of the word

$$\begin{aligned}
&\llbracket^{1}_{\rho_1} \llbracket^{1}_{\rho_{1,1}} \llbracket^{1}_{\rho_4} \rrbracket^{1}_{\rho_4} \rrbracket^{1}_{\rho_{1,1}} \\
&\quad \llbracket^{1}_{\rho_{1,2}} \llbracket^{1}_{\rho_3} \widetilde{\mathrm{b}} \llbracket^{1}_{\rho_{3,1}} \llbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_{3,1}} \rrbracket^{1}_{\rho_3} \rrbracket^{1}_{\rho_{1,2}} \\
&\quad \llbracket^{2}_{\rho_{1,1}} \llbracket^{2}_{\rho_4} \rrbracket^{2}_{\rho_4} \rrbracket^{2}_{\rho_{1,1}} \quad \llbracket^{2}_{\rho_{1,2}} \llbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_{1,2}} \rrbracket^{1}_{\rho_1}
\end{aligned}$$

in $\mathcal{L}(G_{\mathrm{cf}})$ under toCowD. □

In the following, we define a property called *consistency* to discern those cow derivations that correspond to derivations of the MCFG G.

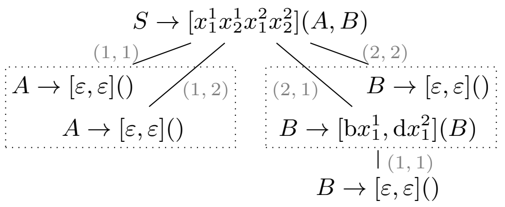
Definition 14. Let $s \in \mathbb{N}_+$ and $t_1, \ldots, t_s$ be cow derivations of G. We call the set $\{t_1, \ldots, t_s\}$ *consistent* if there is a rule $\rho = A \to c(B_1, \ldots, B_k) \in P$ such that $\mathrm{root}(t_1) = \ldots = \mathrm{root}(t_s) = \rho$, $s = \mathit{sort}(A)$, and for each $i \in [k]$: the set $\{\mathrm{sub}_{(i,j)}(t_\ell) \mid \ell \in [s], j \in [\mathit{sort}(B_i)] \colon (i,j) \in \mathrm{out}(t_\ell)\}$ is consistent. If $s = 1$, then we also call t_1 *consistent*. □

The cow derivation shown in fig. 3a is *not* consistent. If we consider the set of nodes that is reachable from the root via edges labelled with a tuple whose first component is 2 (the right dotted box), then it is easy to see that the rules at these nodes are not equal. A consistent cow derivation is shown in the appendix (fig. 6).

Proposition 15. $\textsc{toDeriv} \circ \mathrm{toCowD}^{-1}$ is a bijection between the consistent cow derivations in $\mathrm{cowD}^{\mathrm{c}}_G$ and $\mathrm{D}^{\mathrm{c}}_G$. ■

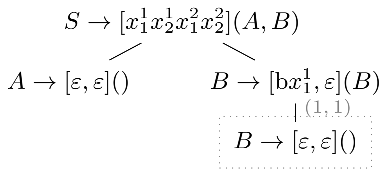
4 Optimisations

In this section, we describe several improvements to the vanilla parser (cf. end of sec. 2). Since the definitions of $\mathcal{A}_G$, hom_G, and mD_G do not depend on the word w, we may compute appropriate representations for these objects before the beginning of the parsing process, and store them persistently.



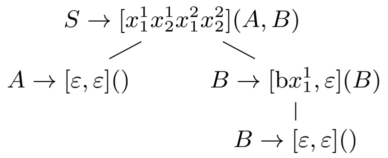
$S \to [x_1^1 x_2^1 x_1^2 x_2^2](A, B)$
$(1,1)$
$(2,2)$
$A \to [\varepsilon, \varepsilon]()$
$(1,2)$
$(2,1)$
$B \to [\varepsilon, \varepsilon]()$
$A \to [\varepsilon, \varepsilon]()$
$B \to [\mathrm{b}x_1^1, \mathrm{d}x_1^2](B)$
$(1,1)$
$B \to [\varepsilon, \varepsilon]()$

(a) A cow derivation. The dotted boxes show clusters of nodes that are reachable from the root via edges labelled with matching first components.



$S \to [x_1^1 x_2^1 x_1^2 x_2^2](A, B)$
$A \to [\varepsilon, \varepsilon]()$
$B \to [\mathrm{b}x_1^1, \varepsilon](B)$
$(1,1)$
$B \to [\varepsilon, \varepsilon]()$

(b) Construction of new rules for each cluster in fig. 3a. If there were any unused nonterminals in these constructed rules, they are removed and the indices of variables changed accordingly. For each cluster, all reachable nodes are clustered via the first component of the labels as in fig. 3a.



$S \to [x_1^1 x_2^1 x_1^2 x_2^2](A, B)$
$A \to [\varepsilon, \varepsilon]()$
$B \to [\mathrm{b}x_1^1, \varepsilon](B)$
$B \to [\varepsilon, \varepsilon]()$

(c) Construction of new rules from the clusters in fig. 3b.

Figure 3: A strategy to convert a non-consistent cow derivation into a complete derivation of an MCFG.

In the following, we briefly describe each improvement that we applied to the vanilla parser:

1. Let us call a rule ρ in G *w-consistent* if each string of terminals that occurs in (the composition function of) ρ is a substring of w. A rule is called *useful w.r.t. w* if it occurs in some complete derivation of G in which each rule is w-consistent. In the construction of the FSA for $\mathcal{R}_G \cap \mathrm{hom}_G^{-1}(w)$, we only calculate the transitions that relate to rules of G that are useful w.r.t. w.

2. The function $\textsc{filter}(\mathrm{mD}_G)$ is decomposed into $\textsc{filter}(\mathrm{mD}_G) \circ \textsc{filter}(\mathrm{D}_{\Delta_G})$ in preparation for the next two items.

3. $\textsc{filter}(\mathrm{D}_{\Delta_G}) \circ \textsc{sort}(\mu')$ is implemented with the algorithm $\textsc{extractDyck}(G, \mu')$ that uses dynamic programming to extract Dyck words from the language of the given FSA more efficiently. For this, we extend alg. 1 by Hulden (2011) to use weights such that it returns the elements in descending order w.r.t. μ' and $\trianglelefteq$ (see appendix A.3, alg. 3). In our implementation, we change this al-

Algorithm 1 reads off cow derivations from words of the CFA of G.

Input: $v \in \mathcal{L}(G_{\text{cf}})$
Output: $\text{toCowD}(v)$

```
1:  function TOCOWDERIV($v$)
2:      $t \leftarrow$ empty graph
3:      $v \leftarrow$ new vertex
4:      $pd \leftarrow \varepsilon$                    ▷ pushdown of vertices
5:      for all $\sigma$ in $v$ from left to right do
6:          if $\sigma$ is of the form $[\![^{\ell}_{\rho}$ then
7:              add vertex $v$ to $t$ with label $\rho$
8:              push vertex $v$ on top of $pd$
9:          else if $\sigma$ is of the form $[\![^{j}_{\rho,i}$ then
10:             $v' \leftarrow$ top-most vertex on $pd$
11:             $v \leftarrow$ new vertex
12:             add edge $(v' \to v)$ to $t$ with label $(i, j)$
13:         else if $\sigma$ is of the form $]\!]^{j}_{\rho,i}$ then
14:             remove top-most vertex from $pd$
15:     return $t$
```

gorithm even further such that items are explored in a similar fashion as in the CKY-algorithm (Kasami, 1966; Younger, 1967; Cocke and Schwartz, 1970).

4. For FILTER(mD_G), instead of *isMember*$'$ by Denkinger (2017, p. 28–30), which runs in quadratic time, we use the composition of two algorithms that run in linear time:
   - alg. 1, which reads a cow derivation off a given word in $\mathcal{R}_G \cap \text{D}_{\Delta_G}$, and
   - an algorithm that checks a given cow derivation for consistency. (This is similar to alg. 2; but instead of derivations, we return Boolean values. The algorithm is given explicitly in sec. A.3.)

5. Algorithm 2 computes the bijection between cowD^{c}_G and D^{c}_G (see prop. 15). Analogously to def. 14, the function TOMCFGDERIV' checks a set of cow derivations for equivalence of the root symbol and the function COLLECTCHILDREN groups the subtrees via the first component of the successor labels. It is easy to see that TOMCFGDERIV(t) is only defined if the cow derivation t is consistent (cf. item 4). Thus, we use TOMCFGDERIV in combination with TOCOWDERIV to replace MAP(TODERIV) $\circ$ FILTER(mD_G).

   The time complexity of alg. 2 is linear in the number of vertices of the given cow derivation. This number, in turn, is linear in the length of the processed candidate.

Algorithm 2 converts a consistent element of cowD^{c}_G into a complete derivation of G.

Input: $t \in \text{cowD}^{\text{c}}_G$

Output: $\begin{cases} \text{TODERIV}\big(\text{toCowD}^{-1}(t)\big) & \text{if } t \text{ is consistent} \\ \text{undefined} & \text{otherwise} \end{cases}$

```
1:  function TOMCFGDERIV($t$)
2:      return TOMCFGDERIV'($\{t\}$)

3:  function TOMCFGDERIV'($T$)
4:      if not $\bigwedge_{t,t' \in T} \big(\text{root}(t) = \text{root}(t')\big)$ then
5:          return undefined
6:      $(T_1, \ldots, T_k) \leftarrow$ COLLECTCHILDREN($T$)
7:      for $i \in [k]$ do
8:          $t_i \leftarrow$ TOMCFGDERIV'($T_i$)
9:          if $t_i =$ undefined then
10:             return undefined
11:     $\{\sigma\} \leftarrow \{\text{root}(t) \mid t \in T\}$
12:     return $\sigma(t_1, \ldots, t_k)$

13: function COLLECTCHILDREN($T$)
14:     $\{k\} \leftarrow \{rank(\text{root}(t)) \mid t \in T\}$
15:     for $i \in [k]$ do
16:         $T_i \leftarrow \{\text{sub}_{(i,j)}(t)$
                    $\mid t \in T, (i, j) \in \text{out}(t)\}$
17:     return $(T_1, \ldots, T_k)$
```

The parser obtained by applying items 1 to 5 to the vanilla parser is visualised in fig. 2 (bottom). It is sound and complete.[3] The following two modifications (items 6 and 7) destroy both soundness and completeness. Item 6 allows only the best intermediate results to be processed further and limits the results to a subset of those of the vanilla parser. In item 7, we compensate this by an approximation we consider useful in practise.

6. EXTRACTDYCK is extended with an optional implementation of beam search by limiting the amount of items for certain groups of state spans to a specific number (*beam width*), cf. Collins (1999). In our implementation, we chose these groups of state spans such that they correspond to equal states in

[3] A parser is *complete* if it (eventually) computes all complete derivations of the given word in the given grammar. A parser is called *sound* if all computed parses are complete derivations of the given word in the given grammar.

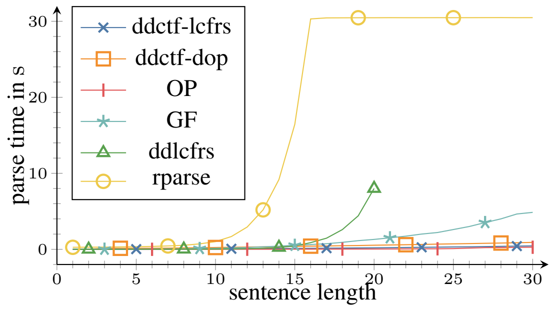
the automaton for $\hom_G^{-1}(w)$. Moreover, we introduce a variable that limits the number of candidates that are yielded by Algorithm 3 (*candidate count*). Both variables are the *meta-parameters* of our parser.

7. We introduce a fallback mechanism for the case that FILTER(mD_G) has input candidates but an empty output. Usually, in that case, we would suggest there is no derivation for w in G, yet for robustness, it is preferable to output some parse. Figure 3 illustrates a strategy to construct a complete derivation from any complete *cow* derivation with an example.

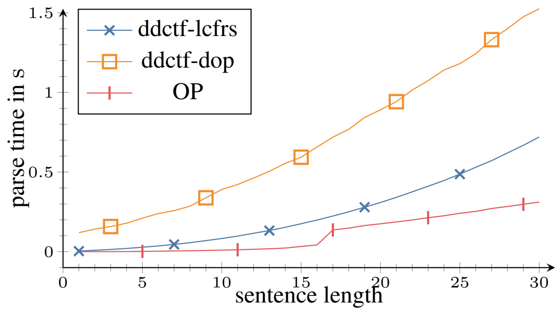
5 Evaluation and Conclusion

We implemented the parser with the modifications sketched in sec. 4 for ε-free and simple wMCFGs,[4] but no problems should arise generalising this implementation to arbitrary wMCFGs. The implementation is available as a part of Rustomata,[5] a framework for weighted automata with storage written in the programming language Rust. We used the *NeGra corpus* (German newspaper articles, 20,602 sentences, 355,096 tokens; Skut et al., 1998) to compare our parser to *Grammatical Framework* (Angelov and Ljunglöf, 2014), *rparse* (Kallmeyer and Maier, 2013), and *disco-dop* (van Cranenburgh et al., 2016) with respect to parse time and accuracy.[6] Our experiments were conducted on defoliated trees, i.e. we removed the leaves from each tree in the corpus. Parsing was performed on gold part-of-speech tags.

We performed a variant of *ten-fold cross validation* (short: TFCV; cf. Mosteller and Tukey, 1968), i.e. we split the corpus into ten consecutive parts; each part becomes the validation set in one iteration while the others serve as training set. We used the first iteration to select suitable values for our meta-parameters and the remaining nine for validation. In case of Rustomata, a binarised and markovized grammar was induced with discodop (head-outward binarisation, $v = 1$, $h = 2$, cf. Klein and Manning, 2003) in each iteration. For all other parsers, we induced a probabilistic LCFRS with the respective default configurations (for details, cf. the evaluation scripts). After that, we ran our parser on each sentence of the validation set and recorded the parse time and the computed 1-best parse. The computed parses were evaluated against the gold parses of the validation set w.r.t. precision, recall, and f_1-score (according to the labelled parseval measures, cf. Black et al., 1991; Collins, 1997, we used the implementation by van Cranenburgh et al., 2016).



(a) NeGra corpus, $|w| \leq 30$ (for ddlcfrs: $|w| \leq 20$)



(b) Lassy corpus, $|w| \leq 30$

Figure 4: Median parse times

Previous experiments with an implementation of the vanilla parser already struggled with small subsets (we used grammars extracted from 250–1500 parse trees) of the NeGra corpus. Therefore, we omit evaluation of the vanilla parser.

Meta-parameters. A grid search for meta-parameters was performed on sentences of up to 20 tokens (see the appendix, tab. 2, for a detailed listing). The results suggested to set the beam width to 200 and the candidate count to 10,000.

Comparison to other parsers. The experiments were performed on sentences with up to 30 tokens. We instructed rparse, Grammatical Framework (short: GF) and Rustomata (short: OP) to stop parsing each sentence after 30 seconds (*timeout*). Disco-dop did not permit passing a timeout. In the case of disco-dop's LCFRS parser (short: ddlcfrs), we limited the validation set to sentences

[4]A wMCFG G is called ε-free and simple if each composition function that occurs in the rules of G is either of the form $[u_1, \ldots, u_s]$ for some non-empty strings of variables $u_1, \ldots, u_s$, or of the form $[t]$ for some terminal symbol t.

[5]available on `https://github.com/tud-fop/rustomata`. We used commit `867a451` for evaluation.

[6]The evaluation scripts are available on `https://github.com/truprecht/rustomata-eval`.

parser	precision	recall	f_1-score	coverage
	NeGra corpus, $\|w\| \leq 20$			
ddctf-dop	81.34	81.44	81.39	100%
ddlcfrs	74.85	73.99	74.42	100%
ddctf-lcfrs	74.78	73.89	74.33	100%
OP	74.38	74.22	74.30	99.20%
GF	70.98	72.17	71.57	98.15%
rparse	71.15	41.67	52.56	76.31%
	NeGra corpus, $\|w\| \leq 30$			
ddctf-dop	77.91	78.22	78.07	100%
ddctf-lcfrs	70.32	69.70	70.01	100%
OP	68.81	69.27	69.04	99.22%
GF	66.56	68.16	67.35	98.19%
rparse	71.19	23.73	35.59	57.33%
	Lassy corpus, $\|w\| \leq 30$			
ddctf-dop	73.78	73.37	73.57	100%
ddctf-lcfrs	61.23	59.73	60.52	100%
OP	50.92	51.42	51.17	100%

Table 1: Precision, recall, f_1-score, and coverage

of at most 20 tokens, since ddlcfrs frequently exceeded 30 seconds of parse time for longer sentences in preliminary tests. Disco-dop's coarse-to-fine data-oriented parser (short: ddctf-dop) and disco-dop's coarse-to-fine LCFRS parser (short: ddctf-lcfrs) rarely exceeded 30 seconds of parse time in preliminary tests and we let them run on sentences of up to 30 tokens without the timeout.

Figure 4a shows the parse times for each sentence length and parser. The parsers ddctf-dop, ddctf-lcfrs, GF, and OP perform similar for sentences of up to 20 tokens. The parse times of rparse and ddlcfrs grow rapidly after 10 and 16 tokens, respectively. Rparse even exceeds the timeout for more than half of the test sentences that are longer than 15 tokens. For sentences with up to 30 tokens, the parse times of ddctf-dop, ddctf-lcfrs and OP seem to remain almost constant.

Table 1 shows the accuracy (i.e. precision, recall, and f_1-score) and the coverage (i.e. the percentage of sentences that could be parsed) for each parser on the validation set. We report these scores to assert a correct implementation of our parser and to compare the different approximation strategies (and our fallback mechanism) implemented in the parsers. The low coverage of rparse stems from the frequent occurrences of timeouts. They also depress the recall for rparse. For sentences with at most 20 tokens, ddlcfrs, ddctf-lcfrs and OP perform very similar. These three parsers are outperformed by ddctf-dop in all aspects. For sentences of up to 30 tokens, the scores of all tested parsers drop similarly. However, ddctf-dop's scores drop the least amount.

We repeated a part of the experiments with the Lassy corpus (Lassy Small, various kinds of written Dutch, 65,200 sentences, 975,055 tokens; van Noord et al., 2013). Since it is considerably larger than the NeGra corpus, we limited the experiments to one iteration of TFCV, and we only investigate OP, ddctf-lcfrs, and ddctf-dop. The results are shown in fig. 4b (parse time) and at the bottom of tab. 1 (accuracy). Figure 4b shows the difference of ddctf-lcfrs, ddctf-dop and OP in terms of parse times (which is not discernible in fig. 4a). This plot shows that OP maintains very small parse times – even for large copora – compared to the state-of-the-art parser disco-dop.

All in all, our parser performs comparable to state-of-the-art MCFG parsers (GF, rparse, ddlcfrs, ddctf-lcfrs) and, using the NeGra corpus, it shows excellent results in parse time and good results in accuracy. Moreover, our parser can deal with any ε-free and simple MCFG provided by an external tool, making it more flexible than disco-dop and rparse. However, we are not able to compete with ddctf-dop in terms of accuracy, since discontinuous data-oriented parsing is a more accurate formalism (van Cranenburgh and Bod, 2013).

6 Future Work

We see potential to improve the fallback mechanism explained in sec. 4. For now, we only considered reporting the first cow derivation. By introducing some *degree of consistency* of cow derivations, we could select a cow derivation that is closer to a derivation of G.

Since recognisable languages are closed under inverse homomorphisms, we can use any recognisable language as input for hom_G^{-1} (cf. fig. 2) without changing the rest of the pipeline. This is useful when the input of the parsing task is ambiguous, as in lattice-based parsing (e.g. Goldberg and Tsarfaty, 2008).

Moreover, since weighted recognisable languages are closed under inverse homomorphisms and scalar product, we can even use a weighted recognisable language as input for hom_G^{-1}, as in the setting of Rastogi et al. (2016).

Acknowledgements

We thank our colleague Kilian Gebhardt as well as the anonymous reviewers for their insightful comments on drafts of this paper.

References

Krasimir Angelov and Peter Ljunglöf. 2014. 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.

Yehoshua Bar-Hillel, Micha Asher Perles, and Eli Shamir. 1961. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14:143–172.

François Barthélemy, Pierre Boullier, Philippe Deschamp, and Éric Villemonte de la Clergerie. 2001. Guided parsing of range concatenation languages. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*.

Ezra Black, Steven Abney, Dan Flickinger, Claudia Gdaniec, Ralph Grishman, Philip Harrison, Donald Hindle, Robert Ingria, Fred Jelinek, Judith Klavans, Mark Liberman, and Tomek Strzalkowski. 1991. A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars. In *Proceedings of the Workshop on Speech and Natural Language*, pages 306–311. Association for Computational Linguistics.

Noam Chomsky. 1956. Three models for the description of language. *IEEE Transactions on Information Theory*, 2(3):113–124.

Noam Chomsky. 1959. On certain formal properties of grammars. *Information and control*, 2(2):137–167.

Noam Chomsky and Marcel Paul Schützenberger. 1963. The algebraic theory of context-free languages. *Computer Programming and Formal Systems, Studies in Logic*, pages 118–161.

Maximin Coavoux and Benoit Crabbé. 2017. Incremental discontinuous phrase structure parsing with the gap transition. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1259–1270. Association for Computational Linguistics.

John Cocke and J. T. Schwartz. 1970. Programming languages and their compilers: Preliminary notes. techreport, Courant Institute of Mathematical Sciences, New York University.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th annual meeting on Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics.

Michael Collins. 1999. *Head-driven statistical models for natural language parsing*. Ph.D. thesis.

Andreas van Cranenburgh and Rens Bod. 2013. Discontinuous parsing with an efficient and accurate dop model. In *Proceedings of the 13th International Conference on Parsing Technologies*.

Andreas van Cranenburgh, Remko Scha, and Rens Bod. 2016. Data-oriented parsing with discontinuous constituents and function tags. *Journal of Language Modelling*, 4(1):57.

Tobias Denkinger. 2017. Chomsky-Schützenberger parsing for weighted multiple context-free languages. *Journal of Language Modelling*, 5(1):3.

Yoav Goldberg and Reut Tsarfaty. 2008. A single generative model for joint morphological segmentation and syntactic parsing. *Proceedings of ACL-08: HLT*, pages 371–379.

John Edward Hopcroft and Jeffrey David Ullman. 1979. *Introduction to Automata Theory, Languages and Computation*, 1st edition.

Mans Hulden. 2011. Parsing CFGs and PCFGs with a Chomsky-Schützenberger representation. In *Human Language Technology. Challenges for Computer Science and Linguistics*, volume 6562 of *Lecture Notes in Computer Science*, pages 151–160.

Aravind K. Joshi. 1985. *Tree adjoining grammars: How much context-sensitivity is needed for characterizing structural descriptions?*, chapter 6. Cambdrige University Press.

Aravind Krishna Joshi, Leon S. Levy, and Masako Takahashi. 1975. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163.

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

T. Kasami. 1966. An efficient recognition and syntax-analysis algorithm for context-free languages. techreport R-257, AFCRL.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics.

Wolfgang Maier. 2015. Discontinuous incremental shift-reduce parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1202–1212, Beijing, China. Association for Computational Linguistics.

Frederick Mosteller and John Wilder Tukey. 1968. Data analysis, including statistics. In G. Lindzey and E. Aronson, editors, *Handbook of Social Psychology*, volume 2. Addison-Wesley.

Gertjan van Noord, Gosse Bouma, Frank Van Eynde, Daniël de Kok, Jelmer van der Linde, Ineke Schuurman, Erik Tjong Kim Sang, and Vincent Vandeghinste. 2013. *Large Scale Syntactic Annotation of Written Dutch: Lassy*, pages 147–164. Springer Berlin Heidelberg, Berlin, Heidelberg.

Pushpendre Rastogi, Ryan Cotterell, and Jason Eisner. 2016. Weighting finite-state transductions with neural context. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 623–633, San Diego, California. Association for Computational Linguistics.

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

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

Wojciech Skut, Thorsten Brants, Brigitte Krenn, and Hans Uszkoreit. 1998. A Linguistically Interpreted Corpus of German Newspaper Text. In *Proceedings of the 10th European Summer School in Logic, Language and Information. Workshop on Recent Advances in Corpus Annotation.*

Krishnamurti Vijay-Shanker, David Jeremy Weir, and Aravind Krishna Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Meeting on Association for Computational Linguistics*, pages 104–111.

Ryo Yoshinaka, Yuichi Kaji, and Hiroyuki Seki. 2010. Chomsky-Schützenberger-type characterization of multiple context-free languages. In Adrian-Horia Dediu, Henning Fernau, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications*, pages 596–607.

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n3. *Information and Control*, 10(2):189–208.

A Appendices

A.1 Additional Examples

Example 1 (continuing from p. 2). Figure 5 shows graphical representations of a derivation and the corresponding term over composition functions. □

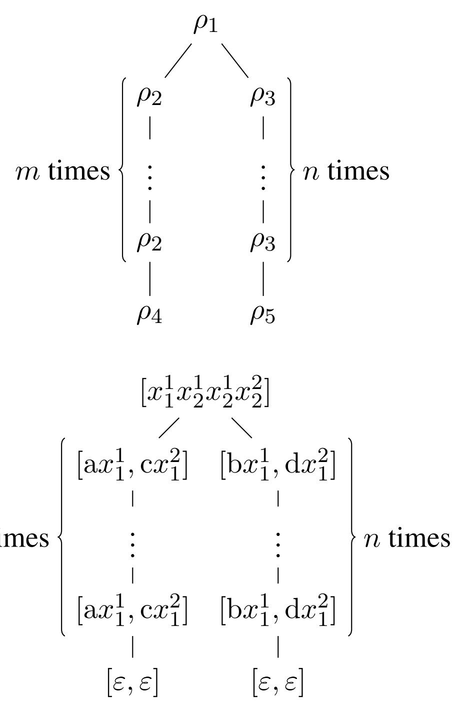
Example 7 (continuing from p. 4). The following calculation reduces a word of $\mathcal{A}_G$ to ε using the equivalence relation $\equiv_{\mathfrak{P}_G}$ (abbreviated by $\equiv$) and thereby proves that it is an element of mD_G. In each step, we point out, which cell of $\mathfrak{P}_G$ was/were used. Note that the set obtained after two applications of $\equiv$ has two elements.

$$
\begin{aligned}
&\Big\{ \llbracket^{1}_{\rho_1} \llbracket^{1}_{\rho_1,1} \llbracket^{1}_{\rho_4} \rrbracket^{1}_{\rho_4} \rrbracket^{1}_{\rho_1,1} \llbracket^{1}_{\rho_1,2} \llbracket^{1}_{\rho_3} \widetilde{\mathrm{b}} \llbracket^{1}_{\rho_3,1} \llbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_3,1} \rrbracket^{1}_{\rho_3} \rrbracket^{1}_{\rho_1,2} \\
&\llbracket^{2}_{\rho_1,1} \llbracket^{2}_{\rho_4} \rrbracket^{2}_{\rho_4} \rrbracket^{2}_{\rho_1,1} \llbracket^{2}_{\rho_1,2} \llbracket^{2}_{\rho_3} \widetilde{\mathrm{d}} \llbracket^{2}_{\rho_3,1} \llbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_3,1} \rrbracket^{2}_{\rho_3} \rrbracket^{2}_{\rho_1,2} \rrbracket^{1}_{\rho_1} \Big\} \\
&\equiv \Big\{ \llbracket^{1}_{\rho_1,1} \llbracket^{1}_{\rho_4} \rrbracket^{1}_{\rho_4} \rrbracket^{1}_{\rho_1,1} \llbracket^{1}_{\rho_1,2} \llbracket^{1}_{\rho_3} \widetilde{\mathrm{b}} \llbracket^{1}_{\rho_3,1} \llbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_3,1} \rrbracket^{1}_{\rho_3} \rrbracket^{1}_{\rho_1,2} \\
&\qquad \llbracket^{2}_{\rho_1,1} \llbracket^{2}_{\rho_4} \rrbracket^{2}_{\rho_4} \rrbracket^{2}_{\rho_1,1} \llbracket^{2}_{\rho_1,2} \llbracket^{2}_{\rho_3} \widetilde{\mathrm{d}} \llbracket^{2}_{\rho_3,1} \llbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_3,1} \rrbracket^{2}_{\rho_3} \rrbracket^{2}_{\rho_1,2} \Big\} \\
&\hfill \text{(because } \{\llbracket^{1}_{\rho_1}\} \in \mathfrak{P}_G) \\
&\equiv \Big\{ \llbracket^{1}_{\rho_4} \rrbracket^{1}_{\rho_4} \llbracket^{2}_{\rho_4} \rrbracket^{2}_{\rho_4}, \\
&\qquad \llbracket^{1}_{\rho_1,2} \llbracket^{1}_{\rho_3} \widetilde{\mathrm{b}} \llbracket^{1}_{\rho_3,1} \llbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_3,1} \rrbracket^{1}_{\rho_3} \rrbracket^{1}_{\rho_1,2} \\
&\qquad \llbracket^{2}_{\rho_1,2} \llbracket^{2}_{\rho_3} \widetilde{\mathrm{d}} \llbracket^{2}_{\rho_3,1} \llbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_3,1} \rrbracket^{2}_{\rho_3} \rrbracket^{2}_{\rho_1,2} \Big\} \\
&\hfill \text{(because } \{\llbracket^{1}_{\rho_1,1}, \llbracket^{2}_{\rho_1,1}\} \in \mathfrak{P}_G) \\
&\equiv \Big\{ \varepsilon, \quad \llbracket^{1}_{\rho_1,2} \llbracket^{1}_{\rho_3} \widetilde{\mathrm{b}} \llbracket^{1}_{\rho_3,1} \llbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_3,1} \rrbracket^{1}_{\rho_3} \rrbracket^{1}_{\rho_1,2} \\
&\qquad \llbracket^{2}_{\rho_1,2} \llbracket^{2}_{\rho_3} \widetilde{\mathrm{d}} \llbracket^{2}_{\rho_3,1} \llbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_3,1} \rrbracket^{2}_{\rho_3} \rrbracket^{2}_{\rho_1,2} \Big\} \\
&\hfill \text{(because } \{\llbracket^{1}_{\rho_4}, \llbracket^{2}_{\rho_4}\} \in \mathfrak{P}_G) \\
&\equiv \Big\{ \varepsilon, \ \llbracket^{1}_{\rho_3} \widetilde{\mathrm{b}} \llbracket^{1}_{\rho_3,1} \llbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_3,1} \rrbracket^{1}_{\rho_3} \llbracket^{2}_{\rho_3} \widetilde{\mathrm{d}} \llbracket^{2}_{\rho_3,1} \llbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_3,1} \rrbracket^{2}_{\rho_3} \Big\} \\
&\hfill \text{(because } \{\llbracket^{1}_{\rho_1,2}, \llbracket^{2}_{\rho_1,2}\} \in \mathfrak{P}_G) \\
&\equiv \Big\{ \varepsilon, \quad \widetilde{\mathrm{b}} \llbracket^{1}_{\rho_3,1} \llbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_3,1} \quad \widetilde{\mathrm{d}} \llbracket^{2}_{\rho_3,1} \llbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_3,1} \Big\} \\
&\hfill \text{(because } \{\llbracket^{1}_{\rho_3}, \llbracket^{2}_{\rho_3}\} \in \mathfrak{P}_G) \\
&\equiv \Big\{ \varepsilon, \quad \llbracket^{1}_{\rho_3,1} \llbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_3,1} \quad \widetilde{\mathrm{d}} \llbracket^{2}_{\rho_3,1} \llbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_3,1} \Big\} \\
&\hfill \text{(because } \{\llbracket_{\mathrm{b}}\} \in \mathfrak{P}_G \text{ and } \widetilde{\mathrm{b}} = \llbracket_{\mathrm{b}} \rrbracket_{\mathrm{b}}) \\
&\equiv \Big\{ \varepsilon, \quad \llbracket^{1}_{\rho_3,1} \llbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_3,1} \quad \llbracket^{2}_{\rho_3,1} \llbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_3,1} \Big\} \\
&\hfill \text{(because } \{\llbracket_{\mathrm{d}}\} \in \mathfrak{P}_G \text{ and } \widetilde{\mathrm{d}} = \llbracket_{\mathrm{d}} \rrbracket_{\mathrm{d}}) \\
&\equiv \Big\{ \varepsilon, \quad \llbracket^{1}_{\rho_5} \rrbracket^{1}_{\rho_5} \quad \llbracket^{2}_{\rho_5} \rrbracket^{2}_{\rho_5} \Big\} \\
&\hfill \text{(because } \{\llbracket^{1}_{\rho_3,1}, \llbracket^{2}_{\rho_3,1}\} \in \mathfrak{P}_G) \\
&\equiv \{\varepsilon\} \hfill \text{(because } \{\llbracket^{1}_{\rho_5}, \llbracket^{2}_{\rho_5}\} \in \mathfrak{P}_G) \quad \square
\end{aligned}
$$

A.2 Additional Proofs

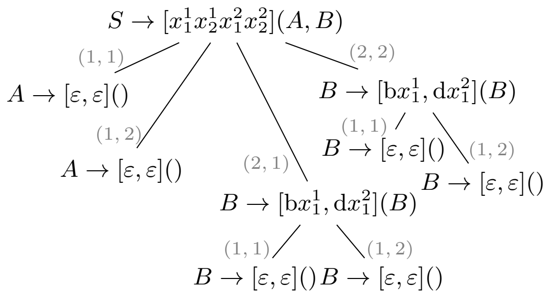
Lemma 12. There is a bijection toCowD between $\mathcal{L}(G_{\mathrm{cf}})$ and $\mathrm{cowD}^{\mathrm{c}}_{G}$.

Proof. For each $\ell \in \mathbb{N}_+$, we define the partial function f_ℓ from $(\Delta_G \cup \overline{\Delta_G})^*$ to ℓ-cow derivations of G as follows: $f_\ell(u) = \rho\langle (i(\kappa), j(\kappa))/f_{j(\kappa)}(v_\kappa) \mid \kappa \in [n]\rangle$ if u is of the



ρ_1
ρ_2
ρ_3
m times
n times
ρ_2
ρ_3
ρ_4
ρ_5
$[x_1^1 x_2^1 x_2^1 x_2^2]$
$[\mathrm{a}x_1^1, \mathrm{c}x_1^2]$
$[\mathrm{b}x_1^1, \mathrm{d}x_1^2]$
m times
n times
$[\mathrm{a}x_1^1, \mathrm{c}x_1^2]$
$[\mathrm{b}x_1^1, \mathrm{d}x_1^2]$
$[\varepsilon, \varepsilon]$
$[\varepsilon, \varepsilon]$

Figure 5: A derivation $d_{m,n}$ (top) together with the corresponding term over composition functions (bottom), cf. ex. 1.



$S \to [x_1^1 x_2^1 x_1^2 x_2^2](A, B)$
$(1,1)$
$(2,2)$
$A \to [\varepsilon, \varepsilon]()$
$B \to [\mathrm{b}x_1^1, \mathrm{d}x_1^2](B)$
$(1,2)$
$(1,1)$
$B \to [\varepsilon, \varepsilon]()$
$(1,2)$
$A \to [\varepsilon, \varepsilon]()$
$(2,1)$
$B \to [\varepsilon, \varepsilon]()$
$B \to [\mathrm{b}x_1^1, \mathrm{d}x_1^2](B)$
$(1,1)$
$(1,2)$
$B \to [\varepsilon, \varepsilon]()$
$B \to [\varepsilon, \varepsilon]()$

Figure 6: A consistent cow derivation.

form

$$\left\llbracket^{\ell}_{\rho} \widetilde{u_0} \llbracket^{j(1)}_{\rho,i(1)} v_1 \rrbracket^{j(1)}_{\rho,i(1)} \widetilde{u_1} \cdots \llbracket^{j(n)}_{\rho,i(n)} v_n \rrbracket^{j(n)}_{\rho,i(n)} \widetilde{u_n} \right\rrbracket^{\ell}_{\rho}$$

for some rule $\rho = A \to c(B_1, \dots, B_k)$ where the ℓ-th component of c is $u_0 x_{i(1)}^{j(1)} u_1 \cdots x_{i(n)}^{j(n)} u_n$ with $u_1, \dots, u_n \in \Sigma^*$; otherwise, $f_\ell(u)$ is undefined. Note that $f_1, f_2, \dots$ are pairwise disjoint (in the set-theoretic sense).

To prove that the function f_ℓ is bijective, we show that it is injective and surjective.

(Injectivity) We show, for each $\ell \in \mathbb{N}_+$, by induction on the structure of cow derivations that $f_\ell(v) = f_\ell(v')$ implies $v = v'$ for each v, v' in the domain of f_ℓ (i.e. $f_\ell(v)$ and $f_\ell(v')$ are both defined).

Let $v, v' \in (\Delta_G \cup \overline{\Delta_G})^*$ be in the domain of f_ℓ and let $t = f_\ell(v) = f_\ell(v')$. Furthermore, let $\rho = A \to c(B_1, \dots, B_k) = \mathrm{root}(t)$, $t_{(i,j)} = \mathrm{sub}_{(i,j)}(t)$ for each $(i,j) \in \mathrm{out}(t)$, and let the ℓ-th component of c be of the form $u_0 x_{i(1)}^{j(1)} u_1 \cdots x_{i(n)}^{j(n)} u_n$ with $u_1, \dots, u_n \in \Sigma^*$. By definition of f_ℓ, we know that u is the string

$$\left\llbracket^{\ell}_{\rho} \widetilde{u_0} \llbracket^{j(1)}_{\rho,i(1)} v_1 \rrbracket^{j(1)}_{\rho,i(1)} \widetilde{u_1} \cdots \llbracket^{j(n)}_{\rho,i(n)} v_n \rrbracket^{j(n)}_{\rho,i(n)} \widetilde{u_n} \right\rrbracket^{\ell}_{\rho}$$

for some $v_1, \dots, v_n \in (\Delta_G \cup \overline{\Delta_G})^*$, u' is the string

$$\left\llbracket^{\ell}_{\rho} \widetilde{u_0} \llbracket^{j(1)}_{\rho,i(1)} v_1' \rrbracket^{j(1)}_{\rho,i(1)} \widetilde{u_1} \cdots \llbracket^{j(n)}_{\rho,i(n)} v_n' \rrbracket^{j(n)}_{\rho,i(n)} \widetilde{u_n} \right\rrbracket^{\ell}_{\rho}$$

for some $v_1', \dots, v_n' \in (\Delta_G \cup \overline{\Delta_G})^*$, and $f_{j(\kappa)}(v_\kappa) = f_{j(\kappa)}(v_\kappa') = t_{(i(\kappa),j(\kappa))}$ for each $\kappa \in [n]$. By principle of induction, we get $v_\kappa = v_\kappa'$ for each $\kappa \in [n]$. Hence $v = v'$.

(Surjectivity) We show by induction on the structure of cow derivations that for each $A \in N$, $\ell \in \mathit{sort}(A)$, and $t \in \ell\text{-cowD}_G^A$, there is a string $v \in \mathcal{L}(G_{\mathrm{cf}}, A^\ell)$ such that $f_\ell(v) = t$.

Let $A \in N$, $\ell \in \mathit{sort}(A)$, and $t \in \ell\text{-cowD}_G^A$. Furthermore, let $\rho = A \to c(B_1, \dots, B_k) = \mathrm{root}(t)$, $t_{(i,j)} = \mathrm{sub}_{(i,j)}(t)$ for each $(i,j) \in \mathrm{out}(t)$, and let the ℓ-th component of c be of the form $u_0 x_{i(1)}^{j(1)} u_1 \cdots x_{i(n)}^{j(n)} u_n$ with $u_1, \dots, u_n \in \Sigma^*$. By principle of induction, we know that there are $v_\kappa \in \mathcal{L}(G_{\mathrm{cf}}, B_{i(\kappa)}^{j(\kappa)})$ with $f_{j(\kappa)}(v_\kappa) = t_{(i(\kappa),j(\kappa))}$ for each $\kappa \in [n]$. Now let v be the string

$$\left\llbracket^{\ell}_{\rho} \widetilde{u_0} \llbracket^{j(1)}_{\rho,i(1)} v_1 \rrbracket^{j(1)}_{\rho,i(1)} \widetilde{u_1} \cdots \llbracket^{j(n)}_{\rho,i(n)} v_n \rrbracket^{j(n)}_{\rho,i(n)} \widetilde{u_n} \right\rrbracket^{\ell}_{\rho}.$$

By definition of the rule $\rho^{(\ell)}$ (def. 8), we know that $v \in \mathcal{L}(G_{\mathrm{cf}}, A^\ell)$. By definition of f_ℓ, we know that $f_\ell(v) = t$. Hence, f_ℓ is surjective.

It is easy to see that $\mathrm{toCowD} = \bigcup_{\ell \in \mathbb{N}_+} f_\ell$. If we restrict the domain of toCowD to $\mathcal{L}(G_{\mathrm{cf}})$, then (since each element of $\mathcal{L}(G_{\mathrm{cf}})$ starts with a bracket of the form $\llbracket^1_\rho$ for some $\rho \in P$) the resulting function is a subset of f_1. Since f_1 is bijective, we know that toCowD is a bijection between $\mathcal{L}(G_{\mathrm{cf}})$ and $\mathrm{cowD}_G^{\mathrm{c}}$. ■

Lemma 16. Let $v \in \mathcal{L}(G_{\mathrm{cf}})$. Then

$$v \in \mathrm{mD}_G \iff \mathrm{toCowD}(v) \text{ is consistent.}$$

Proof. Let $s \in \mathbb{N}_+$ and let $t^1, \ldots, t^s$ be cow derivations in G. We abbreviate the following property by $\mathfrak{C}(t^1, \ldots, t^s)$: (i) $\{t^1, \ldots, t^s\}$ is consistent, (ii) $s = \mathrm{fanout}(\mathrm{root}(t^1))$, and (iii) t^ℓ is an ℓ-cow derivation for each $\ell \in [s]$. Now, we show by structural induction that $\mathfrak{C}(\mathrm{toCowD}(v^1), \ldots, \mathrm{toCowD}(v^s))$ holds iff each permutation of $v^1, \ldots, v^s$ is in mD_G. Let $\rho = A \to [c_1, \ldots, c_s](B_1, \ldots, B_k)$ be a rule in P. Let $v_i^j \in \mathcal{L}(G, B_i^j)$ for each $i \in [k]$ and $j \in [\mathit{sort}(B_i)]$ such that

$$\begin{aligned} &\mathrm{Per}(v_i^1, \ldots, v_i^{\mathit{sort}(B_i)}) \subseteq \mathrm{mD}_G \\ &\quad \iff \mathfrak{C}\big(t_i^1, \ldots, t_i^{\mathit{sort}(B_i)}\big) \end{aligned} \tag{IH}$$

for each $i \in [k]$ where t_i^j abbreviates $\mathrm{toCowD}(v_i^j)$ and $\mathrm{Per}(v_i^1, \ldots, v_i^{\mathit{sort}(B_i)})$ is the set of permutations of $v_i^1, \ldots, v_i^{\mathit{sort}(B_i)}$. For each $\ell \in [s]$, let v^ℓ be obtained from the right-hand side of $\rho^{(\ell)}$ by replacing each non-terminal of the form B_i^j by v_i^j. Clearly, for each $\ell \in [s]$, the sequence v^ℓ is an element of $\mathcal{L}(G, A^\ell)$. From now on, we will abbreviate $\mathrm{toCowD}(v^\ell)$ by t^ℓ for each $\ell \in [s]$. Now, let $B_{i(\ell,1)}^{j(\ell,1)}, \ldots, B_{i(\ell,n_\ell)}^{j(\ell,n_\ell)}$ be the nonterminals on the rhs of $\rho^{(\ell)}$, then t^ℓ is the cow derivation

$$\rho\big\langle (i(\ell,\kappa), j(\ell,\kappa))/t_{i(\ell,\kappa)}^{j(\ell,\kappa)} \quad \big| \quad \kappa \in [n_\ell] \big\rangle.$$

Note that ρ is the root symbol of each $t^1, \ldots, t^s$ and the set of indices $\{(i(\ell,\kappa), j(\ell,\kappa)) \mid \ell \in [s], \kappa \in [n_\ell]\}$ is the set $\{(i,j) \mid i \in [k], j \in [\mathit{sort}(B_i)]\}$. We derive

$$\begin{aligned} &\mathfrak{C}(t^1, \ldots, t^s) \\ &\iff \{t^1, \ldots, t^s\} \text{ is consistent} && \text{(by def. 10)} \\ &\iff \forall i \in [k]\colon \{t_i^1, \ldots, t_i^{\mathit{sort}(B_i)}\} \text{ is consistent} && \text{(by def. 14)} \\ &\iff \forall i \in [k]\colon \mathrm{Per}(v_i^1, \ldots, v_i^{\mathit{sort}(B_i)}) \subseteq \mathrm{mD}_G && \text{(by eq. (IH))} \\ &\iff \mathrm{Per}(v^1, \ldots, v^s) \subseteq \mathrm{mD}_G && \text{(by defs. 6 and 8)} \end{aligned}$$

Since the v_i^js and ρ were selected arbitrarily, we can obtain any element of $\mathcal{L}(G, A^\ell)$ in that manner. In particular, for each $v^1 \in \mathcal{L}(G_{\mathrm{cf}}, S^1) = \mathcal{L}(G_{\mathrm{cf}})$, the cow derivation $\mathrm{toCowD}(v^1)$ is consistent iff v^1 is in mD_G. ■

Proposition 15. TODERIV $\circ$ toCowD^{-1} is a bijection between the consistent cow derivations in $\mathrm{cowD}_G^{\mathrm{c}}$ and $\mathrm{D}_G^{\mathrm{c}}$.

Proof. By lems. 12 and 16, there is a bijection between the consistent cow derivations in $(1\text{-cowD}_G)_S$ and $\mathcal{R}_G \cap \mathrm{mD}_G$, and there is a bijection between $\mathcal{R}_G \cap \mathrm{mD}_G$ and $\mathrm{D}_G^{\mathrm{c}}$ (Denkinger, 2017, cor. 3.9). ■

A.3 Additional Algorithms

Algorithm 3 is a modification of the algorithm given by Hulden (2011, alg. 1). The changes involve an introduction of weights in the algorithm; elements of A are drawn by maximum weight instead of being drawn randomly. In our implementation, we defined the weight of an item (p, v, q) as the weight $\mu'(v)$ defined in def. 6.

Algorithm 3 extracts Dyck words from an FSA.

Input: a weight assignment $wt\colon Q \times (\Delta \cup \overline{\Delta})^* \times Q \to \mathcal{M}$, and an automaton $\mathcal{A} = (Q, \Delta \cup \overline{\Delta}, q_{\mathrm{init}}, q_{\mathrm{final}}, T)$

Output: a sequence $v_1, v_2, \ldots$ of elements in $\mathcal{L}(\mathcal{A}) \cap \mathrm{D}_\Delta$ such that $wt(q_{\mathrm{init}}, v_1, q_{\mathrm{final}}) \trianglerighteq wt(q_{\mathrm{init}}, v_2, q_{\mathrm{final}}) \trianglerighteq \ldots$

```
1: procedure EXTRACTDYCK($wt$)($\mathcal{A}$)
2:   $(A, C) \leftarrow (\varnothing, \varnothing)$
3:   for $(p, \delta, q), (q, \overline{\delta}, r) \in T$ do
4:     $A \leftarrow A \cup \{(p, \delta\overline{\delta}, r)\}$
5:   for $(p, v, q) \in \mathrm{argmax}_{\trianglelefteq, wt} A$ do
6:     if $(p, q) = (q_{\mathrm{init}}, q_{\mathrm{final}})$ then yield $v$
7:     $A \leftarrow A \setminus \{(p, v, q)\}$
8:     $C \leftarrow C \cup \{(p, v, q)\}$
9:     for $(r, \delta, p), (q, \overline{\delta}, s) \in T$ do
10:      $A \leftarrow A \cup \{(r, \delta v \overline{\delta}, s)\}$
11:    for $(q, w, r) \in C$ do
12:      $A \leftarrow A \cup \{(p, vw, r)\}$
13:    for $(o, u, p) \in C$ do
14:      $A \leftarrow A \cup \{(o, uv, q)\}$
```

Since the function given in alg. 4 is very similar to def. 14, we omit further discussions.

Algorithm 4 checks consistency of a set of cow derivations.

Input: a set T of cow derivations

Output: $\begin{Bmatrix} \text{true} & \text{if } T \text{ is consistent} \\ \text{false} & \text{otherwise} \end{Bmatrix}$

```
1: function ISCONSISTENT($T$)
2:   if not $\bigwedge_{t,t' \in T} \big(\mathrm{root}(t) = \mathrm{root}(t')\big)$ then
3:     return false
4:   $(T_1, \ldots, T_k) \leftarrow$ COLLECTCHILDREN($T$)
5:   return $\bigwedge_{i=1}^{k}$ ISCONSISTENT'($T_i$)
```

			candidate count			
			10^2	10^3	10^4	10^5
beam width	20	median parse time	8.04 ms	8.55 ms	8.52 ms	9.61 ms
		mean parse time	8.18 ms	11.48 ms	34.71 ms	256.08 ms
		f_1-score	67.83 %	68.44 %	68.45 %	68.80 %
		coverage (no fallback)	81.20 %	87.49 %	90.40 %	92.10 %
		coverage (with fallback)	97.31 %	97.31 %	97.31 %	97.31 %
	50	median parse time	13.23 ms	15.26 ms	15.44 ms	15.77 ms
		mean parse time	13.11 ms	17.99 ms	53.51 ms	299.63 ms
		f_1-score	70.54 %	71.82 %	72.17 %	72.46 %
		coverage (no fallback)	78.36 %	85.88 %	90.25 %	93.47 %
		coverage (with fallback)	98.31 %	98.31 %	98.31 %	98.31 %
	100	median parse time	19.68 ms	23.18 ms	24.43 ms	24.67 ms
		mean parse time	18.81 ms	25.11 ms	70.12 ms	374.90 ms
		f_1-score	70.27 %	71.43 %	71.99 %	72.86 %
		coverage (no fallback)	76.82 %	84.11 %	88.72 %	92.93 %
		coverage (with fallback)	98.93 %	98.93 %	98.93 %	98.93 %
	200	median parse time	31.33 ms	36.04 ms	**39.75 ms**	40.05 ms
		mean parse time	30.10 ms	35.80 ms	**85.94 ms**	412.16 ms
		f_1-score	70.50 %	71.50 %	**72.13 %**	72.98 %
		coverage (no fallback)	76.66 %	83.57 %	**88.33 %**	92.79 %
		coverage (with fallback)	99.00 %	99.00 %	**99.00 %**	99.00 %
	500	median parse time	53.16 ms	58.07 ms	68.22 ms	68.86 ms
		mean parse time	50.21 ms	56.70 sm	105.94 ms	431.90 ms
		f_1-score	70.51 %	71.45 %	72.08 %	72.90 %
		coverage (no fallback)	76.66 %	83.50 %	88.26 %	92.79 %
		coverage (with fallback)	99.00 %	99.00 %	99.00 %	99.00 %
	1000	median parse time	53.10 ms	58.59 ms	76.07 ms	77.37 ms
		mean parse time	53.53 ms	59.26 ms	109.46 ms	443.56 ms
		f_1-score	70.51 %	71.45 %	72.08 %	72.90 %
		coverage (no fallback)	76.66 %	83.50 %	88.26 %	92.79 %
		coverage (with fallback)	99.00 %	99.00 %	99.00 %	99.00 %

Table 2: Results of the grid search for meta-parameters.

A.4 Results of Grid Search for Meta-Parameters

Table 2 shows the results of the grid search for the two introduced meta-parameters. For each combination of *beam width* and *candidate count*, we list the *median* and *mean parse time*s (since medians hide outliers, those two may differ drastically) for all sentences of length 20 and f_1-*score* over all test sentences. Moreover, we show the percentage of sentences (*coverage*) that we were able to parse with and without the fallback mechanism. The results for the combination of meta-parameters that was selected for later experiments (i.e. a beam width of 200 and a candidate count of 10^4) are written in bold.