Chomsky-Schützenberger parsing for weighted multiple context-free languages

Tobias Denkinger
Faculty of Computer Science
Technische Universität Dresden
Germany

ABSTRACT

We prove a Chomsky-Schützenberger representation theorem for multiple context-free languages weighted over complete commutative strong bimonoids. Using this representation we devise a parsing algorithm for a restricted form of those devices.[1]

Keywords: Chomsky-Schützenberger, parsing, multiple context-free grammars, linear context-free rewriting systems

1 INTRODUCTION

Mildly context-sensitive languages receive much attention in the natural language processing community since they are able to express non-projective constituents (Maier and Søgaard 2008) and non-projective dependencies (Kuhlmann and Satta 2009). Figure 1 shows an example of a non-projective constituency tree. Figure 2 shows an example of a non-projective dependency tree. Non-projectivity is evident from crossings of edges, highlighted by circles. The phenomenon of non-projectivity occurs frequently in natural language corpora, e.g. about 28 percent of all sentences in both the NeGra corpus[2] and the TIGER corpus (Brants *et al.* 2004) contain non-projective constituents

[1] The CS parser for weighted MCFL (Sections 5 and 6) is original to this work. Sections 2 to 4 are a substantially revised version of Denkinger (2015).

[2] `http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/`

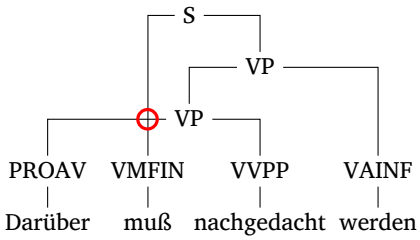

Figure 1:
A non-projective constituency tree (taken from Maier and Søgaard 2008, Figure 3)

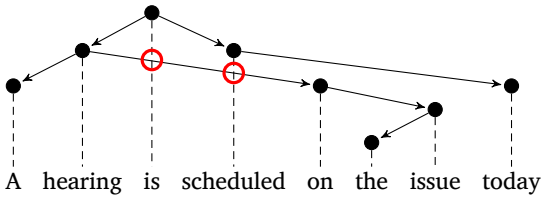

Figure 2:
A non-projective dependency tree (taken from Kuhlmann and Satta 2009, Figure 2)

(Maier and Søgaard 2008, Table 6) and about 23 percent of all sentences in the Prague Dependency Treebank[3] contain non-projective dependencies (Kuhlmann and Satta 2009, Section 1).

Multiple context-free grammars describe the language classes induced by many mildly context-sensitive grammar formalisms, e.g. head grammars (Seki *et al.* 1991), linear context-free rewriting systems (Seki *et al.* 1991), combinatory categorial grammars (Vijay-Shanker *et al.* 1986; Weir and Joshi 1988), linear indexed grammars (Vijay-Shanker 1988), minimalist grammars (Michaelis 2001b,a), and finite-copying lexical functional grammars (Seki *et al.* 1993).

Parsing, i.e. the annotation of a sentence with syntactic structure, is one of the main concerns of natural language processing. Many parsing approaches are known for multiple context-free grammars, e.g.

- Cocke-Younger-Kasami-style parsing (Seki *et al.* 1991, Procedure MEMBER; and Burden and Ljunglöf 2005, Section 3),
- guided parsing (Barthélemy *et al.* 2001; Burden and Ljunglöf 2005, Section 4; and van Cranenburgh 2012),
- active parsing (Burden and Ljunglöf 2005, Section 5),
- incremental parsing (Villemonte de la Clergerie 2002; Burden and Ljunglöf 2005, Section 6; and Angelov 2009), and
- LR-style parsing (Kallmeyer and Maier 2015).

[3] `https://ufal.mff.cuni.cz/pdt/Corpora/PDT_1.0/`

The Chomsky-Schützenberger (CS) representation for context-free languages (Chomsky and Schützenberger 1963, Proposition 2) has been generalised to a variety of unweighted and weighted settings, e.g. context-free languages weighted with commutative semirings (Salomaa and Soittola 1978, Theorem 4.5), tree adjoining languages (Weir 1988, Lemma 3.5.2), multiple context-free languages (Yoshinaka *et al.* 2010, Theorem 3), context-free languages weighted with unital valuation monoids (Droste and Vogler 2013, Theorem 2), yields of simple context-free tree languages (Kanazawa 2014, Theorem 8.3), indexed languages (Duske *et al.* 1979, Theorems 1 and 2; Fratani and Voundy 2015, Theorem 4; and Fratani and Voundy 2016, Theorem 18), and automata with storage weighted with unital valuation monoids (Herrmann and Vogler 2015, Theorem 11).

We give a generalisation to the case of multiple context-free languages weighted with a complete commutative strong bimonoid and apply it to devise a parsing algorithm. Sections 3 to 5 contain the main contributions of this article.

- In Section 3 we provide a CS representation for weighted multiple context-free languages by means of a modular proof that first separates the weights from the given grammar and then employs the result for the unweighted case (using the same overall idea as in Droste and Vogler 2013).
- In Section 4 we give a more algebraic variant of multiple Dyck languages using congruence relations together with a decision algorithm for membership that is strongly related to those congruence relations. Also we show that congruence multiple Dyck languages can be used to develop a CS representation of weighted MCFLs.
- Using the CS representation based on congruence multiple Dyck languages and given a partial order on the weights, we derive a parsing algorithm (Section 5) similar to the one described by Hulden (2011). It employs a regular language and a weight function to generate output that is then filtered by an acceptor for a specific congruence multiple Dyck language in order to obtain the best derivations with respect to the partial order.
- The idea behind the Chomsky-Schützenberger parser presented in Section 5 is similar to already established approaches. Section 6 relates the known approaches to the one presented in this article.

Since our proofs do not require distributivity, we can be slightly more general than complete commutative semirings: We consider complete commutative strong bimonoids.

2 PRELIMINARIES

In this section we recall formalisms used in this article and fix some notation: We denote by $\mathbb{N}$ the set of natural numbers (including zero), and by $\mathbb{N}_+$ the set $\mathbb{N} \setminus \{0\}$. For every $n \in \mathbb{N}$ we abbreviate $\{1, \dots, n\}$ by $[n]$. Let A be a set. The *power set of* A is denoted by $\mathscr{P}(A)$. Let B be a finite set. A *partition of* B is a set $\mathfrak{P} \subseteq \mathscr{P}(B)$ where the elements of $\mathfrak{P}$ (called *cells*) are non-empty, pairwise disjoint, and $\bigcup_{\mathfrak{p} \in \mathfrak{P}} \mathfrak{p} = B$.

Let A and B be sets and $A' \subseteq A$. The *set of functions from* A *to* B is denoted by $A \to B$, we still write $f : A \to B$ rather then $f \in A \to B$. Let f be a function. The *domain and codomain of* f are denoted by $\mathrm{dom}(f)$ and $\mathrm{codom}(f)$, respectively. The *restriction of* f *to* A', denoted by $f|_{A'}$, is a function from A' to B such that $f|_{A'}(a') = f(a')$ for every $a' \in A'$. Let g be a function such that $\mathrm{codom}(f) \subseteq \mathrm{dom}(g)$. We denote the function obtained by applying g *after* f by $g \circ f$. Let F be a set of functions and $B \subseteq \bigcap_{f \in F} \mathrm{dom}(f)$. The set $\{f(B) \mid f \in F\} \subseteq \mathscr{P}(\bigcup_{f \in F} \mathrm{codom}(f))$ is denoted by $F(B)$. Let G and H be sets of functions such that $\bigcap_{g \in G} \mathrm{codom}(g) \subseteq \bigcap_{h \in H} \mathrm{dom}(h)$. The set $\{h \circ g \mid h \in H, g \in G\}$ of functions is denoted by $H \circ G$.

Let A be a set and $\approx \subseteq A \times A$ a binary relation on A. We call $\approx$ an *equivalence relation (on A)* if it is reflexive, symmetric, and transitive. Let $a \in A$ and $\approx$ be an equivalence relation. The *equivalence class of* a *in* $\approx$, denoted by $[a]_\approx$, is $\{b \in A \mid a \approx b\}$. Let $f : A^k \to A$ be a function. We say that $\approx$ *respects* f if for every $(a_1, b_1), \dots, (a_k, b_k) \in \approx$ holds $f(a_1, \dots, a_k) \approx f(b_1, \dots, b_k)$. Now let $\mathscr{A}$ be an algebra with domain A. We call $\approx$ a *congruence relation (on* $\mathscr{A}$*)* if $\approx$ is an equivalence relation on A and respects every operation of $\mathscr{A}$.

Let $\trianglelefteq \subseteq A \times A$ be a binary relation on A. We call $\trianglelefteq$ a *partial order (on A)* if it is reflexive, antisymmetric, and transitive.

2.1 *Sorts*

We will use the concept of sorts to formalise restrictions on building terms (or trees), e.g. derivation trees or terms over functions. One can think of sorts as data types in a programming language: Every concrete

value has a sort (type) and every function requires its arguments to be of fixed sorts (types) and returns a value of some fixed sort (type).

Let S be a countable set (of *sorts*) and $s \in S$. An *S-sorted set* is a tuple (B, sort) where B is a set and sort is a function from B to S. We denote the preimage of s under sort by B_s and abbreviate (B, sort) by B; sort will always be clear from the context. Let A be an $(S^* \times S)$-sorted set. The *set of terms over A*, denoted by T_A, is the smallest S-sorted set T where $\xi = a(\xi_1, \ldots, \xi_k) \in T_s$ if there are $s_1, \ldots, s_k, s \in S$ such that $a \in A_{(s_1 \cdots s_k, s)}$ and $\xi_i \in T_{s_i}$ for every $i \in [k]$. Let $\xi = a(\xi_1, \ldots, \xi_k) \in T_A$. The *set of positions in* ξ is defined as $\mathrm{pos}(\xi) = \{\varepsilon\} \cup \{iu \mid i \in [k], u \in \mathrm{pos}(\xi_i)\}$ and for every $\pi \in \mathrm{pos}(\xi)$ the *symbol in ξ at position π* is defined as $\xi(\pi) = a$ if $\pi = \varepsilon$ and as $\xi(\pi) = \xi_i(u)$ if $\pi = iu$ for some $i \in [k]$ and $u \in \mathrm{pos}(\xi_i)$.

2.2 *Weight algebras*

A *monoid* is an algebra $(\mathscr{A}, \cdot, 1)$ where $\cdot$ is associative and 1 is neutral with respect to $\cdot$. A *bimonoid* is an algebra $(\mathscr{A}, +, \cdot, 0, 1)$ where $(\mathscr{A}, +, 0)$ and $(\mathscr{A}, \cdot, 1)$ are monoids. We call a bimonoid *strong* if $(\mathscr{A}, +, 0)$ is commutative and for every $a \in \mathscr{A}$ we have $0 \cdot a = 0 = a \cdot 0$. Intuitively, a strong bimonoid is a semiring without distributivity. A strong bimonoid is called *commutative* if $(\mathscr{A}, \cdot, 1)$ is commutative. A commutative strong bimonoid is *complete* if there is an infinitary sum operation $\sum$ that maps every indexed family of elements of $\mathscr{A}$ to $\mathscr{A}$, extends $+$, and satisfies infinitary associativity and commutativity laws (cf. Droste and Vogler 2013, Section 2), i.e. for every countable set I and every I-indexed family $a \colon I \to \mathscr{A}$ it holds that:

(i) $\sum_{i \in \varnothing} a(i) = 0$;

(ii) for every $j \in I$: $\sum_{i \in \{j\}} a(i) = a(j)$;

(iii) for every $j, k \in I$ with $j \neq k$: $\sum_{i \in \{j,k\}} a(i) = a(j) + a(k)$; and

(iv) for every countable set J and family $\mathscr{I} \colon J \to \mathscr{P}(I)$ with $I = \bigcup_{j \in J} \mathscr{I}(j)$ and for every $j, j' \in J$ with $j \neq j' \implies \mathscr{I}(j) \cap \mathscr{I}(j') = \varnothing$, we have $\sum_{j \in J} \sum_{i \in \mathscr{I}(j)} a(i) = \sum_{i \in I} a(i)$.

For the rest of this article let $(\mathscr{A}, +, \cdot, 0, 1)$, abbreviated by $\mathscr{A}$, be a complete commutative strong bimonoid.

Example 2.1. We provide a list of complete commutative strong bimonoids (cf. Droste *et al.* 2010, Example 1) some of which are relevant for natural language processing:

- any complete commutative semiring, e.g.
  - the *Boolean semiring* $\mathbb{B} = \big(\{0,1\}, \vee, \wedge, 0, 1\big)$,
  - the *probability semiring* $\Pr = \big(\mathbb{R}_{\geq 0}, +, \cdot, 0, 1\big)$,
  - the *Viterbi semiring* $\big([0,1], \max, \cdot, 0, 1\big)$,
  - the *tropical semiring* $\big(\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0\big)$,
  - the *arctic semiring* $\big(\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0\big)$,
- any complete lattice, e.g.
  - any non-empty finite lattice $\big(L, \vee, \wedge, 0, 1\big)$ where L is a non-empty finite set,
  - the lattice $(\mathscr{P}(A), \cup, \cap, \varnothing, A)$ where A is an arbitrary set,
  - the lattice $(\mathbb{N}, \operatorname{lcm}, \gcd, 1, 0)$,
- the *tropical bimonoid* $\big(\mathbb{R}_{\geq 0} \cup \{\infty\}, +, \min, 0, \infty\big)$,
- the *arctic bimonoid* $\big(\mathbb{R}_{\geq 0} \cup \{-\infty\}, +, \max, 0, -\infty\big)$, and
- the algebras $\Pr_1 = ([0,1], \oplus_1, \cdot, 0, 1)$ and $\Pr_2 = ([0,1], \oplus_2, \cdot, 0, 1)$ where $a \oplus_1 b = a + b - a \cdot b$ and $a \oplus_2 b = \min\{a + b, 1\}$ for every $a, b \in [0,1]$.

where $\mathbb{R}$ and $\mathbb{R}_{\geq 0}$ denote the set of reals and the set of non-negative reals, respectively; $+$, $\cdot$, $\max$, $\min$ denote the usual operations; $\vee$, $\wedge$ denote disjunction and conjunction, respectively, for the boolean semiring and join and meet, respectively, for any non-empty finite lattice; and lcm and $\gcd$ are binary functions that calculate the least common multiple and the greatest common divisor, respectively.

Also, there are some bimonoids that are interesting for natural language processing but are *not* complete commutative strong bimonoids. E.g.

- the *semiring of formal languages* $\big(\mathscr{P}(\Sigma^*), \cup, \cdot, \varnothing, \{\varepsilon\}\big)$ where Σ is an alphabet and $\cdot$ is language concatenation, i.e. $L_1 \cdot L_2 = \{uv \mid u \in L_1, v \in L_2\}$ for every $L_1, L_2 \subseteq \Sigma^*$; and
- the semiring $\big(\Sigma^* \cup \{\infty\}, \wedge, \cdot, \infty, \varepsilon\big)$ where Σ is an alphabet, $\cdot$ is concatenation, $\wedge$ calculates the longest common prefix of its arguments, and ∞ is a new element that is neutral with respect to $\wedge$ and annihilating with respect to $\cdot$ (cf. Mohri 2000).

None of the two examples is commutative. □

An *$\mathscr{A}$-weighted language (over Δ)* is a function $L\colon \Delta^* \to \mathscr{A}$. The *support of* L, denoted by $\mathrm{supp}(L)$, is $\{w \in \Delta^* \mid L(w) \neq 0\}$. If $|\mathrm{supp}(L)| \leq 1$, we call L a *monomial*. We write $\mu.w$ for L if $L(w) = \mu$ and for every $w' \in \Delta^* \setminus \{w\}$ we have $L(w') = 0$.

2.3 *Recognisable languages*

For the many known results concerning finite state automata and regular languages, we will rely on Hopcroft and Ullman (1969) and Hopcroft and Ullman (1979). We nevertheless recall the basic definitions:

Definition 2.2. A *finite state automaton*, for short: FSA, is a tuple $\mathscr{M} = (Q, \Delta, q_0, F, T)$ where Δ is an alphabet (*terminals*), Q is a finite set (*states*) disjoint from Δ, $q_0 \in Q$ (*initial state*), $F \subseteq Q$ (*final states*), and $T \subseteq Q \times \Delta^* \times Q$ is a finite set (*transitions*). □

A *run in* $\mathscr{M}$ is a string $\kappa \in (Q \cup \Delta)^*$ where for every substring of κ of the form quq' (for some $q, q' \in Q$ and $u \in \Delta^*$) we have that $(q, u, q') \in T$, the first symbol of κ is q_0, and the last symbol of κ is in F. If a run κ contains a substring of the form quq' (for some $q, q' \in Q$ and $u \in \Delta^*$), we say that the transition (q, u, q') *occurs in* κ. The *word corresponding to* κ is obtained by removing the elements of Q from κ. The *language of* $\mathscr{M}$ is denoted by $L(\mathscr{M})$. The *set of recognisable languages*, denoted by REC, is the set of languages L for which there is an FSA $\mathscr{M}$ with $L = L(\mathscr{M})$.

2.4 *Weighted string homomorphisms*

Let Δ and Γ be alphabets and $g\colon \Delta \to (\Gamma^* \to \mathscr{A})$ (i.e. a function that takes an element of Δ and returns a function that takes an element of Γ^* and returns an element of $\mathscr{A}$) such that $g(\delta)$ is a monomial for every $\delta \in \Delta$. We define $\widehat{g}\colon \Delta^* \to (\Gamma^* \to \mathscr{A})$ where for every $k \in \mathbb{N}$, $\delta_1, \ldots, \delta_k \in \Delta$, and $v \in \Gamma^*$ we have

$$\widehat{g}(\delta_1 \cdots \delta_k)(v) = \sum_{\substack{v_1, \ldots, v_k \in \Gamma^* \\ v = v_1 \cdots v_k}} g(\delta_1)(v_1) \cdot \ldots \cdot g(\delta_k)(v_k).$$

We call $\widehat{g}$ an *$\mathscr{A}$-weighted (string) homomorphism*. Since $g(\delta_1), \ldots, g(\delta_k)$ are monomials for each $\delta_1, \ldots, \delta_k \in \Delta$, there is at most one tuple $(v_1, \ldots, v_k) \in (\Gamma^*)^k$ such that $g(\delta_i)(v_i) \neq 0$ for every $i \in [k]$. Hence, there is at most one $v \in \Gamma^*$ with $\widehat{g}(\delta_1 \cdots \delta_k)(v) \neq 0$ (namely $v = v_1 \cdots v_k$). Therefore, $\widehat{g}(u)$ is a monomial for every $u \in \Delta^*$. We call $\widehat{g}$

alphabetic if there is a function $h\colon \Delta \to (\Gamma \cup \{\varepsilon\} \to \mathcal{A})$ with $\widehat{g} = \widehat{h}$. If $\widehat{g}(u) = \mu.w$ (recall the end of Section 2.2) for $u \in \Delta^*$, then we will sometimes say "$\widehat{g}$ maps u to w" (leaving out the weight μ) or "$\widehat{g}$ weights u with μ" (leaving out the word w). Now assume that $\mathcal{A} = \mathbb{B}$ and we have $|\mathrm{supp}(g(\delta))| = 1$ for every $\delta \in \Delta$. Then g can be construed as a function from Δ to Γ^* and $\widehat{g}$ can be construed as a function from Δ^* to Γ^*. In this case we call $\widehat{g}$ a *(string) homomorphism*. The sets of all $\mathcal{A}$-weighted homomorphisms, $\mathcal{A}$-weighted alphabetic homomorphisms, homomorphisms, and alphabetic homomorphisms are denoted by $\mathrm{HOM}(\mathcal{A})$, $\alpha\mathrm{HOM}(\mathcal{A})$, HOM, and αHOM, respectively.

2.5 *Weighted multiple context-free languages*

We fix a set $X = \{x_i^j \mid i, j \in \mathbb{N}_+\}$ of *variables*. Let Δ be an alphabet. The set of *composition representations over* Δ is the $(\mathbb{N}^* \times \mathbb{N})$-sorted set RF_Δ where for every $s_1, \dots, s_\ell, s \in \mathbb{N}$ we define $X_{(s_1 \cdots s_\ell, s)} = \{x_i^j \mid i \in [\ell], j \in [s_i]\} \subseteq X$ and $(\mathrm{RF}_\Delta)_{(s_1 \cdots s_\ell, s)}$ as the set that contains $[u_1, \dots, u_s]_{(s_1 \cdots s_\ell, s)}$ for every $u_1, \dots, u_s \in (\Delta \cup X_{(s_1 \cdots s_\ell, s)})^*$. We will often write X_f instead of $X_{(s_1 \cdots s_\ell, s)}$. Let $f = [u_1, \dots, u_s]_{(s_1 \cdots s_\ell, s)} \in \mathrm{RF}_\Delta$. The *string function of* f, also denoted by f, is the function from $(\Delta^*)^{s_1} \times \cdots \times (\Delta^*)^{s_\ell}$ to $(\Delta^*)^s$ such that $f((w_1^1, \dots, w_1^{s_1}), \dots, (w_\ell^1, \dots, w_\ell^{s_\ell})) = (u_1', \dots, u_s')$ where $(u_1', \dots, u_s')$ is obtained from $(u_1, \dots, u_s)$ by replacing each occurrence of x_i^j by w_i^j for every $i \in [\ell]$ and $j \in [s_\ell]$. The set of all string functions for some composition representation over Δ is denoted by F_Δ. From here on we no longer distinguish between composition representations and string functions. We define the *rank of* f, denoted by $\mathrm{rank}(f)$, and the *fan-out of* f, denoted by $\text{fan-out}(f)$, as ℓ and s, respectively. Also, we will denote s_i by $\text{fan-out}_i(f)$ for every $i \in [\ell]$. The string function f is called *linear* if in $u_1 \cdots u_s$ every element of X_f occurs at most once, f is called *non-deleting* if in $u_1 \cdots u_s$ every element of X_f occurs at least once, and f is called *terminal-free* if $u_1, \dots, u_s \in X_f^*$. The subscript is dropped from the string function if its sort is clear from the context.

Note that for every $s' \in \mathbb{N}^* \times \mathbb{N}$, the set of linear terminal-free string functions of sort s' is finite.

Definition 2.3. A *multiple context-free grammar (over Δ)*, for short: MCFG, is a tuple (N, Δ, S, P) where N is a finite $\mathbb{N}$-sorted set (*non-terminals*), $S \in N_1$ (*initial non-terminal*), and $P \subseteq_{\mathrm{fin}} \big\{(A, f, A_1 \cdots A_\ell) \in N \times F_\Delta \times N^\ell \mid \mathrm{sort}(f) = (\mathrm{sort}(A_1) \cdots \mathrm{sort}(A_\ell), \mathrm{sort}(A)), f \text{ is linear}, \ell \in \mathbb{N}\big\}$

(*productions*). We construe P as an $(N^* \times N)$-sorted set where for every $\rho = (A, f, A_1 \cdots A_\ell) \in P$ we have $\mathrm{sort}(\rho) = (A_1 \cdots A_\ell, A)$. □

Let $G = (N, \Delta, S, P)$ be an MCFG and $w \in \Delta^*$. A production $(A, f, A_1 \cdots A_\ell) \in P$ is usually written as $A \to f(A_1, \ldots, A_\ell)$; it inherits its rank, fan-out, and fan-out$_1, \ldots,$ fan-out$_\ell$ from f. Also, $\mathrm{rank}(G) = \max_{\rho \in P} \mathrm{rank}(\rho)$ and $\text{fan-out}(G) = \max_{\rho \in P} \text{fan-out}(\rho)$. MCFGs of fan-out at most k and rank at most r are called (k, r)-*MCFGs*.

The function yield: $T_P \to \bigcup_{s=0}^{\max_{\rho \in P} \text{fan-out}(\rho)} (\Delta^*)^s$ assigns to every tree $d \in T_P$ the tuple obtained by projecting every production in d to the contained function (i.e. the second component) and then evaluating the resulting term over F_Δ.

Let $A \in N$. The *set of subderivations in G from A*, denoted by $D_G(A)$, is the set of all terms over P with sort A, i.e. $D_G(A) = (T_P)_A$. The *set of derivations in G* is $D_G = D_G(S)$. Let $w \in \Delta^*$. The *set of derivations of w in G* is $D_G(w) = \{d \in D_G \mid \mathrm{yield}(d) = w\}$.[4]

The *language of G* is $L(G) = \{w \in \Delta^* \mid D_G(w) \neq \varnothing\}$. A language L is called *multiple context-free* if there is an MCFG G with $L = L(G)$. The *set of multiple context-free languages (for which a (k, r)-MCFG exists)* is denoted by MCFL ((k, r)-MCFL, respectively).

The language class (k, r)-MCFL is a substitution-closed full abstract family of languages (Seki *et al.* 1991, Theorem 3.9). Thus (k, r)-MCFL is closed under homomorphisms and under intersection with regular languages.

Definition 2.4. An *$\mathscr{A}$-weighted MCFG (over Δ)* is a tuple (N, Δ, S, P, μ) where (N, Δ, S, P) is an MCFG and $\mu: P \to \mathscr{A} \setminus \{0\}$ (*weight assignment*). □

Let $G = (N, \Delta, S, P, \mu)$ be an $\mathscr{A}$-weighted MCFG and $w \in \Delta^*$. The *set of derivations of w in G* is the set of derivations of w in (N, Δ, S, P). G inherits fan-out and rank from (N, Δ, S, P); $\mathscr{A}$-weighted MCFGs of fan-out at most k and rank at most r are called *$\mathscr{A}$-weighted (k, r)-MCFGs*. We define a function $\widehat{\mu}: D_G \to \mathscr{A}$ that applies μ at every position of a given derivation and then multiplies the resulting values (in any order, since $\cdot$ is commutative). The *$\mathscr{A}$-weighted language induced by G* is the function $[\![G]\!]: \Delta^* \to \mathscr{A}$ where for every $w \in \Delta^*$ we have $[\![G]\!](w) = \sum_{d \in D_G(w)} \widehat{\mu}(d)$.

[4] We identify the 1-tuple containing a word $w \in \Delta^*$ with the word w itself.

Two ($\mathscr{A}$-weighted) MCFGs are *equivalent* if they induce the same ($\mathscr{A}$-weighted) language. An $\mathscr{A}$-weighted language L is called *multiple context-free* if there is an $\mathscr{A}$-weighted MCFG G such that $L = [\![G]\!]$; (k,r)-MCFL($\mathscr{A}$) denotes the set of languages induced by multiple context-free $\mathscr{A}$-weighted grammars of fan-out at most k and rank at most r.

Example 2.5. Consider the Pr_2-weighted MCFG $G = \big(N, \Delta, S, P, \mu\big)$ where $N_1 = \{S\}$, $N_2 = \{A, B\}$, $N = N_1 \cup N_2$, $\Delta = \{a, b, c, d\}$, and P and μ are given by

$$\begin{aligned} P\colon\ & \rho_1 = S \to [x_1^1 x_2^1 x_1^2 x_2^2](A, B) && \mu\colon\ \mu(\rho_1) = 1 \\ & \rho_2 = A \to [a x_1^1, c x_1^2](A) && \mu(\rho_2) = 1/2 \\ & \rho_3 = B \to [b x_1^1, d x_1^2](B) && \mu(\rho_3) = 1/3 \\ & \rho_4 = A \to [\varepsilon, \varepsilon]() && \mu(\rho_4) = 1/2 \\ & \rho_5 = B \to [\varepsilon, \varepsilon]() && \mu(\rho_5) = 2/3\,. \end{aligned}$$

Note that G has fan-out 2 and rank 2. We observe that $\mathrm{supp}([\![G]\!]) = \{a^m b^n c^m d^n \mid m, n \in \mathbb{N}\}$ and for every $m, n \in \mathbb{N}$ we have

$$\begin{aligned} [\![G]\!](a^m b^n c^m d^n) &= \mu(\rho_1) \cdot \big(\mu(\rho_2)\big)^m \cdot \mu(\rho_4) \cdot \big(\mu(\rho_3)\big)^m \cdot \mu(\rho_5) \\ &= 1/(2^m \cdot 3^{n+1}). \end{aligned}$$

The derivation d of $w = ac$ and the derivation $\bar{d}$ of $\bar{w} = aabccd$ in G are shown in Figure 3, their weights are $1/(2^1 \cdot 3^{0+1}) = 1/6$ and $1/(2^2 \cdot 3^{1+1}) = 1/36$, respectively. Since d and $\bar{d}$ are unique derivations for w and $\bar{w}$, we have $[\![G]\!](w) = 1/6$ and $[\![G]\!](\bar{w}) = 1/36$. □

A non-terminal is called *productive in an ($\mathscr{A}$-weighted) MCFG* if there is at least one subderivation starting from this non-terminal. It is obvious that every ($\mathscr{A}$-weighted) (k,r)-MCFL can be recognised by an ($\mathscr{A}$-weighted) (k,r)-MCFG that only has productive non-terminals.

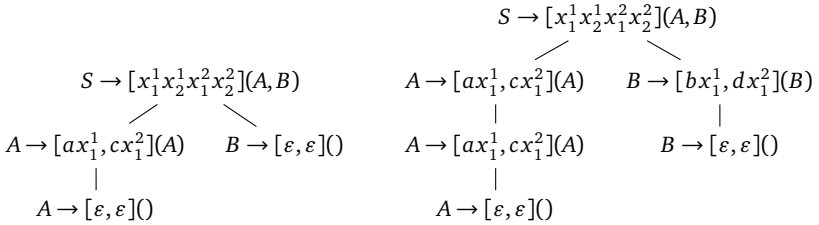

Figure 3: Derivation d of ac (left) and derivation $\bar{d}$ of $aabccd$ (right) in G (Example 2.5)

Non-deleting normal form

An ($\mathscr{A}$-weighted) MCFG is called *non-deleting* if the string function in every production is linear and non-deleting. Non-deleting MCFGs are also called linear context-free rewriting systems (Vijay-Shanker *et al.* 1987) in the literature. Seki *et al.* (1991, Lemma 2.2) proved that for every (k,r)-MCFG there is an equivalent non-deleting (k,r)-MCFG. We generalise this to $\mathscr{A}$-weighted MCFGs.

Lemma 2.6. For every $\mathscr{A}$-weighted (k,r)-MCFG there is an equivalent non-deleting $\mathscr{A}$-weighted (k,r)-MCFG.

Proof idea. We modify the construction for the unweighted case (Seki *et al.* 1991, Lemma 2.2) such that it preserves the structure of derivations. Then a weight assignment can be defined in an obvious manner.

Proof. Let $G = (N, \Delta, S, P, \mu)$ be an $\mathscr{A}$-weighted (k,r)-MCFG. We construct the (k,r)-MCFG $G' = (N', \Delta, S\langle\varnothing\rangle, P')$ where $N' = \{A\langle\Psi\rangle \mid A \in N, \Psi \subseteq [\mathrm{sort}(A)]\}$, $P' = \{\rho_\Psi \mid \rho \in P, \Psi \subseteq [\text{fan-out}(\rho)]\}$, and $\rho_\Psi = A\langle\Psi\rangle \to [u_{j_1}, \ldots, u_{j_\ell}](A_1\langle\Psi_1\rangle, \ldots, A_k\langle\Psi_k\rangle)$ for every $\rho = A \to [u_1, \ldots, u_m](A_1, \ldots, A_k) \in P$ and $\Psi \subseteq [\mathrm{sort}(A)]$ such that

(i) $\{j_1, \ldots, j_\ell\} = [\mathrm{sort}(A)] \setminus \Psi$ with $j_1 < \cdots < j_\ell$ and
(ii) $\Psi_i = \{j \subseteq [\mathrm{sort}(A_i)] \mid x_i^j$ does not occur in $u_{j_1} \cdots u_{j_\ell}\}$ for each $i \in [k]$.

The construction of G' here is a slight modification of the original construction (Lemma 2.2 in Seki *et al.* 1991, step 2 of Procedure 1) where we dropped the restrictions that $\Psi \neq [\mathrm{sort}(A)]$ and $\Psi \neq [\text{fan-out}(\rho)]$ in the definitions of N' and P', respectively.[5] Note that for each ρ and Ψ, the sets $\Psi_1, \ldots, \Psi_k$ are uniquely defined. Let $g\colon P' \to P$ such that $g(\rho_\Psi) = \rho$ and $\widehat{g}\colon D_{G'} \to D_G$ be the function obtained by applying g point-wise. We show the following hypothesis by induction on the structure of subderivations:
Induction hypothesis: For every $A \in N$ and $\Psi \subseteq [\mathrm{sort}(A)]$: $\widehat{g}$ is a bijection between $D_{G'}(A\langle\Psi\rangle)$ and $D_G(A)$.
Induction step: Let $d \in D_G(A)$ and $\Psi \subseteq [\mathrm{sort}(A)]$ with $d = \rho(d_1, \ldots, d_k)$ for some production $\rho \in P$ and derivations $d_1 \in D_G(A_1)$, …, $d_k \in D_G(A_k)$. By construction, $\Psi_1 \subseteq [\mathrm{sort}(A_1)]$, …, $\Psi_k \subseteq [\mathrm{sort}(A_k)]$ and therefore

[5] This construction may therefore create productions of fan-out 0.

ρ_Ψ are uniquely defined for every ρ and Ψ. By the induction hypothesis, we know that there are derivations $d'_1, \dots, d'_k$ which are unique for $(d_1, \Psi_1), \dots, (d_k, \Psi_k)$, respectively. Therefore, $d' = \rho(d'_1, \dots, d'_k)$ is unique for d and Ψ. Hence for every Ψ, $\widehat{g}$ is a bijection between $D_{G'}(A\langle\Psi\rangle)$ and $D_G(A)$.

By construction, the new start symbol is $S\langle\varnothing\rangle$; hence for the elements of $D_{G'}$, we set $\Psi = \varnothing$ and by induction hypothesis we obtain that $\widehat{g}$ is bijective. Since $\widehat{g}$ preserves the structure of derivations and is a bijection, we obtain $\widehat{\mu \circ g} = \widehat{\mu} \circ \widehat{g}$. Hence $[\![(N', \Delta, S\langle\varnothing\rangle, P', \mu \circ g)]\!] = [\![G]\!]$. Fan-out and rank are not increased by this construction. ∎

3 CS CHARACTERISATION FOR WEIGHTED MCFLS

In this section we generalise the CS characterisation of (unweighted) MCFLs (Yoshinaka *et al.* 2010, Theorem 3) to the weighted case. We prove that an $\mathscr{A}$-weighted MCFL L can be decomposed into an $\mathscr{A}$-weighted alphabetic homomorphism h, a regular language R and a multiple Dyck language mD such that $L = h(R \cap mD)$.

To show this, we use the proof idea from Droste and Vogler (2013): We separate the weight from our grammar formalism and then use the unweighted CS representation on the unweighted part. The outline of our proof is as follows:

(i) We separate the weights from L (Lemma 3.3), obtaining an MCFL L' and a weighted alphabetic homomorphism.

(ii) We use a corollary of the CS representation of (unweighted) MCFLs (Corollary 3.8) to obtain a CS representation of L'.

(iii) Using the two previous items and Lemma 3.10 for the composition of weighted and unweighted alphabetic homomorphisms, we obtain a CS representation of L (Theorem 3.12).

Figure 4 outlines the proof of Theorem 3.12. The boxes represent sub-diagrams for which the corresponding lemmas prove existence of the arrows and that the sub-diagram commutes.

3.1 *Separating the weights*

We split a given weighted MCFG G into an unweighted MCFG $G_{\mathbb{B}}$ and a weighted alphabetic homomorphism $\mathrm{weights}_G$ such that $[\![G]\!](w) = \sum_{u \in L(G_{\mathbb{B}})} \mathrm{weights}_G(u)(w)$ for every $w \in \Delta^*$.

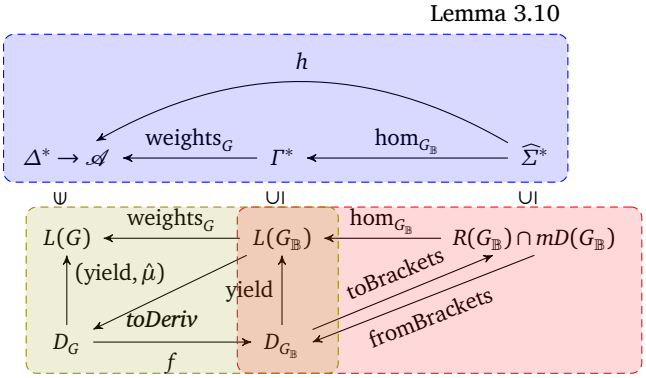

Figure 4:
Outline of the proof of Theorem 3.12

Definition 3.1. Let $G = (N, \Delta, S, P, \mu)$ be a non-deleting $\mathscr{A}$-weighted k-MCFG. The *unweighted MCFG for* G is the non-deleting k-MCFG $G_{\mathbb{B}} = (N, \Gamma, S, P_{\mathbb{B}})$ where $\Gamma = \Delta \cup \{\rho^i \mid \rho \in P, i \in [\text{fan-out}(\rho)]\}$ and $P_{\mathbb{B}}$ is the smallest set P' such that for every production $\rho = A \to [u_1, \ldots, u_s](A_1, \ldots, A_m) \in P$ there is a production

$$A \to [\rho^1 u_1, \ldots, \rho^s u_s](A_1, \ldots, A_m) \in P'. \qquad \square$$

Definition 3.2. Let $G = (N, \Delta, S, P, \mu)$ be a non-deleting $\mathscr{A}$-weighted MCFG. The *weight homomorphism for* G is the $\mathscr{A}$-weighted alphabetic homomorphism $\text{weights}_G \colon \Gamma^* \to (\Delta^* \to \mathscr{A})$ where $\text{weights}_G(\delta) = 1.\delta$, $\text{weights}_G(\rho^1) = \mu(\rho).\varepsilon$, and $\text{weights}_G(\rho^i) = 1.\varepsilon$ for every $\delta \in \Delta$, $\rho \in P$ and $i \in \{2, \ldots, \text{fan-out}(\rho)\}$. $\square$

$L(G_{\mathbb{B}})$ stands in bijection to D_G via the function *toDeriv* given in Algorithm 1.

Lemma 3.3. $(k,r)\text{-MCFL}(\mathscr{A}) = \alpha\text{HOM}(\mathscr{A})\big((k,r)\text{-MCFL}\big)$

Proof. **($\subseteq$)** Let $L \in (k,r)\text{-MCFL}(\mathscr{A})$. By Lemma 2.6 there is a non-deleting $\mathscr{A}$-weighted (k,r)-MCFG $G = (N, \Delta, S, P, \mu)$ such that $[\![G]\!] = L$. Let $f : P \to P_{\mathbb{B}}$ where for every $\rho = A \to [u_1, \ldots, u_s](A_1, \ldots, A_m) \in P$ we have $f(\rho) = A \to [\rho^1 u_1, \ldots, \rho^s u_s](A_1, \ldots, A_m)$, in other words f represents the construction of rules in $P_{\mathbb{B}}$ from the corresponding rules in P (see Definition 3.1). We extend f to $\widehat{f} \colon D_G \to D_{G_{\mathbb{B}}}$ by position-wise application, i.e. $\widehat{f}(d) = (f(\rho))(\widehat{f}(d_1), \ldots, \widehat{f}(d_k))$ for every subderivation $d = \rho(d_1, \ldots, d_k)$ in G; and we write f instead of $\widehat{f}$. For every

Algorithm 1: Function *toDeriv* to calculate for every word in $L(G_{\mathbb{B}})$ the corresponding derivation in D_G, cf. Lemma 3.3

Input: $w \in L(G_{\mathbb{B}})$
Output: derivation tree $t \in D_G$ corresponding to w (represented as a partial function from $\mathbb{N}^*$ to P)

```
1  function toDeriv(w)
2    let t be the empty function
3    descend(t, ε, 1)
4    return t
5  end function

6  procedure descend(t: ℕ* → P, π ∈ ℕ*, j ∈ ℕ)
7    let ρ = A → [u_1, ..., u_s](A_1, ..., A_k) ∈ P and u such that ρ^j u = w
8    add the assignment π ↦ ρ to t
9    remove ρ^j from the beginning of w
10   for every symbol δ' in u_j do
11     if δ' ∈ Δ then
12       remove δ' from the beginning of w
13     else
14       let i, j' such that x_i^{j'} = δ'
15       descend(t, πi, j')
16     end if
17   end for
18 end procedure
```

$w \in L(G_{\mathbb{B}})$ we can calculate the corresponding derivation t in G (as a function with domain $\mathrm{dom}(t)$ and labelling function t) using *toDeriv* (Algorithm 1), hence $\mathrm{yield} \circ f$ is bijective. We derive for every $w \in \Delta^*$:

$$\begin{aligned}
L(w) &= [\![G]\!](w) \\
&= \textstyle\sum_{d \in D_G(w)} \mu(d) \\
&= \textstyle\sum_{d \in D_G} (\mathrm{weights}_G \circ \mathrm{yield} \circ f)(d)(w) && \text{(by †)} \\
&= \textstyle\sum_{\substack{d \in D_G, u \in L(G_{\mathbb{B}}) \\ u = (\mathrm{yield} \circ f)(d)}} \mathrm{weights}_G(u)(w) \\
&= \textstyle\sum_{u \in L(G_{\mathbb{B}})} \mathrm{weights}_G(u)(w) && (L(G_{\mathbb{B}}) \text{ and } D_G \text{ are in bijection}) \\
&= \mathrm{weights}_G(L(G_{\mathbb{B}}))(w)
\end{aligned}$$

For †, one can immediately see from the definitions of f, yield, and $\mathrm{weights}_G$ that for every $w \in \Delta^*$ we have $(\mathrm{weights}_G \circ \mathrm{yield} \circ f)(d)(w) =$

$\mu(d)$ if $d \in D_G(w)$ and $(\text{weights}_G \circ \text{yield} \circ f)(d)(w) = 0$ otherwise. Hence $L = \text{weights}_G(L(G_{\mathbb{B}}))$.

($\supseteq$) Let $L \in (k,r)$-MCFL and $h\colon \Gamma^* \to (\Delta^* \to \mathcal{A})$ be an $\mathcal{A}$-weighted alphabetic homomorphism. By Seki *et al.* (1991, Lemma 2.2) there is a non-deleting (k,r)-MCFG $G = (N, \Gamma, S, P)$ such that $L(G) = L$. We construct the $\mathcal{A}$-weighted (k,r)-MCFG $G' = (N, \Delta, S, P', \mu)$ as follows: We extend h to $h'\colon (\Gamma \cup X)^* \to ((\Delta \cup X)^* \to \mathcal{A})$ where $h'(x) = 1.x$ for every $x \in X$ and $h'(\gamma) = h(\gamma)$ for every $\gamma \in \Gamma$. We define P' as the smallest set such that for every $\rho = A \to [u_1, \ldots, u_s](A_1, \ldots, A_m) \in P_{(s_1 \cdots s_m, s)}$ and $(u'_1, \ldots, u'_s) \in \text{supp}(h'(u_1)) \times \ldots \times \text{supp}(h'(u_s))$ we have that P' contains the production $\rho' = A \to [u'_1, \ldots, u'_s](A_1, \ldots, A_m)$ and $\mu(\rho') = h'(u_1)(u'_1) \cdot \ldots \cdot h'(u_s)(u'_s)$. Since $\cdot$ is commutative and G is non-deleting, we obtain $[\![G']\!] = h(L(G))$. ■

By setting $k = 1$ in the above lemma we reobtain the equivalence of 1 and 3 in Theorem 2 of Droste and Vogler (2013) for the case of complete commutative strong bimonoids.

Example 3.4. Recall the Pr_2-weighted MCFG G from Example 2.5. The set of productions and the weight assignment of G are:

$$\begin{array}{lll} P\colon & \rho_1 = S \to [x_1^1 x_2^1 x_1^2 x_2^2](A,B) & \mu\colon \ \mu(\rho_1) = 1 \\ & \rho_2 = A \to [a x_1^1, c x_1^2](A) & \quad\ \mu(\rho_2) = 1/2 \\ & \rho_3 = B \to [b x_1^1, d x_1^2](B) & \quad\ \mu(\rho_3) = 1/3 \\ & \rho_4 = A \to [\varepsilon, \varepsilon]() & \quad\ \mu(\rho_4) = 1/2 \\ & \rho_5 = B \to [\varepsilon, \varepsilon]() & \quad\ \mu(\rho_5) = 2/3\,. \end{array}$$

By Definitions 3.1 and 3.2 we obtain the MCFG $G_{\mathbb{B}} = \left(N, \Gamma, S, P'\right)$ where $\Gamma = \{a, b, c, d, \rho_1^1, \rho_2^1, \rho_2^2, \rho_3^1, \rho_3^2, \rho_4^1, \rho_4^2, \rho_5^1, \rho_5^2\}$ and P' is given by

$$\begin{array}{lll} P'\colon & \rho'_1 = S \to [\rho_1^1 x_1^1 x_2^1 x_1^2 x_2^2](A,B) & \\ & \rho'_2 = A \to [\rho_2^1 a x_1^1, \rho_2^2 c x_1^2](A) & \rho'_4 = A \to [\rho_4^1, \rho_4^2]() \\ & \rho'_3 = B \to [\rho_3^1 b x_1^1, \rho_3^2 d x_1^2](B) & \rho'_5 = B \to [\rho_5^1, \rho_5^2](), \end{array}$$

and the $\mathcal{A}$-weighted alphabetic homomorphism $\text{weights}_G\colon \Gamma^* \to (\Delta^* \to \mathcal{A})$ where weights_G is given for every $\gamma \in \Gamma$ and $\omega \in \Delta \cup \{\varepsilon\}$ by

$$\text{weights}_G(\gamma)(\omega) = \begin{cases} \mu(\rho_i) & \text{if } \gamma = \rho_i^1 \text{ and } \omega = \varepsilon \text{ for } 1 \le i \le 5 \\ 1 & \text{if } \gamma = \rho_i^2 \text{ and } \omega = \varepsilon \text{ for } 2 \le i \le 5 \\ 1 & \text{if } \gamma \in \Delta \text{ and } \omega = \gamma \\ 0 & \text{otherwise,} \end{cases}$$

Now recall the derivation $d = \rho_1\big(\rho_2(\rho_4), \rho_5\big)$ of $w = ac$. Then

$$\begin{aligned} f(d) &= \rho_1'\big(\rho_2'(\rho_4'), \rho_5'\big) && =: d', \\ g(d') &= \rho_1^1\, \rho_2^1\, a\, \rho_4^1\, \rho_5^1\, \rho_2^2\, c\, \rho_4^2\, \rho_5^2 && =: w', \text{ and} \\ \text{weights}_G(w') &= (1 \cdot 1/2 \cdot 1 \cdot 1/2 \cdot 2/3 \cdot 1 \cdot 1 \cdot 1 \cdot 1).w && = (1/6).w\,. \quad \square \end{aligned}$$

3.2 *The unweighted CS characterisation*

We recall the definition of multiple Dyck languages (Yoshinaka *et al.* 2010, Definition 1):

Definition 3.5. Let Δ be a finite $\mathbb{N}$-sorted set,[6] $\overline{(\cdot)}$ be a bijection between Δ and some alphabet $\overline{\Delta}$, $k = \max_{\delta \in \Delta} \text{sort}(\delta)$, and $r \ge k$. The *multiple Dyck grammar with respect to Δ and r* is the (k, r)-MCFG $G_\Delta^r = \big(\{A_1, \dots, A_k\}, \widehat{\Delta}, A_1, P\big)$ where $\widehat{\Delta} = \{\delta^{[i]}, \bar{\delta}^{[i]} \mid \delta \in \Delta, i \in [\text{sort}(\delta)]\}$, $\text{sort}(A_i) = i$ for every $i \in [k]$, and P is the smallest set such that

(i) for every linear non-deleting[7] terminal-free string function $f \in (F_\Delta)_{(s_1 \cdots s_\ell, s)}$ with $\ell \in [r]$ and $s_1, \dots, s_\ell, s \in [k]$ we have

$$A_s \to f(A_{s_1}, \dots, A_{s_\ell}) \in P\,,$$

(ii) for every $\delta \in \Delta$ with sort s we have

$$A_s \to \big[\delta^{[1]} x_1^1 \bar{\delta}^{[1]}, \dots, \delta^{[s]} x_1^s \bar{\delta}^{[s]}\big](A_s) \in P\,, \text{ and}$$

(iii) for every $s \in [k]$ we have

$$A_s \to [u_1, \dots, u_s](A_s) \in P$$

where $u_i \in \big\{x_i,\ x_i \delta^{[1]} \bar{\delta}^{[1]},\ \delta^{[1]} \bar{\delta}^{[1]} x_i \mid \delta \in \Delta_1\big\}$ for every $i \in [s]$.

[6] In Yoshinaka *et al.* (2010), $\mathbb{N}$-sorted sets are called indexed sets and sort is denoted as dim.

[7] We add the restriction "non-deleting" in comparison to the original definition since the proof of Lemma 1 in Yoshinaka *et al.* (2010) only uses non-deleting rules.

The *multiple Dyck language with respect to Δ and r*, denoted by $mD(\Delta, r)$, is $L(G^r_\Delta)$. We call $\max_{\delta\in\Delta} \mathrm{sort}(\delta)$ the *dimension* and r the *rank* of $mD(\Delta, r)$. The *set of multiple Dyck languages of dimension at most k and rank at most r* is denoted by (k, r)-mDYCK. □

Yoshinaka *et al.* (2010) define (in Section 3.2) a sorted alphabet Δ, a right-linear regular grammar R, and a homomorphism h for some given non-deleting MCFG G that has no rule with two or more identical non-terminals on the right-hand side (this form of G can be assumed without loss of generality). We recall their construction. To fit our notation and highlight the connection to G, we will conceive R as an FSA and call it $\mathscr{M}(G)$; also, h will be called hom_G.

Definition 3.6. Let $G = (N, \Gamma, S, P)$ be an MCFG. The *generator set with respect to G* is the $\mathbb{N}$-sorted alphabet

$$\Sigma = \{[\![_\gamma \mid \gamma \in \Gamma\} \cup \{[\![_\rho \mid \rho \in P\} \cup \{[\![_{\rho,i} \mid \rho \in P, i \in [\mathrm{rank}(\rho)]\}$$

where $\mathrm{sort}([\![_\gamma) = 1$, $\mathrm{sort}([\![_\rho) = \text{fan-out}(\rho)$, and $\mathrm{sort}([\![_{\rho,i}) = \text{fan-out}_i(\rho)$ for every $\gamma \in \Gamma$, $\rho \in P$, and $i \in \mathrm{rank}(\rho)$. The *generator alphabet with respect to G* is

$$\widehat{\Sigma} = \{[\![_u^{[i]},]\!]_u^{[i]} \mid [\![_u \in \Sigma, i \in [\mathrm{sort}(\sigma)]\}.$$

For each $u = \gamma_1 \cdots \gamma_m \in \Gamma^*$ (with $\gamma_1, \ldots, \gamma_m \in \Gamma$), we will abbreviate $[\![^{[1]}_{\gamma_1}]\!]^{[1]}_{\gamma_1} \cdots [\![^{[1]}_{\gamma_m}]\!]^{[1]}_{\gamma_m}$ by $\tilde{u}$. The *generator automaton with respect to G* is the FSA $\mathscr{M}(G) = (Q, \widehat{\Delta}, S^{[1]}, \{T\}, \tau)$ where $Q = \{A^{[k]} \mid A \in N, k \in [\mathrm{sort}(A)]\} \cup \{T\}$ and τ is the smallest set that contains for every production $\rho = A \to [v_1, \ldots, v_s](B_1, \ldots, B_m) \in P$ and each $k \in [s]$ (where v_k is of the form $u_{k,0} x^{j(j,1)}_{i(k,1)} u_{k,1} \cdots x^{j(k,p_k)}_{i(k,p_k)} u_{k,p_k}$ with $u_{k,0}, \ldots, u_{k,p_k} \in \Gamma^*$), the transitions

$$\begin{aligned}
&(A^{[k]}, [\![^{[k]}_\rho \tilde{u}_{k,0}]\!]^{[k]}_\rho, T) && \text{if } p_k = 0,\\
&(A^{[k]}, [\![^{[k]}_\rho \tilde{u}_{k,0} [\![^{[j(k,1)]}_{\rho,i(k,1)}, B^{[j(k,1)]}_{i(k,1)}) && \text{if } p_k > 0,\\
&(T,]\!]^{[j(k,\ell-1)]}_{\rho,i(k,\ell-1)} \tilde{u}_{k,\ell-1} [\![^{[j(k,\ell)]}_{\rho,i(k,\ell)}, B^{[j(k,\ell)]}_{i(k,\ell)}) && \text{if } p_k > 0, \text{ for every } \ell \in [p_k],\\
&(T,]\!]^{[j(k,p_k)]}_{\rho,i(k,p_k)} \tilde{u}_{k,p_k}]\!]^{[k]}_\rho, T) && \text{if } p_k > 0.
\end{aligned}$$

The *generator language with respect to G* is $R(G) = L(\mathscr{M}(G))$. The homomorphism $\mathrm{hom}_G \colon \widehat{\Delta} \to \Gamma^*$ is given by

$$\mathrm{hom}_G(\sigma) = \begin{cases} \gamma & \text{if } \sigma = [\![^{[1]}_\gamma \text{ for some } \gamma \in \Gamma \\ \varepsilon & \text{otherwise.} \end{cases}$$ □

From the four types of transitions in $\mathscr{M}(G)$, it is easy to see that $\mathscr{M}(G)$ is deterministic, i.e. for each given state and input, there is at most one successor state. We will denote $L(D_\Sigma^{\mathrm{rank}(G)})$ (cf. Definition 3.5) by $mD(G)$ to highlight its connection to the MCFG G.

Example 3.7 (Examples 2.5 and 3.4 continued)**.** Figure 5 shows the FSA $\mathscr{M}(G_{\mathbb{B}})$. An edge labelled with a set L of words denotes a set of transitions each reading a word in L. Note that $R(G_{\mathbb{B}})$ is not finite. □

The following is Theorem 3 of Yoshinaka *et al.* (2010) where "homomorphism" is replaced by "alphabetic homomorphism".

Corollary 3.8. Let L be a language, $k \in \mathbb{N}$, and $r \in \mathbb{N}_+$. Then the following are equivalent:

(i) $L \in (k, r)$-MCFL

(ii) There are an alphabetic homomorphism h_2, a regular language R, and a multiple Dyck language mD of at most dimension k and rank r with $L = h_2(R \cap mD)$.

Proof. The construction of the homomorphism in Yoshinaka *et al.* (2010, Section 3.2) already satisfies the definition of an alphabetic homomorphism. ■

Corollary 3.9. For every MCFG G, there is a bijection between D_G and $R(G) \cap mD(G)$.

Proof. The constructions in Lemmas 1 and 3 in Yoshinaka *et al.* (2010) already hint at the bijection between $R(G) \cap mD(G)$ and D_G, we will merely point out the respective functions $\mathrm{toBrackets}\colon D_G \to R(G) \cap mD(G)$ and $\mathrm{fromBrackets}\colon R(G) \cap mD(G) \to D_G$ here.

Let Δ be the generator set with respect to G and $r = \mathrm{rank}(G)$. We examine the proof of Lemma 1 in Yoshinaka *et al.* (2010). They construct for every rule $A \to f(B_1, \dots, B_k)$ in G and all tuples $\bar\tau_1, \dots, \bar\tau_k$ that are generated by $B_1, \dots, B_k$ in G, respectively, a tuple $\bar u = (u_1, \dots, u_m)$ that is generated from A_m in G^r_Δ. For each $i \in [m]$, $\mathscr{M}(G)$ recognises u_i on the way from $A^{[i]}$ to T, and $f(\hom_G(\tau_1), \dots, \hom_G(\tau_k)) = \hom_G(\bar u)$, where $\hom_G$ is applied to tuples component-wise. Now we only look at the initial non-terminal S. Then $\bar u$ has only one component and this construction can be conceived as a function $\mathrm{toBrackets}\colon D_G \to R(G) \cap mD(G)$ such that $\hom_G \circ \mathrm{toBrackets} = \mathrm{yield}$.

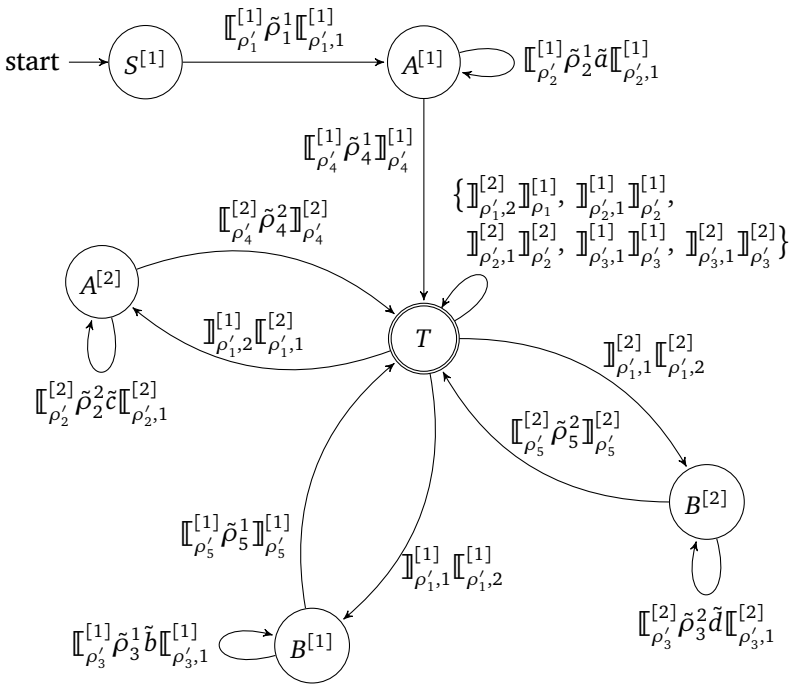

Figure 5: Automaton $\mathscr{M}(G_{\mathbb{B}})$ (cf. Example 3.7)

In Lemma 3, Yoshinaka *et al.* (2010) give a construction for the opposite direction by recursion on the structure of derivations in G_Δ. In a similar way as above, we view this construction as a function fromBrackets: $R(G) \cap mD(G) \to D_G$ such that yield $\circ$ fromBrackets $=$ $\hom_G$. Then we have $\hom_G \circ$ toBrackets $\circ$ fromBrackets $= \hom_G$, and hence toBrackets $\circ$ fromBrackets is the identity on $R(G) \cap mD(G)$. ■

3.3 *Composing the homomorphisms*

Lemma 3.10. $\alpha\mathrm{HOM}(\mathscr{A}) \circ \alpha\mathrm{HOM} = \alpha\mathrm{HOM}(\mathscr{A})$

Proof. **(⊆)** Let $h_1 \colon \Gamma^* \to (\Delta^* \to \mathscr{A})$ be an alphabetic $\mathscr{A}$-weighted homomorphism and $h_2 \colon \Sigma^* \to \Gamma^*$ be an alphabetic homomorphism. By the definitions of $\alpha\mathrm{HOM}(\mathscr{A})$ and $\alpha\mathrm{HOM}$, there exist $h'_1 \colon \Gamma \to (\Delta \cup \{\varepsilon\} \to \mathscr{A})$ and $h'_2 \colon \Sigma \to \Gamma \cup \{\varepsilon\}$ such that $\widehat{h'_1} = h_1$ and $\widehat{h'_2} = h_2$. Since $h_1(\mathrm{codom}(h'_2)) \subseteq (\Delta \cup \{\varepsilon\} \to \mathscr{A})$ there is some $h \in \alpha\mathrm{HOM}(\mathscr{A})$ such that $h = h_1 \circ h_2$; hence $h_1 \circ h_2 \in \alpha\mathrm{HOM}(\mathscr{A})$.

(⊇) Let $h\colon \Sigma \to (\Gamma^* \to \mathscr{A})$ be an alphabetic $\mathscr{A}$-weighted homomorphism. Clearly $i\colon \Sigma^* \to \Sigma^*$ with $i(w) = w$ for every $w \in \Sigma^*$ is an alphabetic homomorphism. Then we have $h \circ i = h$. ■

Example 3.11 (Examples 3.4 and 3.7 continued)**.** The homomorphism $h\colon (\Sigma \cup \bar{\Sigma})^* \to (\Delta^* \to \mathscr{A})$ obtained from $\mathrm{weights}_G\colon \Gamma^* \to (\Delta^* \to \mathscr{A})$ and $\mathrm{hom}_{G_{\mathbb{B}}}\colon (\Sigma \cup \bar{\Sigma})^* \to \Gamma^*$ by the construction for $\subseteq$ in Lemma 3.10 is given for every $\sigma \in \Sigma$ and $\omega \in \Delta \cup \{\varepsilon\}$ by

$$h(\sigma)(\omega) = \begin{cases} \mu(\rho_i) & \text{if } \sigma = [\![\rho_i]\!]^{[1]} \text{ and } \omega = \varepsilon \text{ for some } i \in [5] \\ 1 & \text{if } \sigma \notin \{[\![\rho_i]\!]^{[1]} \mid i \in [5]\} \cup \{[\![\delta]\!]^{[1]} \mid \delta \in \Delta\} \text{ and } \omega = \varepsilon \\ 1 & \text{if } \sigma = [\![\delta]\!]^{[1]} \text{ and } \omega = \delta \text{ for some } \delta \in \Delta \\ 0 & \text{otherwise.} \end{cases}$$ □

3.4 *The weighted CS characterisation*

Theorem 3.12. Let L be an $\mathscr{A}$-weighted language over Σ, $k \in \mathbb{N}$, and $r \in \mathbb{N}_+$. The following are equivalent:

(i) $L \in (k, r)$-MCFL($\mathscr{A}$)

(ii) there are an $\mathscr{A}$-weighted alphabetic homomorphism h, a regular language R, and an multiple Dyck language mD of dimension at most k and rank r with $L = h(R \cap mD)$.

Proof. **(i) ⇒ (ii)** There are some $L' \in (k, r)$-MCFL, $h, h_1 \in \alpha\mathrm{HOM}(\mathscr{A})$, $h_2 \in \alpha\mathrm{HOM}$, $mD \in k$-mDYCK$_{\mathrm{c}}$, and $R \in \mathrm{REC}$ such that

$$\begin{aligned} L &= h_1(L') && \text{(by Lemma 3.3)} \\ &= h_1(h_2(R \cap mD)) && \text{(by Corollary 3.8)} \\ &= h(R \cap mD) && \text{(by Lemma 3.10)} \end{aligned}$$

(ii) ⇒ (i) We use Definition 3.5 and Lemma 3.3, and the closure of (k, r)-MCFG under intersection with regular languages and application of homomorphisms. ■

Corollary 3.13. For every $\mathscr{A}$-weighted MCFG G, there is a bijection between D_G and $R(G_{\mathbb{B}}) \cap mD(G_{\mathbb{B}})$.

Proof. There are bijections between D_G and $L(G_{\mathbb{B}})$ by claims in the proof of Lemma 3.3, between $L(G_{\mathbb{B}})$ and $D_{G_{\mathbb{B}}}$ by claims in the proof of Lemma 3.3, and between $D_{G_{\mathbb{B}}}$ and $R(G_{\mathbb{B}}) \cap mD(G_{\mathbb{B}})$ by Corollary 3.9. ■

4 CONGRUENCE MULTIPLE DYCK LANGUAGES

According to Kanazawa (2014, Section 1) there is no definition of multiple Dyck languages using a cancellation law. The congruence multiple Dyck languages (Definition 4.2) close this gap. Even though congruence multiple Dyck languages turn out to be quite different from the multiple Dyck languages by Yoshinaka *et al.* (2010) (see Proposition 4.5 and Observation 4.6), we argue that they are still useful since they allow a CS representation (Theorem 4.11) and they can be utilised more efficiently for CS parsing than multiple Dyck languages (see Section 5.5).

For the rest of this section let Σ be an alphabet. Also let $\overline{\Sigma}$ be a set (disjoint from Σ) and $\overline{(\cdot)}$ be a bijection between Σ and $\overline{\Sigma}$. Intuitively Σ and $\overline{\Sigma}$ are sets of opening and closing parentheses and $\overline{(\cdot)}$ matches an opening to its closing parenthesis.

We define $\equiv_\Sigma$ as the smallest congruence relation on the free monoid $(\Sigma \cup \overline{\Sigma})^*$ where for every $\sigma \in \Sigma$ the cancellation rule $\sigma\overline{\sigma} \equiv_\Sigma \varepsilon$ holds. The *Dyck language with respect to* Σ, denoted by $D(\Sigma)$, is $[\varepsilon]_{\equiv_\Sigma}$. The *set of Dyck languages* is denoted by DYCK.

Example 4.1. Let $\Sigma = \{(, \langle, [, [\![\}$. We abbreviate $\overline{(}$, $\overline{\langle}$, $\overline{[}$, and $\overline{[\![}$ by $)$, $\rangle$, $]$, and $]\!]$, respectively. Then we have for example $[\![()]\!]\langle\rangle() \equiv_\Sigma [\![]\!]\langle\rangle \equiv_\Sigma [\![]\!] \equiv_\Sigma \varepsilon$ and $([\![)]\!]\langle\rangle() \equiv_\Sigma ([\![)]\!]() \equiv_\Sigma ([\![)]\!] \not\equiv_\Sigma \varepsilon$. □

Let $\mathfrak{P}$ be a partition of Σ. We define $\equiv_{\Sigma,\mathfrak{P}}$ as the smallest congruence relation on the free monoid $(\Sigma \cup \overline{\Sigma})^*$ such that if $v_1 \cdots v_\ell \equiv_{\Sigma,\mathfrak{P}} \varepsilon$ with $v_1, \ldots, v_\ell \in D(\Sigma)$, then the *cancellation rule*

$$u_0\sigma_1 v_1 \overline{\sigma_1} u_1 \cdots \sigma_\ell v_\ell \overline{\sigma_\ell} u_\ell \equiv_{\Sigma,\mathfrak{P}} u_0 \cdots u_\ell$$

holds for every $\{\sigma_1, \ldots, \sigma_\ell\} \in \mathfrak{P}$ and $u_0, \ldots, u_\ell \in D(\Sigma)$. Intuitively, every cell of $\mathfrak{P}$ denotes a set of *linked* opening parentheses, i.e. parentheses that must be consumed simultaneously by $\equiv_{\Sigma,\mathfrak{P}}$.

Definition 4.2. The *congruence multiple Dyck language with respect to* Σ *and* $\mathfrak{P}$, denoted by $mD_{\mathrm{c}}(\Sigma, \mathfrak{P})$, is $[\varepsilon]_{\equiv_{\Sigma,\mathfrak{P}}}$. □

Example 4.3. Let $\Sigma = \{(, \langle, [, [\![\}$ and $\mathfrak{P} = \{\mathfrak{p}_1, \mathfrak{p}_2\}$ where $\mathfrak{p}_1 = \{(, \langle\}$ and $\mathfrak{p}_2 = \{[, [\![\}$. We abbreviate $\overline{(}$, $\overline{\langle}$, $\overline{[}$, and $\overline{[\![}$ by $)$, $\rangle$, $]$, and $]\!]$, respectively. Then, using the cancellation rule, we have for example $[\![()]\!][\langle\rangle] \equiv_{\Sigma,\mathfrak{P}} \varepsilon$ since $\mathfrak{p}_2 = \{[, [\![\} \in \mathfrak{P}$, $()\langle\rangle \equiv_{\Sigma,\mathfrak{P}} \varepsilon$, and $u_0 = u_1 = u_2 = \varepsilon$.

But $[\![()]\!]\langle[]\rangle \not\equiv_{\Sigma,\mathfrak{P}} \varepsilon$ since when instantiating the cancellation rule with any of the two cells of $\mathfrak{P}$, we can not reduce $[\![()]\!]\langle[]\rangle$:

(i) If we choose $\{\sigma_1,\sigma_2\} = \{[\![,[\}$ then we would need to set $u_1 = \langle$ and $u_2 = \rangle$, but they are not in $D(\Sigma)$, also $() \not\equiv_{\Sigma,\mathfrak{P}} \varepsilon$;

(ii) If we choose $\{\sigma_1,\sigma_2\} = \{(,\langle\}$ then we would need to set $u_0 = [\![$ and $u_1 =]\!]$, but they are not in $D(\Sigma)$, also $[] \not\equiv_{\Sigma,\mathfrak{P}} \varepsilon$.

Hence $[\![()]\!][\langle\rangle], ()\langle\rangle \in mD_{\mathrm{c}}(\Sigma,\mathfrak{P})$ and $[\![()]\!]\langle[]\rangle \notin mD_{\mathrm{c}}(\Sigma,\mathfrak{P})$. □

Observation 4.4. From the definition of $\equiv_{\Sigma,\mathfrak{P}}$ it is easy to see that for every $u_1,\dots,u_k \in D(\Sigma)$ and $v_1,\dots,v_\ell \in D(\Sigma)$ we have that $u_1\cdots u_k$, $v_1\cdots v_\ell \in mD_{\mathrm{c}}(\Sigma,\mathfrak{P})$ implies that every permutation of $u_1,\dots,u_k$, $v_1,\dots,v_\ell$ is in $mD_{\mathrm{c}}(\Sigma,\mathfrak{P})$. ■

The *dimension of* $mD_{\mathrm{c}}(\Sigma,\mathfrak{P})$ is $\max_{\mathfrak{p}\in\mathfrak{P}}|\mathfrak{p}|$. The *set of congruence multiple Dyck languages (of at most dimension k)* is denoted by $\mathrm{mDYCK}_{\mathrm{c}}$ (k-$\mathrm{mDYCK}_{\mathrm{c}}$, respectively).

Proposition 4.5. For each $mD \in (k,r)$-mDYCK there is an $mD_{\mathrm{c}} \in k$-$\mathrm{mDYCK}_{\mathrm{c}}$ such that $mD \subseteq mD_{\mathrm{c}}$.

Proof idea. We construct a congruence multiple Dyck language mD_{c} of dimension at most k such that, if a tuple $(w_1,\dots,w_m)$ can be generated in G^r_Δ from non-terminal A_m, then $w_1,\dots,w_m \in D(\Sigma)$ and $w_1\cdots w_m \in mD_{\mathrm{e}}$. We prove this by induction on the structure of derivations in G^r_Δ.

Proof. Let $mD \in (k,r)$-mDYCK. Then there is an $\mathbb{N}$-sorted set Δ such that $mD = mD(\Delta,r)$ and $\max_{\delta\in\Delta}\mathrm{sort}(\delta) \le k$. We define $\mathfrak{p}_\delta = \{\delta^{[i]} \mid i \in [\mathrm{sort}(\delta)]\}$ for every $\delta \in \Delta$, $\Sigma = \bigcup_{\delta\in\Delta}\mathfrak{p}_\delta$, and $\mathfrak{P} = \{\mathfrak{p}_\delta \mid \delta \in \Delta\}$. Clearly $\max_{\mathfrak{p}\in\mathfrak{P}}|\mathfrak{p}| \le k$. Thus $mD_{\mathrm{c}}(\Sigma,\mathfrak{P}) \in k$-$\mathrm{mDYCK}_{\mathrm{c}}$. Let $\mathrm{Tup}(G^r_\Delta,A)$ denote the set of tuples generated in G^r_Δ when starting with non-terminal A where A is not necessarily initial. In the following we show that for every $m \in [\max_{\delta\in\Delta}\mathrm{sort}(\delta)]$ and $w_1,\dots,w_m \in (\Sigma\cup\bar{\Sigma})^*$:

$$(w_1,\dots,w_m) \in \mathrm{Tup}(G^r_\Delta,A_m) \Longrightarrow w_1\cdots w_m \in mD_{\mathrm{c}}(\Sigma,\mathfrak{P}) \wedge w_1,\dots,w_m \in D(\Sigma)\,. \qquad (*)$$

It follows from the definitions of Tup and G^r_Δ that $(w_1,\dots,w_m) \in \mathrm{Tup}(G^r_\Delta,A_m)$ implies that there are a rule $A_m \to f(A_{m_1},\dots,A_{m_\ell})$ in G^r_Δ and tuples $\vec{u}_i = (u_i^1,\dots,u_i^{m_i}) \in \mathrm{Tup}(G^r_\Delta,A_{m_i})$ for every $i \in [\ell]$ such

that $f(\vec{u}_1,\dots,\vec{u}_\ell) = (w_1,\dots,w_m)$. By applying the induction hypothesis ℓ times, we also have that $u_1^1,\dots,u_1^{m_1},\dots,u_\ell^1,\dots,u_\ell^{m_\ell} \in D(\Sigma)$ and $u_1^1\cdots u_1^{m_1},\dots,u_\ell^1\cdots u_\ell^{m_\ell} \in mD_{\mathbf{c}}(\Sigma,\mathfrak{P})$. We distinguish three cases (each corresponding to one type of rule in G_Δ^r):

(i) f is linear, non-deleting, and terminal-free. Then we have for every $i \in [m]$ that $w_i \in \{u_1^1,\dots,u_1^{m_1},\dots,u_\ell^1,\dots,u_\ell^{m_\ell}\}^*$ and therefore also $w_i \in D(\Sigma)$. Furthermore, by applying Observation 4.4 $(\ell-1)$ times, we have that $w_1\cdots w_m \in mD_{\mathbf{c}}(\Sigma,\mathfrak{P})$.

(ii) $f = [\delta^{[1]}x_1^1\bar{\delta}^{[1]},\dots,\delta^{[m]}x_1^m\bar{\delta}^{[m]}]$; then $\ell = 1$, $m_1 = m$, and for every $i \in [m]$ we have $w_i = \delta^{[i]}u_1^i\bar{\delta}^{[i]}$ and since $u_1^i \in D(\Sigma)$, also $w_i \in D(\Sigma)$. Furthermore, $w_1\cdots w_m = \delta^{[1]}u_1^1\bar{\delta}^{[1]}\cdots\delta^{[m]}u_1^m\bar{\delta}^{[m]} \in mD_{\mathbf{c}}(\Sigma,\mathfrak{P})$ due to the cancellation rule.

(iii) $f = [u_1,\dots,u_m]$ where $u_i \in \left\{x_i^1,\ x_i^1\delta^{[1]}\bar{\delta}^{[1]},\ \delta^{[1]}\bar{\delta}^{[1]}x_i^1 \mid \delta \in \Delta_1\right\}$ for every $i \in [m]$; then $w_i \in \left\{u_i^1,\ u_i^1\delta^{[1]}\bar{\delta}^{[1]},\ \delta^{[1]}\bar{\delta}^{[1]}u_i^1 \mid \delta \in \Delta_1\right\}$ for every $i \in [m]$, $\ell = 1$, and $m_1 = m$. Since $\equiv_\Sigma$ is a congruence relation (in particular, $\equiv_\Sigma$ respects composition), we have that $w_1,\dots,w_m \in D(\Sigma)$. By applying Observation 4.4 m times, we have that $w_1\cdots w_m \in mD_{\mathbf{c}}(\Sigma,\mathfrak{P})$. ∎

From the above proof we can easily see that the rank r of a multiple Dyck language can not be taken into account by a congruence multiple Dyck language. This leads us to the next observation.

Observation 4.6. Let mD be a multiple Dyck language and $mD_{\mathbf{c}}$ be a congruence multiple Dyck language. Then $mD \neq mD_{\mathbf{c}}$. ∎

Similar to multiple Dyck languages, the congruence multiple Dyck languages cover the Dyck languages if we set the dimension to 1. Also they form a hierarchy with increasing dimension.

Proposition 4.7. $\mathrm{DYCK} = 1\text{-mDYCK}_{\mathbf{c}} \subsetneq 2\text{-mDYCK}_{\mathbf{c}} \subsetneq \dots$

Proof. We have the equality since the dimension of some partition $\mathfrak{P}$ of Σ is 1 if and only if $\mathfrak{P} = \{\{\sigma\} \mid \sigma \in \Sigma\}$. Then we have $\equiv_\Sigma = \equiv_{\Sigma,\mathfrak{P}}$ and thus $D(\Sigma) = mD_{\mathbf{c}}(\Sigma,\mathfrak{P})$. Hence $\mathrm{DYCK} = 1\text{-mDYCK}_{\mathbf{c}}$. We get "$\subsetneq$" from the definition of k-$\mathrm{mDYCK}_{\mathbf{c}}$. ∎

4.1 *Membership in a congruence multiple Dyck language*

We give an algorithm to decide membership in a congruence multiple Dyck language (Algorithm 2). It is closely related to the cancellation

Algorithm 2: Function *isMember* to decide membership in $mD_{\mathbf{c}}(\Sigma, \mathfrak{P})$

Input: Σ, $\mathfrak{P}$, and $w \in (\Sigma \cup \overline{\Sigma})^*$
Output: True if $w \in mD_{\mathbf{c}}(\Sigma, \mathfrak{P})$, False otherwise

```
1  function isMember(Σ, 𝔓, w)
2    if w ∉ D(Σ) then return False end if
3    let (σ_1 u_1 σ̄_1, ..., σ_ℓ u_ℓ σ̄_ℓ) = split(Σ, w) such that σ_1, ..., σ_ℓ ∈ Σ
4    let 𝒥 = ∅
5    for each I = {i_1, ..., i_k} ⊆ [ℓ] with {σ_{i_1}, ..., σ_{i_k}} ∈ 𝔓 do
6      if isMember(Σ, 𝔓, u_{i_1} ··· u_{i_k}) then
7        add I as an element to 𝒥
8      end if
9    end for
10   for each J ⊆ 𝒥 do
11     if J is a partition of [ℓ] then return True end if
12   end for
13   return False
14 end function
```

rule and thus provides an algorithmic view on congruence multiple Dyck languages. Although the algorithm is at least exponential in a polynomial of the length of the input word, it becomes quadratic if we only accept input words of a specific form. The parsing algorithm presented in Section 5 will only consider words of that form.

Algorithm 2 works roughly as follows: It is a recursive algorithm. In every call of *isMember*, it checks if the given word can be reduced to ε by applications of the cancellation rule. For substrings $\sigma_1 \nu_1 \bar{\sigma}_1, \ldots, \sigma_\ell \nu_\ell \bar{\sigma}_\ell$ to be cancelled, the string $\nu_1 \cdots \nu_\ell$ must be an equivalence multiple Dyck word; this is checked with a recursive call to *isMember*. Note that it suffices to only apply the cancellation rule from left to right. To account for all possible applications of the cancellation rule, *isMember* must consider all decompositions of the input string into Dyck words. For this purpose, recall *split* (from Section 4) that splits a given Dyck word into shortest non-empty Dyck words.

Outline of *isMember*

In the following, all line numbers refer to Algorithm 2. We first check if w is in $D(\Sigma)$, e.g. with the context-free grammar in (7.6) in Salomaa (1973). If w is not in $D(\Sigma)$, it is also not in $mD_{\mathbf{c}}(\Sigma, \mathfrak{P})$ and we return False. Otherwise, we split w into shortest non-empty

Dyck words (on line 3), i.e. we compute the tuple $(u_1, \dots, u_\ell)$ such that $u_1, \dots, u_\ell \in D(\Sigma) \setminus \{\varepsilon\}$, $u_1 \cdots u_\ell = u$, and for every $i \in [\ell]$ there are no $u_i', v_i' \in D(\Sigma) \setminus \{\varepsilon\}$ with $u_i' v_i' = u_i$. We denote $(u_1, \dots, u_\ell)$ by $\mathit{split}(u)$. Note that $\mathit{split}(u)$ can be calculated in time and space linear in $|u|$ with the help of a pushdown transducer (Aho and Ullman 1972, Section 3.1.4): We initially write "(" on the output tape. Whenever we read an element of Σ, we write that element on the output tape and push it on the stack. Whenever we read an element of $\overline{\Sigma}$, we write that element on the output tape and pop it from the stack. Upon reaching the bottom of the stack, we write a "," on the output tape. Finally, we write ")" on the output tape. Then the inscription of the output tape is $(u_1, \dots, u_\ell)$. Since each of those shortest non-empty Dyck words has the form $\sigma u \overline{\sigma}$ for some $\sigma \in \Sigma$ and $u \in (\Sigma \cup \overline{\Sigma})^*$, we write $(\sigma_1 u_1 \overline{\sigma}_1, \dots, \sigma_\ell u_\ell \overline{\sigma}_\ell)$ for the left-hand side of the assignment on line 3. On lines 4 to 9 we calculate the set $\mathscr{I}$ of sets of indices $I = \{i_1, \dots, i_k\}$ such that the outer parenteses of the substrings $\sigma_{i_1} u_{i_1} \overline{\sigma}_{i_1}, \dots, \sigma_{i_k} u_{i_k} \overline{\sigma}_{i_k}$ of w can be reduced with one step of the cancellation rule. This reduction is possible if there exists an appropriate cell in $\mathfrak{P}$ (checked on line 5) and if $u_{i_1} \cdots u_{i_k}$ is in $mD_{\mathbf{c}}(\Sigma, \mathfrak{P})$ (checked on line 6). Therefore, at the end of line 9, each element of $\mathscr{I}$ represents one possible application of the cancellation rule. In order for $\equiv_{\Sigma,\mathfrak{P}}$ to reduce w to ε, each component of $(\sigma_1 u_1 \overline{\sigma}_1, \dots, \sigma_\ell u_\ell \overline{\sigma}_\ell)$ needs to be reduced (exactly once) by an application of the cancellation rule. This is equivalent to a subset of $\mathscr{I}$ being a partition of $[\ell]$ (lines 10 to 12). If no such subset exists, then w can not be reduced to ε and we return False on line 13.

Example 4.8 (Example 4.3 continued)**.** Table 1 shows a run of Algorithm 2 on the word $[\![()]\!][\langle\rangle]$ where we report return values and a subset of the variable assignment, when necessary, at the line ending. The recursive calls to *isMember* are indented. Table 2 shows the run of Algorithm 2 on the word $[\![()]\!][]\![]\!][\langle\rangle]$. □

In light of the close link between Algorithm 2 and the relation $\equiv_{\Sigma,\mathfrak{P}}$ we omit the proof of correctness.

Proof of termination for Algorithm 2. If $w = \varepsilon$, then $\ell = 0$, the loop on lines 5 to 9 has no I's to consider, the loop on lines 10 to 13 has only $J = \varnothing$ to consider, which is a partition of $[\ell] = \varnothing$, and hence the algorithm terminates on line 11. If $w \notin D(\Sigma)$, the algorithm terminates

Table 1: Run of Algorithm 2 on the word $[\![()]\!][\langle\rangle]$, cf. Examples 4.3 and 4.8.

	isMember$(\Sigma, \mathfrak{P}, [\![()]\!][\langle\rangle])$
line 3:	$\ell = 2, \sigma_1 = [\![, \sigma_2 = [, u_1 = (), u_2 = \langle\rangle$
line 5:	$\mathscr{I} = \varnothing,\ \ k = 2, i_1 = 1, i_2 = 2$
line 6:	*isMember*$(\Sigma, \mathfrak{P}, ()\langle\rangle)$
	line 3: $\ell = 2, \sigma_1 = (, \sigma_2 = \langle, u_1 = \varepsilon = u_2$
	line 5: $\mathscr{I} = \varnothing,\ \ k = 2, i_1 = 1, i_2 = 2$
	line 6: *isMember*$(\Sigma, \mathfrak{P}, \varepsilon)$
	line 3: $\ell = 0$
	line 10: $J = \varnothing$
	line 11: **return** True
	line 7: $\mathscr{I} = \{\{1, 2\}\}$
	line 10: $J = \varnothing$
	line 10: $J = \{\{1, 2\}\}$
	line 11: **return** True
line 7:	$\mathscr{I} = \{\{1, 2\}\}$
line 10:	$J = \varnothing$
line 10:	$J = \{\{1, 2\}\}$
line 11:	**return** True

on line 2. The loop on lines 5 to 9 considers only finitely many values I. Thus there are only finitely many calls to *isMember* on line 6 for each recursion. In the call of *isMember*, the length of the third argument $u_{i_1} \cdots u_{i_k}$ is strictly smaller then the length of w. Therefore, after a finite number of recursions, the fourth argument passed to *isMember* becomes the empty word and the algorithm terminates. ■

Properties of *isMember*

Algorithm 2 is at least exponential in a polynomial of the length of the input word. This is due to the cardinality of $\mathscr{I}$ and the **for**-loop on lines 10 to 12. Let κ be the number of different cells $\mathfrak{p} \in \mathfrak{P}$ that occur in $\sigma_1 \cdots \sigma_\ell$, and let each symbol occurs at most r times. Both κ and r have upper bound ℓ. Let m be the dimension of $\mathfrak{P}$. Then there are at most $\kappa \cdot r^{m-1} \leq \ell^m$ values of I considered in the **for**-loop on lines 5 to 9. Since $\ell < |w|$, we execute this **for**-loop at most $|w|^m$ times. Hence, $\mathscr{I}$ has cardinality at most $|w|^m$. Therefore, the **for**-loop on lines 10 to 12 considers $2^{|\mathscr{I}|} \leq 2^{|w|^m}$ different values of J in the worst case.

Let us now turn to the modification of *isMember* we will use in Section 5. Let $u \in D(\Sigma)$ and $(\sigma_1 u_1' \overline{\sigma}_1, \ldots, \sigma_\ell u_\ell' \overline{\sigma}_\ell) = \mathit{split}(u)$. For every $\sigma \in \Sigma$, we define $\operatorname{occ}_\sigma u = |\{i \in [\ell] \mid \sigma_i = \sigma\}|$. Furthermore for every $\mathfrak{p} \in \mathfrak{P}$, we define $\operatorname{occ}_\mathfrak{p} u = \max\{\operatorname{occ}_\sigma u \mid \sigma \in \mathfrak{p}\}$ and we define

Table 2: Run of Algorithm 2 on the word $[\![()]\!][\,][\![\,]\!][\,[\langle\rangle\,]]$.

isMember$(\Sigma, \mathfrak{P}, [\![()]\!][\,][\![\,]\!][\,[\langle\rangle\,]])$
line 3: $\ell = 4, \sigma_1 = [\![= \sigma_3, \sigma_2 = [\, = \sigma_4, u_1 = (), u_2 = \varepsilon = u_3, u_4 = \langle\rangle$

line 5: $\mathscr{I} = \varnothing,\ \ k = 2, i_1 = 1, i_2 = 2$
line 6: *isMember*$(\Sigma, \mathfrak{P}, ())$
line 3: $\ell = 1, \sigma_1 = (, u_1 = \varepsilon$
line 9: $\mathscr{I} = \varnothing$
line 10: $J = \varnothing$
line 13: **return** False
line 8: $\mathscr{I} = \varnothing$

line 5: $k = 2, i_1 = 1, i_2 = 4$
line 6: *isMember*$(\Sigma, \mathfrak{P}, ()\langle\rangle)$
line 3: $\ell = 2, \sigma_1 = (, \sigma_2 = \langle, u_1 = \varepsilon = u_2$
line 5: $\mathscr{I} = \varnothing,\ \ k = 2, i_1 = 1, i_2 = 2$
line 6: *isMember*$(\Sigma, \mathfrak{P}, \varepsilon)$
line 3: $\ell = 0$
line 9: $\mathscr{I} = \varnothing$
line 10: $J = \varnothing$
line 11: **return** True
line 7: $\mathscr{I} = \big\{\{1,2\}\big\}$
line 10: $J = \varnothing$
line 10: $J = \big\{\{1,2\}\big\}$
line 13: **return** True
line 7: $\mathscr{I} = \big\{\{1,4\}\big\}$

line 5: $k = 2, i_1 = 2, i_2 = 3$
line 6: *isMember*$(\Sigma, \mathfrak{P}, \varepsilon)$
line 3: $\ell = 0$
line 10: $\mathscr{I} = \varnothing,\ \ J = \varnothing$
line 11: **return** True
line 7: $\mathscr{I} = \big\{\{1,4\},\{2,3\}\big\}$

line 5: $k = 2, i_1 = 3, i_2 = 4$
line 6: *isMember*$(\Sigma, \mathfrak{P}, \langle\rangle)$
line 3: $\ell = 1, \sigma_1 = \langle, u_1 = \varepsilon$
line 10: $\mathscr{I} = \varnothing,\ \ J = \varnothing$
line 13: **return** False
line 8: $\mathscr{I} = \big\{\{1,4\},\{2,3\}\big\}$

line 10: $J = \varnothing$
line 10: $J = \big\{\{1,4\}\big\}$
line 10: $J = \big\{\{2,3\}\big\}$
line 10: $J = \big\{\{1,4\},\{2,3\}\big\}$
line 11: **return** True

$\operatorname{occ}_{\mathfrak{P}} u = \sum_{\mathfrak{p}\in\mathfrak{P}} \operatorname{occ}_{\mathfrak{p}} u$. We call a word $w \in (\Sigma\cup\bar{\Sigma})^*$ $\mathfrak{P}$-*simple* if $w \notin D(\Sigma)$; or $\operatorname{occ}_{\mathfrak{p}} w \leq 1$ for each cell $\mathfrak{p} \in \{\mathfrak{p}' \in \mathfrak{P} \mid |\mathfrak{p}'| \geq 2\}$ and $v_1 \cdots v_\ell$ is $\mathfrak{P}$-simple whenever there are $u_0, \ldots, u_\ell \in D(\Sigma)$, $v_1, \ldots, v_\ell \in D(\Sigma)$, and $\mathfrak{p} = \{\sigma_1, \ldots, \sigma_\ell\} \in \mathfrak{P}$ with $w = u_0\sigma_1 v_1 \bar{\sigma}_1 u_1 \cdots \sigma_\ell v_\ell \bar{\sigma}_\ell u_\ell$. In other words, w is $\mathfrak{P}$-simple if, whenever the cancellation rule can be applied to w to cancel an occurrence of the cell $\mathfrak{p}$ (where $\mathfrak{p}$ has more than one element), then there is only one such occurrence and the string $v_1 \cdots v_\ell$ (from the definition of the cancellation rule) is also $\mathfrak{P}$-simple.

In order for *isMember* to recognise w only if it is $\mathfrak{P}$-simple, we check between lines 1 and 2 in the algorithm whether $\operatorname{occ}_{\mathfrak{p}} w \leq 1$ for each cell $\mathfrak{p} \in \{\mathfrak{p}' \in \mathfrak{P} \mid |\mathfrak{p}'| \geq 2\}$. If this is the case, we continue, otherwise we return False. Let us call the function obtained in this manner *isMember'*. Note that the check between lines 1 and 2 can be done in time linear in the input word. Then the I's that Algorithm 2 considers in the **for**-loop on lines 5 to 9 are pairwise disjoint. This means that each u_i (for $i \in [\ell]$) occurs in at most one recursive call on line 6. Then the elements of $\mathscr{I}$ are always pairwise disjoint and we only need to consider $J = \mathscr{I}$ in the **for**-loop on lines 10 to 12. We can decide in time $\mathcal{O}(\ell)$ whether $\mathscr{I}$ is a partition of $[\ell]$. Lines 2 and 3 can be computed in time $\mathcal{O}(|w|)$. Since $\ell < |w|$, we know that for each call of *isMember*, we have to invest time linear in the length of the third argument. The maximum depth of recursion is $|w|/2$ because the third argument in the call on line 6 has at most length $|w|-2$. For every recursion depth, the sum of the lengths of all third arguments is at most $|w|$ because u_i (for $i \in [\ell]$) occurs in at most one recursive call on line 6. Therefore *isMember'*$(\Sigma, \mathfrak{P}, w)$ can be calculated in time $\mathcal{O}(|w|^2)$.

4.2 *A CS representation using congruence multiple Dyck languages*

Definition 4.9. Let $G = (N, \Gamma, S, P)$ be an MCFG. The *congruence multiple Dyck language with respect to* G, denoted by $mD_{\mathrm{c}}(G)$, is $mD_{\mathrm{c}}(\Sigma, \mathfrak{P})$ where $\mathfrak{P}$ is the smallest set such that

- $\mathfrak{p}_\gamma = \{[\![\gamma]\!]^{[1]}\} \in \mathfrak{P}$ for every $\gamma \in \Gamma$,
- $\mathfrak{p}_\rho = \{[\![\rho]\!]^{[j]} \mid j \in [\text{fan-out}(\rho)]\} \in \mathfrak{P}$ for every $\rho \in P$, and
- $\mathfrak{p}_{\rho,i} = \{[\![\rho]\!]^{[j]}_{i} \mid j \in [\text{fan-out}_i(\rho)]\} \in \mathfrak{P}$ for every $\rho \in P$ and $i \in [\text{rank}(\rho)]$,

and $\Sigma = \bigcup_{\mathfrak{p}\in\mathfrak{P}} \mathfrak{p}$. □

Lemma 4.10. $R(G) \cap mD(G) = R(G) \cap mD_{\mathbf{c}}(G)$ for each MCFG G.

Proof. It follows from Definitions 3.5 and 4.9 and Proposition 4.5 that $mD(G) \subseteq mD_{\mathbf{c}}(G)$ and hence $R(G) \cap mD(G) \subseteq R(G) \cap mD_{\mathbf{c}}(G)$. It remains to be shown that $R(G) \cap mD_{\mathbf{c}}(G) \subseteq mD(G)$. Let $G = (N, \Gamma, S, P)$ and Σ and Δ be defined as in Definitions 3.5 and 4.9.

We prove the following statement by induction on the length of $w_1 \cdots w_\ell$: If $B \in N$ and $w_1, \ldots, w_\ell \in D(\Sigma)$ such that $w_1 \cdots w_\ell \in mD_{\mathbf{c}}$ and w_κ is recognised along a path from $B^{[\kappa]}$ to T in $\mathscr{M}(G)$ for every $\kappa \in [\ell]$, then $(w_1, \ldots, w_\ell)$ can be generated by A_ℓ in $G_\Delta^{\mathrm{rank}(G)}$.

By setting $\ell = 1$ and $B = S$, this statement implies our claim. Now let $B \in N$ and $w_1, \ldots, w_k \in D(\Sigma)$ such that $w_1 \cdots w_\ell \in mD_{\mathbf{c}}$ and w_κ is recognised along a path from $B^{[\kappa]}$ to T in $\mathscr{M}(G)$ for every $\kappa \in [\ell]$. By the definitions of $\mathscr{M}(G)$ and $mD_{\mathbf{c}}(G)$, we know that there is some production $\rho = B \to f(B_1, \ldots, B_k) \in P$ with

$$f = [u_{1,0} x_{i(1,1)}^{j(1,1)} u_{1,1} \cdots x_{i(1,p_1)}^{j(1,p_1)} u_{1,p_1}, \ldots, u_{\ell,0} x_{i(\ell,1)}^{j(\ell,1)} u_{\ell,1} \cdots x_{i(\ell,p_\ell)}^{j(\ell,p_\ell)} u_{\ell,p_\ell}]$$

such that for every $\kappa \in [\ell]$ either

(i) $w_\kappa = [\![_\rho^{[\kappa]} \widetilde{u}_{\kappa,0}]\!]_\rho^{[\kappa]}$ or

(ii) $w_\kappa = [\![_\rho^{[\kappa]} \widetilde{u}_{\kappa,0} \; [\![_{\rho,i(\kappa,1)}^{[j(\kappa,1)]} \; v_{i(\kappa,1)}^{j(\kappa,1)} \;]\!]_{\rho,i(\kappa,1)}^{[j(\kappa,1)]} \; \widetilde{u}_{\kappa,1} \; \cdots$
$[\![_{\rho,i(\kappa,p_\kappa)}^{[j(\kappa,p_\kappa)]} \; v_{i(\kappa,p_\kappa)}^{j(\kappa,p_\kappa)} \;]\!]_{\rho,i(\kappa,p_\kappa)}^{[j(\kappa,p_\kappa)]} \; \widetilde{u}_{\kappa,p_\kappa} \;]\!]_\rho^{[\kappa]}$,

and $v_i^j \in D(\Sigma)$ is recognised along a path from $B_i^{[j]}$ to T in $\mathscr{M}(G)$ for every $i \in [k]$ and $j \in [\mathrm{sort}(B_i)]$, and $v_i^1 \cdots v_i^{\mathrm{sort}(B_i)} \in mD_{\mathbf{c}}(G)$ for every $i \in [k]$. Then by induction hypothesis $(v_i^1, \ldots, v_i^{\mathrm{sort}(B_i)})$ can be generated from $A_{\mathrm{sort}(B_i)}$ in $G_\Delta^{\mathrm{rank}(G)}$ for every $i \in [k]$. Using productions of types (ii) and (iii) (cf. Definition 3.5), $A_{\mathrm{sort}(B_1)}, \ldots, A_{\mathrm{sort}(B_k)}$ can generate tuples that together have exactly the components

$$\begin{aligned}
&\widetilde{u}_{1,0} [\![_{\rho,i(1,1)}^{[j(1,1)]} v_{i(1,1)}^{j(1,1)}]\!]_{\rho,i(1,1)}^{[j(1,1)]} \widetilde{u}_{1,1}, [\![_{\rho,i(1,2)}^{[j(1,2)]} v_{i(1,2)}^{j(1,2)}]\!]_{\rho,i(1,2)}^{[j(1,2)]} \widetilde{u}_{1,2}, \ldots, \\
&\qquad\qquad [\![_{\rho,i(1,p_1)}^{[j(1,p_1)]} v_{i(1,p_1)}^{j(1,p_1)}]\!]_{\rho,i(1,p_1)}^{[j(1,p_1)]} \widetilde{u}_{1,p_1}, \quad \ldots, \\
&\widetilde{u}_{\ell,0} [\![_{\rho,i(\ell,1)}^{[j(\ell,1)]} v_{i(\ell,1)}^{j(\ell,1)}]\!]_{\rho,i(\ell,1)}^{[j(\ell,1)]} \widetilde{u}_{\ell,1}, [\![_{\rho,i(\ell,2)}^{[j(\ell,2)]} v_{i(\ell,2)}^{j(\ell,2)}]\!]_{\rho,i(\ell,2)}^{[j(\ell,2)]} \widetilde{u}_{\ell,2}, \ldots, \\
&\qquad\qquad [\![_{\rho,i(\ell,p_\ell)}^{[j(\ell,p_\ell)]} v_{i(\ell,p_\ell)}^{j(\ell,p_\ell)}]\!]_{\rho,i(\ell,p_\ell)}^{[j(\ell,p_\ell)]} \widetilde{u}_{\ell,p_\ell}.
\end{aligned}$$

Set $w'_1, \ldots, w'_\ell \in D(\Sigma)$ such that $w_\kappa = [\![_\rho^{[\kappa]} w'_\kappa]\!]_\rho^{[\kappa]}$ for every $k \in [\ell]$. Then we can derive $(w'_1, \ldots, w'_\ell)$ from A_ℓ in $G_\Delta^{\mathrm{rank}(G)}$ by first using a production

of type (i) with rank exactly $\operatorname{rank}(\rho)$, and for each $\kappa \in [\ell]$ where w_κ has the form $[\![^{[\kappa]}_\rho \widetilde{u}_{\kappa,0}]\!]^{[\kappa]}_\rho$ productions of type (iii). Using a production of type (ii), we finally obtain $(w_1, \dots, w_\ell)$ from A_ℓ in $G_\Delta^{\operatorname{rank}(G)}$. ∎

Theorem 4.11. For every multiple context-free languages L of fan-out at most k, there exist a weighted homomorphism h, a regular language R, and a congruence multiple Dyck language $mD_{\mathbf{c}}$ of dimension at most k such that $L = h(R \cap mD_{\mathbf{c}})$.

Proof. This follows immediately from Lemma 4.10 and Theorem 3.12 when taking $h = \operatorname{weights}_G \circ \operatorname{hom}_{G_{\mathbb{B}}}$, $R = R(G_{\mathbb{B}})$, and $mD_{\mathbf{c}} = mD_{\mathbf{c}}(G_{\mathbb{B}})$. ∎

5 n-BEST PARSING FOR WEIGHTED MCFGS USING A CHOMSKY-SCHÜTZENBERGER REPRESENTATION

In this section we describe an n-best parsing algorithm (cf. Huang and Chiang 2005; Büchse *et al.* 2010) for a subset of weighted MCFGs, i.e. we want to find the best parses of a given word in a weighted MCFG. In our case, "parse" refers to a derivation in the weighted MCFG. We formalise our intuition of when a derivation is "better" than another derivation by means of a partial order $\trianglelefteq$ on the weights. That is, if we have two derivations d_1 and d_2 with weights μ_1 and μ_2, respectively, we call d_1 "not worse than" d_2 if $\mu_1 \trianglelefteq \mu_2$. Note that we think of weights as *costs* here, i.e. better derivations will get a smaller weight with respect to $\trianglelefteq$. We define the *irreflexive relation with respect to* $\trianglelefteq$ as $\triangleleft = \trianglelefteq \setminus \{(a,a) \mid a \in A\} \subseteq A \times A$. We say that $\mathscr{A}$ *respects* $\trianglelefteq$ if $\cdot$ has the following three properties:

(i) it is *(strictly) increasing*, i.e. $a \triangleleft a \cdot b$ for every $a, b \in \mathscr{A} \setminus \{0, 1\}$,

(ii) it has *arbitrarily large powers*, i.e. for every $a, b \in \mathscr{A} \setminus \{0, 1\}$ there is a $k \in \mathbb{N}$ with $b \trianglelefteq a^k$ where $a^k = a^{k-1} \cdot a$ for $k \geq 1$ and $a^0 = 1$, and

(iii) it is *monotone*, i.e. $a \trianglelefteq b$ implies $a \cdot c \trianglelefteq b \cdot c$ for every $a, b, c \in \mathscr{A}$.

For the rest of this section let $\operatorname{argmin}^{\trianglelefteq}$ be a function that assigns for every family $f : B \to \mathscr{A}$ a value $\bar{b} \in B$ such that there is no $b' \in B$ with $f(b') \triangleleft \bar{b}$; we write $\operatorname{argmin}^{\trianglelefteq}_{b \in B}(f(b))$ for $\bar{b}$.

Note that we only need multiplication to obtain the weight of a derivation and therefore the sum operation of our complete commutative strong bimonoid becomes irrelevant within this section.

5.1 *n-best parsing*

We will take the n best parses from the possibly infinite sequence of derivations for some word in an MCFG. We therefore need a notion of infinite strings.

Definition 5.1. Let B be a set. The *set of infinite strings over* B, denoted by B^ω, is the set of partial functions $u\colon \mathbb{N} \setminus \{0\} \to B$ where the fact that $u(n)$ is defined implies that $u(n-1)$ is defined as well, for every $n > 1$. □

Every element u of B^* can be construed as an element of B^ω where $\mathrm{dom}(u)$ is finite. Let $u \in B^*$ and $v \in B^\omega$. The *concatenation of* u *and* v, denoted by $u \cdot^\omega v$, is given by

$$(u \cdot^\omega v)(n) = \begin{cases} u(n) & \text{if } n \leq |u| \\ v(n-|u|) & \text{otherwise.} \end{cases}$$

To work with infinite strings, we define the following functions:

map applies a function to every element in an infinite list.

$$\begin{aligned} \mathrm{map}\colon (B \to C) &\to (B^\omega \to C^\omega) \\ \mathrm{map}(f)(bu) &= f(b) \cdot^\omega \mathrm{map}(f)(u) && \text{(if } b \in B\text{)} \\ \mathrm{map}(f)(\varepsilon) &= \varepsilon \end{aligned}$$

take returns a finite prefix of a given infinite list.

$$\begin{aligned} \mathrm{take}\colon \mathbb{N} &\to (B^\omega \to B^*) \\ \mathrm{take}(n)(bu) &= b \cdot^\omega \mathrm{take}(n-1)(u') && \text{(if } n > 0 \text{ and } b \in B\text{)} \\ \mathrm{take}(n)(u) &= \varepsilon && \text{(if } n = 0 \text{ or } u = \varepsilon\text{)} \end{aligned}$$

filter removes elements that are not in a given set from an infinite list.

$$\begin{aligned} \mathrm{filter}\colon \mathscr{P}(B) &\to (B^\omega \to B^\omega) \\ \mathrm{filter}(B')(b'u) &= b' \cdot^\omega \mathrm{filter}(B')(u) && \text{(if } b' \in B'\text{)} \\ \mathrm{filter}(B')(bu) &= \mathrm{filter}(B')(u) && \text{(if } b \notin B'\text{)} \\ \mathrm{filter}(B')(\varepsilon) &= \varepsilon \end{aligned}$$

sort returns an infinite list that contains each element of a given set exactly once in an order that respects $\trianglelefteq$.

$$\begin{aligned}\text{sort}\colon (B \to \mathscr{A}) \times \mathscr{P}(\mathscr{A} \times \mathscr{A}) &\to (\mathscr{P}(B) \to B^\omega)\\ \text{sort}(f, \trianglelefteq)(B') &= \text{argmin}^{\trianglelefteq}_{b \in B'}(f(b))\\ &\quad \cdot^\omega \text{sort}(f, \trianglelefteq)(B' \setminus \text{argmin}^{\trianglelefteq}_{b \in B'}(f(b))) \quad (\text{if } B' \neq \varnothing)\\ \text{sort}(f, \trianglelefteq)(\varnothing) &= \varepsilon\end{aligned}$$

Definition 5.2. Let B be a set, $f : B \to \mathscr{A}$, $n \in \mathbb{N}$, and $\trianglelefteq$ be a partial order on $\mathscr{A}$. We define the *n-best function with respect to f and $\trianglelefteq$* as a function $n\text{-best}(f, \trianglelefteq)\colon \mathscr{P}(B) \to \mathscr{P}(B^*)$ where for every $B' \subseteq B$ we have that $(b_1, \ldots, b_k) \in n\text{-best}(f, \trianglelefteq)(B')$ if the following conditions hold

(i) $k = \min\{n, |B'|\}$,

(ii) $b_1, \ldots, b_k \in B'$ are pairwise different,

(iii) $f(b_1) \trianglelefteq f(b_2) \trianglelefteq \ldots \trianglelefteq f(b_k)$, and

(iv) there is no $b \in B' \setminus \{b_1, \ldots, b_k\}$ with $f(b) \triangleleft f(b_k)$. □

Note that $|n\text{-best}(f, \trianglelefteq)(B')| = 1$ if $n \leq |B'|$.

Definition 5.3. The *parsing problem for $\mathscr{A}$-weighted MCFG* is:

given an $\mathscr{A}$-weighted MCFG $G = (N, \Sigma, S, P, \mu)$, a partial order $\trianglelefteq$ on $\mathscr{A}$, a word $w \in \Sigma^*$, and an integer $n \in \mathbb{N}$,

output an element of $n\text{-best}(\hat{\mu}, \trianglelefteq)(D_G(w))$. □

The following observation is apparent from the definitions of sort, take, and n-best.

Observation 5.4. $\text{take}(n) \circ \text{sort}(f, \trianglelefteq)\big(B'\big) \in n\text{-best}(f, \trianglelefteq)(B')$
for every set B, subset $B' \subseteq B$, function $f\colon B \to \mathscr{B}$, $n \in \mathbb{N}$, and partial order $\trianglelefteq$ on $\mathscr{B}$. ■

5.2 *Specification of the CS parser*

Given an $\mathscr{A}$-weighted MCFG G, we construct the regular language $R(G_{\mathbb{B}})$, the $\mathscr{A}$-weighted homomorphism $h = \text{weights}_G \circ \text{hom}_{G_{\mathbb{B}}}$, and the congruence multiple Dyck language $mD_{\mathbf{c}}(G_{\mathbb{B}})$. Then $L(G) = h\big(R(G_{\mathbb{B}}) \cap mD_{\mathbf{c}}(G_{\mathbb{B}})\big)$. In a more procedural view on this representation of G, we might say that one obtains words (with weights) in $L(G)$ by (i) generating words in $R(G_{\mathbb{B}})$ (called *candidates*), (ii) discarding the candidates that are not in $mD_{\mathbf{c}}(G_{\mathbb{B}})$, and (iii) applying h to the remaining candidates.

Recall that there is a bijection between D_G and $R(G_{\mathbb{B}}) \cap mD_{\mathbf{c}}(G_{\mathbb{B}})$ (Corollary 3.13). Now let w be the word we want to parse. Furthermore, let $R_{h,w}$ be the set of words that h maps to w. Then there is a bijection between $R_{h,w} \cap R(G_{\mathbb{B}}) \cap mD_{\mathbf{c}}(G_{\mathbb{B}})$ and $D_G(w)$. Our strategy to compute the n best derivations is to enumerate the words in $R_{h,w} \cap R(G_{\mathbb{B}})$ in the order given by the weights defined by the homomorphism h and then checking whether each word is in $mD_{\mathbf{c}}(G_{\mathbb{B}})$ until we have found n words. However, this approach will have to be refined to ensure termination of the parsing algorithm.

First we show that the set $R_{h,w}$ of words mapped by h to w is regular.

Definition 5.5. Let $h\colon \Sigma^* \to (\Gamma^* \to \mathscr{A})$ be an $\mathscr{A}$-weighted alphabetic homomorphism and $\gamma_1, \ldots, \gamma_n \in \Gamma$. Furthermore, let $\Sigma_\varepsilon = \{\sigma \in \Sigma \mid h(\sigma) = \mu.\varepsilon \text{ for some } \mu \in \mathscr{A}\}$, and for every $\gamma \in \Gamma$, define $\sigma_\gamma \in \Sigma$ such that $h(\sigma_\gamma) = \mu.\gamma$ for some $\mu \in \mathscr{A}$. The *domain language of h with respect to $\gamma_1 \cdots \gamma_n$*, is the language

$$R_{h,\gamma_1\cdots\gamma_n} = \{u_0\sigma_{\gamma_1}u_1 \cdots \sigma_{\gamma_n}u_n \mid u_0, \ldots, u_n \in \Sigma_\varepsilon^*\}. \qquad \square$$

Lemma 5.6. $R_{h,w}$ is a regular language for every $\mathscr{A}$-weighted alphabetic homomorphism $h\colon \Sigma^* \to (\Gamma^* \to \mathscr{A})$ and word $w \in \Gamma^*$.

Proof. Since $\{\sigma_{\gamma_1}\}, \ldots, \{\sigma_{\gamma_n}\}$, and Σ_ε are finite sets, $R_{h,w} = \Sigma_\varepsilon^* \cdot \{\sigma_{\gamma_1}\} \cdot \Sigma_\varepsilon^* \cdot \ldots \cdot \{\sigma_{\gamma_n}\} \cdot \Sigma_\varepsilon^*$, and because finite sets are recognisable (Hopcroft and Ullman 1969, Theorem 3.7) and recognisable languages are closed under language concatenation (Hopcroft and Ullman 1969, Theorem 3.8) and Kleene-star (Hopcroft and Ullman 1969, Theorem 3.9), we have that $R_{h,w}$ is regular. ■

Example 5.7 (Examples 3.7 and 3.11 continued)**.** Figure 6 shows a deterministic FSA $\mathscr{M}_{h,ac}$ that recognises $R_{h,ac}$. $\square$

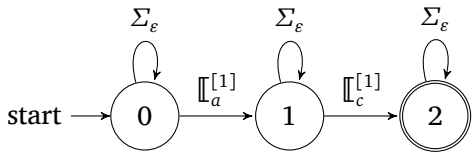

Figure 6:
Deterministic FSA $\mathscr{M}_{h,ac}$ that recognises $R_{h,ac}$ (cf. Example 5.7)

Definition 5.8. Let $\mathscr{M} = (Q, \Delta, q_\mathrm{i}, F, T)$ be an FSA and $h\colon \Delta^* \to (\Gamma^* \to \mathscr{A})$ be an $\mathscr{A}$-weighted homomorphism. The *set of harmful loops in $\mathscr{M}$ with respect to h* is the set of all runs $q_0 u_1 q_1 \cdots u_k q_k$ in $\mathscr{M}$ where $q_1, \ldots, q_k$ are pairwise disjoint elements of Q, $q_0 = q_k$, $u_1, \ldots, u_k \in \Delta^*$, and $h(u_1) = \ldots = h(u_k) = 1.\varepsilon$. □

When examining Example 3.11 and Figure 5, we see that there are seven harmful loops in $\mathscr{M}(G_\mathbb{B})$ with respect to h: the five self-loops of T, the loop between T and $A^{[2]}$, and the loop between T and $B^{[2]}$. Harmful loops are problematic for the termination of a parsing algorithm since they cause an infinite set of candidates that are not distinguishable by their image under h. When generating the sequence of candidates for our parsing algorithm, such a situation creates the contingency of never producing a candidate that is accepted by $mD_\mathbf{c}(G_\mathbb{B})$ even if one exists. To allow our parsing algorithm to solve the above problem, we will

(i) only admit a restricted form of weighted MCFGs and
(ii) require each value in the domain of $\mathscr{A}$ (except 0 and 1) to be viewed as a product of arbitrarily many smaller values from the domain of $\mathscr{A}$.

Restricted weighted MCFG

Definition 5.9. An $\mathscr{A}$-weighted MCFG $G = (N, \Delta, S, P, \mu)$ is called *restricted* if there do not exist a subderivation d in G and a position $\pi = n_1 \cdots n_k \in \mathrm{pos}(d)$ (where $n_1, \ldots, n_k \in \mathbb{N}$) such that $d(\varepsilon) = d(\pi)$, $d(\varepsilon), d(n_1), d(n_1 n_2), \ldots, d(n_1 \cdots n_k)$ are pairwise different, and $\mu(d(\varepsilon)) = \mu(d(n_1)) = \mu(d(n_1 n_2)) = \ldots = \mu(d(n_1 \cdots n_k)) = 1$. □

Restricted weighted MCFG are strictly less powerful than (unrestricted) weighted MCFG, as the next example shows.

Example 5.10. Let us consider an arbitrary $\mathbb{B}$-weighted MCFG G and let m be the number of rules in G. Assume that $L(G)$ is not finite. Then there are subderivations in G of arbitrary height. It is clear that every subderivation d in G with a height greater than $m + 1$ must have a position $\pi \in \mathrm{pos}(d)$ such that $d(\varepsilon) = d(\pi)$. Then, since G has weights from $\mathbb{B}$, we know that 1 is assigned to every production in G and thus G is not restricted. □

Restricted weighted MCFGs are still useful in practice, as the following two observations show.

Definition 5.11. An $\mathscr{A}$-weighted MCFL is called *proper* if $\mathscr{A}$ is the probability semiring, the Viterbi semiring, or one of the algebras Pr_1 or Pr_2 (cf. Example 2.1) and for each non-terminal A the sum (using the usual addition in $\mathbb{R}$) of the weights of all productions with left-hand side A is 1. □

Observation 5.12. Every proper weighted MCFG is restricted.

Proof. Assume that G is proper but not restricted. Then there is a subderivation d in G and a position $\pi \in \mathrm{pos}(d)$ such that the weights of all productions along the path from the root to position π in d are 1 and $d(\varepsilon) = d(\pi) = \rho$. All productions along the path from the root to position π are unique for their respective left-hand side non-terminals since G is probabilistic. This means that every subderivation d' starting from ρ has the position π and $\rho = d'(\varepsilon) = d'(\pi) = d(\varepsilon) = d(\pi)$. But then $\{\varepsilon, \pi, \pi\pi, \pi\pi\pi, \ldots\} \subseteq \mathrm{pos}(d)$ and hence d is not a (finite) term, which contradicts our definition of a subderivation. ■

If we extract a weighted MCFG from a corpus and assign the weights by maximum-likelihood estimation (as for example in Kallmeyer and Maier 2013, p. 107), then we will get a weighted MCFG that is proper and therefore restricted.

The next observation allows us to enrich the weight structure of a $\mathbb{B}$-weighted MCFG to make it suitable for CS-parsing.

Observation 5.13. For every $\mathbb{B}$-weighted MCFG G, there is a restricted weighted MCFG G' such that $\mathrm{supp}(L(G)) = \mathrm{supp}(L(G'))$.

Proof. This can be done by assigning to every $w \in \mathrm{supp}(G)$, the size (i.e. number of productions) of its smallest derivation. To achieve that, we choose the tropical semiring as weight algebra for G', use the productions from G, and give every production the weight 1. Then no production in G' has the semiring-1 (which is 0 for the tropical semiring) as its weight and G' is restricted. ■

Factorisable weight structures

Definition 5.14. Let $\mathscr{A}$ be a complete commutative strong bimonoid and $\trianglelefteq$ be a partial order on $\mathscr{A}$. We say that $\mathscr{A}$ is $\trianglelefteq$-*factorisable* if for

every $a \in \mathscr{A} \setminus \{0,1\}$ and natural number $k \geq 2$, there are $a_1, \ldots, a_k \in \mathscr{A} \setminus \{0,1\}$ such that $a_1 \lhd a, \ldots, a_k \lhd a$ and $a_1 \cdot \ldots \cdot a_k = a$. We then call $a_1 \cdot \ldots \cdot a_k$ a $(\trianglelefteq, k)$-*factorisation of* a. □

Some of the complete commutative strong bimonoids mentioned in Example 2.1 are $\trianglelefteq$-factorisable for some suitable partial order $\trianglelefteq$, as Table 3 shows. The examples where multiplication is idempotent, however, have no suitable partial order since $a \cdot a = a$ contradicts $\cdot$ being increasing (in particular $a \lhd a \cdot a$).

Table 3: List of some complete commutative strong bimonoids $\mathscr{A}$ from Example 2.1 together with a partial order $\trianglelefteq$ that $\mathscr{A}$ respects and a $(\trianglelefteq, k)$-factorisation.

example algebra $(\mathscr{A}, +, \cdot, 0, 1)$	partial order $\trianglelefteq$	$(\trianglelefteq, k)$-factorisation of $a \in \mathscr{A} \setminus \{0,1\}$
Viterbi semiring $([0,1], \max, \cdot, 0, 1)$	$\geq$	$\underbrace{\sqrt[k]{a} \cdot \ldots \cdot \sqrt[k]{a}}_{k \text{ times}}$
tropical semiring $(\mathbb{R}_{\geq 0} \cup \{\infty\}, \min, +, \infty, 0)$	$\leq$	$\underbrace{a/k + \ldots + a/k}_{k \text{ times}}$
arctic semiring $(\mathbb{R}_{\geq 0} \cup \{-\infty\}, \max, +, -\infty, 0)$	$\leq$	$\underbrace{a/k + \ldots + a/k}_{k \text{ times}}$
$\mathrm{Pr}_1 = ([0,1], \oplus_1, \cdot, 0, 1)$ and $\mathrm{Pr}_2 = ([0,1], \oplus_2, \cdot, 0, 1)$	$\geq$	$\underbrace{\sqrt[k]{a} \cdot \ldots \cdot \sqrt[k]{a}}_{k \text{ times}}$

For the probability semiring $\mathrm{Pr} = (\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$ monotonicity and increasingness contradict each other. To show this we first assume that Pr would respect some partial order $\trianglelefteq$. From $1/2 \cdot 1/2 = 1/4$ and $2 \cdot 2 = 4$ follows $1/2 \lhd 1/4$ and $2 \lhd 4$ since $\cdot$ is increasing. By monotonicity then follows $2 \cdot 1/2 \trianglelefteq 4 \cdot 1/2 \trianglelefteq 4 \cdot 1/4$ and hence $1 \trianglelefteq 2 \trianglelefteq 1$. But $\trianglelefteq$ is a partial order and thus antisymmetric.

5.4 *A CS parsing algorithm*

For the remainder of this article let $\mathscr{A}$ be a complete commutative strong bimonoid and $\trianglelefteq$ be a partial order on $\mathscr{A}$ such that $\mathscr{A}$ respects $\trianglelefteq$ and is $\trianglelefteq$-factorisable. Also, we will require our weighted MCFGs to be restricted.

To solve the problem stated in the beginning of Section 5.3, we define a modified generator language and a modified weight function to replace R and h from the CS-theorem (cf. Theorem 3.12).

The modified generator language

Consider an MCFG G with non-terminals N. Then, intuitively, $\mathscr{M}(G)$ has two kinds of states:

(i) For every non-terminal $A \in N$ and every $j \in [\text{fan-out}(A)]$ there is a state $A^{[j]}$ that signifies that the automaton *is about to process* the j-th component of a string tuple generated by G with A.

(ii) There is a state T that signifies that the automaton *just finished processing* some component of a string tuple generated by G with some non-terminal.

We split state T from $\mathscr{M}(G)$ up to formalise the following intuition:

(ii') For every non-terminal $A \in N$ and every $j \in [\text{fan-out}(A)]$ there is a state $\bar{A}^{[j]}$ that signifies that the automaton *just finished processing* the j-th component of a string tuple generated by G with A.

Definition 5.15. Let $G = (N, \Delta, S, R)$ be an MCFG. The *modified generator automaton with respect to* G, denoted by $\mathscr{M}'(G)$, is the finite state automaton $(Q \cup \bar{Q}, \Sigma, S^{[1]}, \{\bar{S}^{[1]}\}, \tau)$ where Σ is the generator alphabet with respect to G, $Q = \{A^{[j]} \mid A \in N, j \in [\text{fan-out}(A)]\}$, $\bar{Q} = \{\bar{A}^{[j]} \mid A \in N, j \in [\text{fan-out}(A)]\}$, and τ is the smallest set such that for every rule

$$\rho = A \to [u_1^0 y_1^1 u_1^1 \cdots y_1^{n_1} u_1^{n_1}, \ldots, u_s^0 y_s^1 u_s^1 \cdots y_s^{n_s} u_s^{n_s}](B_1, \ldots, B_m)$$

in R we have that

(i) $\big(A^{[j]},\ [\![^{[j]}_\rho \tilde{u}_j^0]\!]^{[j]}_\rho,\ \bar{A}^{[j]}\big) \in \tau$ for every $j \in [s]$ with $n_j = 0$,

(ii) $\big(A^{[j]},\ [\![^{[j]}_\rho \tilde{u}_j^0 [\![^{[\ell]}_{\rho,i},\ B_i^{[\ell]}\big) \in \tau$ for every $i \in [m]$, $j \in [s]$, and $\ell \in [\text{fan-out}(B_i)]$ with $n_j \neq 0$ and $y_j^1 = x_i^\ell$,

(iii) $\big(\bar{B}_i^{[\ell]},\ ]\!]^{[\ell]}_{\rho,i} \tilde{u}_j^{\kappa-1} [\![^{[\ell']}_{\rho,i'},\ B_{i'}^{[\ell']}\big) \in \tau$ for every $i, i' \in [m]$, $j \in [s]$, $\kappa \in [n_j]$, $\ell \in [\text{fan-out}(B_i)]$, $\ell' \in [\text{fan-out}(B_{i'})]$ with $n_j \neq 0$, $y_j^{\kappa-1} = x_i^\ell$, $y_j^\kappa = x_{i'}^{\ell'}$, and

(iv) $\big(\bar{B}_i^{[\ell]},\ ]\!]^{[\ell]}_{\rho,i} \tilde{u}_j^{n_j}]\!]^{[j]}_\rho,\ \bar{A}^{[j]}\big) \in \tau$ for every $i \in [m]$, $j \in [s]$, and $\ell \in [\text{fan-out}(B_i)]$ with $n_j \neq 0$ and $y_j^{n_j} = x_i^\ell$.

The *modified generator language with respect to* G is $R'(G) = L(\mathscr{M}'(G))$. □

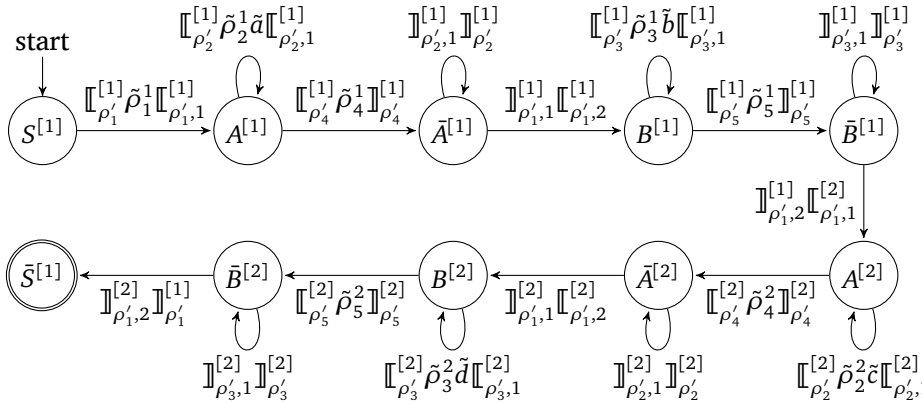

Figure 7: Modified generator automaton $\mathscr{M}'(G')$ with respect to G' from Example 3.4

Lemma 5.16. $R(G) \cap mD_{\mathrm{c}}(G) = R'(G) \cap mD_{\mathrm{c}}(G)$ for every MCFG G.

Proof. Let $G = (N, \Delta, S, P)$ and Σ be the generator alphabet with respect to G.

($\supseteq$) For this we show that $R(G) \supseteq R'(G)$. Let $q_0 u_1 q_1 \cdots u_m q_m$ be a successful run in $\mathscr{M}'(G)$. We define the string $v = t(q_0)u_1 t(q_1)\cdots u_m t(q_m)$ where

$$t(q) = \begin{cases} q & \text{if } q = A^{[j]} \in Q \text{ for some } A \in N \text{ and } j \in \text{fan-out}(A) \\ T & \text{otherwise.} \end{cases}$$

Clearly for every transition (q, u, q') in $\mathscr{M}'(G)$, there is a transition $(t(q), u, t(q'))$ in $\mathscr{M}(G)$. Since $\bar{S}^{[1]}$ is the final state in $\mathscr{M}'(G)$ and $t(\bar{S}^{[1]})$ is a final state in $\mathscr{M}(G)$, we have that v is a successful run in $\mathscr{M}(G)$.

($\subseteq$) Since $R(G) \cap mD_{\mathrm{c}}(G)$ is in bijection with D_G (Corollary 3.9), it suffices to show that $\mathrm{yield}(D_G) \supseteq \mathrm{hom}_G(R'(G))$. We prove the following for every $A \in N$ and $d \in D_G(A)$ by induction on d:

> Let $\mathrm{yield}(d) = (u_1, \ldots, u_m)$. There are $v_1, \ldots, v_m \in \Sigma^*$ such that $\mathscr{M}'(G)$ recognises v_i from $A^{[i]}$ to $\bar{A}^{[i]}$ and $\mathrm{hom}_G(v_i) = u_i$ for every $i \in [m]$.

This statement implies the claim.

Induction base: Let $d = \rho = A \to [u_1, \ldots, u_m]() \in P$. Then for every $i \in [m]$, there is a transition $(A^{[i]}, \llbracket^{[i]}_{\rho} \tilde{u}_i \rrbracket^{[i]}_{\rho}, \bar{A}^{[i]})$ in $\mathscr{M}'(G)$. Clearly, $\mathrm{hom}_G(\llbracket^{[i]}_{\rho} \tilde{u}_i \rrbracket^{[i]}_{\rho}) = u_i$ and $\mathscr{M}'(G)$ recognises $\llbracket^{[i]}_{\rho} \tilde{u}_i \rrbracket^{[i]}_{\rho}$ from $A^{[i]}$ to $\bar{A}^{[i]}$.

Step: Let $d = \rho(d_1, \dots, d_k)$ with $\rho = A \to [u_1, \dots, u_m](B_1, \dots, B_k)$ and $m_i = \text{fan-out}(B_i)$ for every $i \in [k]$. By induction hypothesis there are $v_1^i, \dots, v_{m_i}^i$ for every $i \in [k]$ such that $(\hom_G(v_1^i), \dots, \hom_G(v_{m_i}^i)) = \text{yield}(d_i)$ and $\mathscr{M}'(G)$ recognises v_j^i from $B_i^{[j]}$ to $\bar{B}_i^{[j]}$ for every $j \in [m]$. By the existence of ρ in P and Definition 5.15, we can construct runs in $\mathscr{M}'(G)$ from $A^{[1]}$ to $\bar{A}^{[1]}$, ..., $A^{[m]}$ to $\bar{A}^{[m]}$, recognising $v_1, \dots, v_m$, respectively such that $(\hom_G(v_1), \dots, \hom_G(v_m)) = \text{yield}(d)$. ■

The modified weight function

Examining $\mathscr{M}'(G_{\mathbb{B}})$ from Figure 7 together with h from Example 3.11 we notice that there are still four harmful loops: the self-loops of $A^{[1]}$, $B^{[1]}$, $A^{[2]}$, and $B^{[2]}$. Therefore, we will define a function $\text{wt}_{G,\trianglelefteq}$ from strings of parentheses to $\mathscr{A}$ that assigns a weight different from 1 to the labels along those loops, but still computes the same weights as h on the subset $mD_{\mathrm{c}}(G_{\mathbb{B}})$. Intuitively, we take the weight attached to a symbol $[\![_\rho^{[1]}$ (for some rule ρ of fan-out s) and distribute it along the $2{\cdot}s$ symbols $[\![_\rho^{[1]},]\!]_\rho^{[1]}, \dots, [\![_\rho^{[s]},]\!]_\rho^{[s]}$ using the fact that $\mathscr{A}$ is $\trianglelefteq$-factorisable.

Definition 5.17. Let $\mathscr{A}$ be a complete commutative strong bimonoid, $\trianglelefteq$ be a partial order on $\mathscr{A}$, $\mathscr{A}$ be $\trianglelefteq$-factorisable, and $G = (N, \Delta, S, P, \mu)$ be an $\mathscr{A}$-weighted k-MCFG. Furthermore, let Σ be the generator alphabet with respect to G. For every $\rho = A \to [u_1, \dots, u_s](B_1, \dots, B_k) \in P$, we set $\rho' = A \to [\rho^1 u_1, \dots, \rho^s u_s](B_1, \dots, B_k)$ and fix values $a_1^\rho, \dots, a_{2\cdot s}^\rho \in \mathscr{A} \setminus \{0\}$ such that

- if $\mu(\rho) \neq 1$ then $a_1^\rho, \dots, a_{2\cdot s}^\rho$ are not 1, are smaller or equal to $\mu(\rho)$ with respect to $\trianglelefteq$, and $a_1^\rho \cdot \ldots \cdot a_{2\cdot s}^\rho = \mu(\rho)$; or
- if $\mu(\rho) = 1$ then $a_1^\rho = \ldots = a_{2\cdot s}^\rho = 1$.

The *weight function with respect to G and $\trianglelefteq$* is the function $\text{wt}_{G,\trianglelefteq} \colon \Sigma^* \to \mathscr{A}$ defined for every $u_1, \dots, u_k \in \Sigma$ by

$$\text{wt}_{G,\trianglelefteq}(u_1 \cdots u_k) = \text{wt}'_{G,\trianglelefteq}(u_1) \cdot \ldots \cdot \text{wt}'_{G,\trianglelefteq}(u_k)$$

where $\text{wt}'_{G,\trianglelefteq} \colon \Sigma \to \mathscr{A}$ is given by

$$\text{wt}'_{G,\trianglelefteq}(\sigma) = \begin{cases} a_{2\cdot i-1}^\rho & \text{if } \sigma \text{ is of the form } [\![_\rho^{[i]} \\ a_{2\cdot i}^\rho & \text{if } \sigma \text{ is of the form }]\!]_\rho^{[i]} \\ 1 & \text{otherwise.} \end{cases}$$ □

Example 5.18 (Example 2.5 continued)**.** First, we calculate factorisations of the weights in G as shown in Table 3 for Pr_2:

$$\begin{aligned}
&\text{for } \rho_2 \text{ and } \rho_4\text{:} && 1/2 = \sqrt[4]{1/2} \cdot \sqrt[4]{1/2} \cdot \sqrt[4]{1/2} \cdot \sqrt[4]{1/2}\\
&\text{for } \rho_3\text{:} && 1/3 = \sqrt[4]{1/3} \cdot \sqrt[4]{1/3} \cdot \sqrt[4]{1/3} \cdot \sqrt[4]{1/3}\\
&\text{for } \rho_5\text{:} && 2/3 = \sqrt[4]{2/3} \cdot \sqrt[4]{2/3} \cdot \sqrt[4]{2/3} \cdot \sqrt[4]{2/3}
\end{aligned}$$

Then $\mathrm{wt}_{G,\trianglelefteq}$ is given as follows:

$$\mathrm{wt}_{G,\geq}(\sigma) = \begin{cases} \sqrt[4]{1/2} & \text{if } \sigma \in \{[\![^{[j]}_{\rho'},]\!]^{[j]}_{\rho'} \mid \rho' \in \{\rho'_2, \rho'_4\}, j \in [2]\},\\ \sqrt[4]{1/3} & \text{if } \sigma \in \{[\![^{[j]}_{\rho'_3},]\!]^{[j]}_{\rho'_3} \mid j \in [2]\},\\ \sqrt[4]{2/3} & \text{if } \sigma \in \{[\![^{[j]}_{\rho'_5},]\!]^{[j]}_{\rho'_5} \mid j \in [2]\},\\ 1 & \text{otherwise.} \end{cases}$$ □

Now we examine the FSA $\mathscr{M}'(G_{\mathbb{B}})$ from Figure 7 again, but this time we use the weight function $\mathrm{wt}_{G,\trianglelefteq}$ from Example 5.18 instead of the weights given by the homomorphism h in Example 3.11. For this let $h' : (\Sigma \cup \bar{\Sigma})^* \to (\Delta^* \to \mathscr{A})$ be the $\mathscr{A}$-weighted homomorphism defined by $h'(\sigma) = \mathrm{wt}_{G,\trianglelefteq}(\sigma).u_\sigma$ where $h(\sigma) = \mu_\sigma.u_\sigma$ for every $\sigma \in \Sigma \cup \bar{\Sigma}$. Then there are no more harmful loops in $\mathscr{M}'(G_{\mathbb{B}})$ with respect to h'.

Lemma 5.19. Let $G = (N, \Delta, S, P, \mu)$ be a restricted $\mathscr{A}$-weighted MCFG that only has productive non-terminals and $\trianglelefteq$ be a partial order on $\mathscr{A}$ such that $\mathscr{A}$ is $\trianglelefteq$-factorisable. Furthermore, let $h' : (\Sigma \cup \bar{\Sigma})^* \to (\Delta^* \to \mathscr{A})$ be the $\mathscr{A}$-weighted homomorphism defined by $h'(\sigma) = \mathrm{wt}_{G,\trianglelefteq}(\sigma).u_\sigma$ where $(\mathrm{weights}_G \circ \mathrm{hom}_{G_{\mathbb{B}}})(\sigma) = \mu_\sigma.u_\sigma$ for every $\sigma \in \Sigma \cup \bar{\Sigma}$. Then $\mathscr{M}'(G_{\mathbb{B}})$ has no harmful loops with respect to h'.

Proof. We show the claim by contradiction. For this assume that $q_0 u_1 q_1 \cdots u_k q_k$ is a harmful loop in $\mathscr{M}'(G_{\mathbb{B}})$ with respect to h' where $q_0, \ldots, q_k$ are states and $u_1, \ldots, u_k \in (\Sigma \cup \bar{\Sigma})^*$. Then $q_0 = q_k$ and $\mathrm{wt}_{G,\trianglelefteq}(u_1) = \ldots = \mathrm{wt}_{G,\trianglelefteq}(u_k) = 1$. We now distinguish two cases:

(Case 1) Let $q_0 = q_k \in Q$ and $I = \{i_0, \ldots, i_m\}$ be the maximal subset of $\{0, \ldots, k\}$ (with $i_0 < \cdots < i_m$) such that

(i) $i_0 = 0$ and $i_m = k$,

(ii) for every $i \in I$ we have that q_i is of the form $B_i^{[j_i]}$ for some $B_i \in N$ and $j_i \in [\text{fan-out}(B_i)]$, and

(iii) for every $\kappa \in [m]$ we have that B_{i_κ} occurs on the right-hand side of $\rho_{i_{\kappa-1}}$ where $[\![^{[j_{i_{\kappa-1}}]}_{\rho_{i_{\kappa-1}}}$ is read in the transition that leaves $q_{i_{\kappa-1}}$.

Since every non-terminal in G is productive there is a derivation d and a position $\pi = n_1 \cdots n_m$ in d such that $n_1, \ldots, n_m \in \mathbb{N}$ and $d(\varepsilon) = \rho_{i_1}$, $d(n_1) = \rho_{i_2}$, ..., $d(n_1 \cdots n_{m-1}) = \rho_{i_m}$, and $d(n_1 \cdots n_m) = \rho_{i_1}$. For every $i \in I$ we know that $\mu(\rho_i) = 1$ since $\mathrm{wt}_{G,\trianglelefteq}([\![^{[j_i]}_{\rho_i}) = 1$ and by Definition 5.17. This contradicts G being restricted.

(Case 2) Let $q_0 = q_k \in \bar{Q}$ and $I = \{i_0, \ldots, i_m\}$ be the maximal subset of $\{0, \ldots, k\}$ (with $i_m < \cdots < i_0$) such that

(i) $i_m = 0$ and $i_0 = k$,

(ii) for every $i \in I$ we have that q_i is of the form $\bar{B}_i^{[j_i]}$ for some $B_i \in N$ and $j_i \in [\text{fan-out}(B_i)]$, and

(iii) for every $\kappa \in [m]$ we have that B_{i_κ} occurs on the right-hand side of $\rho_{i_{\kappa-1}}$ where $]\!]^{[j_{i_{\kappa-1}}]}_{\rho_{i_{\kappa-1}}}$ is read in the transition that reaches $q_{i_{\kappa-1}}$.

Since every non-terminal in G is productive there is a derivation d and a position $\pi = n_1 \cdots n_m$ in d such that $n_1, \ldots, n_m \in \mathbb{N}$ and $d(\varepsilon) = \rho_{i_m}$, $d(n_1) = \rho_{i_{m-1}}$, ..., $d(n_1 \cdots n_{m-1}) = \rho_{i_1}$, and $d(n_1 \cdots n_m) = \rho_{i_m}$. For every $i \in I$ we know that $\mu(\rho_i) = 1$ since $\mathrm{wt}_{G,\trianglelefteq}(]\!]^{[j_i]}_{\rho_i}) = 1$ and by Definition 5.17. This contradicts G being restricted. ■

Lemma 5.20. Let $G = (N, \Delta, S, P, \mu)$ be an $\mathscr{A}$-weighted MCFG and $\trianglelefteq$ a partial order on $\mathscr{A}$. Then $h = \mathrm{weights}_G \circ \mathrm{hom}_{G_{\mathbb{B}}}$ assigns the same weight to each word in $mD_{\mathrm{c}}(G_{\mathbb{B}}) \cap R'(G_{\mathbb{B}})$ as $\mathrm{wt}_{G,\trianglelefteq}$.

Proof. Let $\rho \in P$ be an arbitrary production, $m = \text{fan-out}(\rho)$, and $\mathrm{wt}_{G,\trianglelefteq}([\![^{[1]}_{\rho}) = a_{2\cdot i-1}$ and $\mathrm{wt}_{G,\trianglelefteq}(]\!]^{[1]}_{\rho}) = a_{2\cdot i}$ for every $i \in [m]$. By the definition of $R'(G_{\mathbb{B}})$, we know that symbols of the forms $[\![^{[i]}_{\rho}$, $[\![^{[1]}_{\rho^i}$, and $[\![^{[1]}_{\rho^i}$ occur only as a sequence $[\![^{[i]}_{\rho}[\![^{[1]}_{\rho^i}]\!]^{[1]}_{\rho^i}$ (see the construction in Lemma 3.3 and items (i) and (ii) in Definition 5.15). By the definition of the cancellation rule, we also know that for every symbol $[\![^{[i]}_{\rho}$ there must occur corresponding symbols $[\![^{[1]}_{\rho}, \ldots, [\![^{[i-1]}_{\rho}, [\![^{[i+1]}_{\rho}, \ldots, [\![^{[m]}_{\rho}$ and $]\!]^{[1]}_{\rho}, \ldots,]\!]^{[m]}_{\rho}$ in $mD_{\mathrm{c}}(G_{\mathbb{B}})$. Thus in the set $mD_{\mathrm{c}}(G_{\mathbb{B}}) \cap R'(G_{\mathbb{B}})$ we have that $[\![^{[1]}_{\rho}$ occurs iff all the corresponding symbols $[\![^{[2]}_{\rho}, \ldots, [\![^{[m]}_{\rho}$ and $]\!]^{[1]}_{\rho}, \ldots,]\!]^{[m]}_{\rho}$ occur.

Then by the construction of $\mathrm{wt}_{G,\trianglelefteq}$ it follows that

$$\underbrace{\mathrm{wt}_{G,\trianglelefteq}([\![_\rho^{[1]}])}_{=a_1} \cdot \underbrace{\mathrm{wt}_{G,\trianglelefteq}([\![_{\rho^1}^{[1]}]\!]_{\rho^1}^{[1]})}_{=1} \cdot \underbrace{\mathrm{wt}_{G,\trianglelefteq}(]\!]_\rho^{[1]})}_{=a_2} \cdot \ldots \cdot \underbrace{\mathrm{wt}_{G,\trianglelefteq}([\![_\rho^{[m]}])}_{=a_{2m-1}} \cdot \underbrace{\mathrm{wt}_{G,\trianglelefteq}(]\!]_\rho^{[m]})}_{=a_{2m}}$$

is $\mu(\rho)$ and thus exactly the weight assigned to those symbols by h. ■

The parser

Definition 5.21. Let G be an $\mathscr{A}$-weighted MCFG over Δ, $n \in \mathbb{N}$, and $\trianglelefteq$ be a partial order on $\mathscr{A}$. The *n-best CS parser with respect to G and $\trianglelefteq$*, denoted by CS-parse$(G, n, \trianglelefteq)$, is a function from Δ^* to D_G^* that assigns for every $w \in \Delta^*$ the value

$$\mathrm{take}(n) \circ \mathrm{map}(\mathit{toDeriv} \circ \mathrm{hom}_{G_{\mathbb{B}}}) \circ \mathrm{filter}(mD_{\mathbf{c}}(G_{\mathbb{B}})) \circ \mathrm{sort}(\mathrm{wt}_{G,\trianglelefteq}, \trianglelefteq)\big(R'(G_{\mathbb{B}}) \cap R_{h,w}\big)$$

where $h = \mathrm{weights}_G \circ \mathrm{hom}_{G_{\mathbb{B}}}$. □

Theorem 5.22. CS-parse$(G, n, \trianglelefteq)(w) \in n$-best$(\hat{\mu}, \trianglelefteq)(D_G(w))$ for every $\mathscr{A}$-weighted MCFG $G = (N, \Delta, S, P, \mu)$, $n \in \mathbb{N}$, partial order $\trianglelefteq$ that respects $\mathscr{A}$, and word $w \in \Delta^*$.

Proof. Let $h = \mathrm{weights}_G \circ \mathrm{hom}_{G_{\mathbb{B}}}$. We prove the claim by verifying the four conditions from Definition 5.2.

(i) This follows from the definition of sort and the bijection between $R(G_{\mathbb{B}}) \cap mD_{\mathbf{c}}(G_{\mathbb{B}})$ and D_G (Corollary 3.13).

(ii) $d_1, \ldots, d_k$ are pairwise different since $u_1, \ldots, u_k$ are pairwise different and $R(G_{\mathbb{B}}) \cap mD_{\mathbf{c}}(G_{\mathbb{B}})$ and D_G stand in bijection (Corollary 3.13).

(iii) Let $(d_1, \ldots, d_k) =$ CS-parse$(G, n, \trianglelefteq)(w)$ and $(u_1, \ldots, u_k)$ be a prefix of $\mathrm{filter}(mD_{\mathbf{c}}(G_{\mathbb{B}})) \circ \mathrm{sort}(\mathrm{wt}_{G,\trianglelefteq}, \trianglelefteq)\big(R(G_{\mathbb{B}}) \cap R_{h,w}\big)$. Due to the definition of filter, there must be a natural number m such that $(v_1, \ldots, v_m)$ is a prefix of $\mathrm{sort}(\mathrm{wt}_{G,\trianglelefteq}, \trianglelefteq)\big(R(G_{\mathbb{B}}) \cap R_{h,w}\big)$ for some $v_1, \ldots, v_m \in (\Sigma \cup \bar{\Sigma})^*$ and $u_1, \ldots, u_k$ occur in that order in $v_1, \ldots, v_m$. By the definition of sort, we have that $\mathrm{wt}_{G,\trianglelefteq}(v_1) \trianglelefteq \ldots \trianglelefteq \mathrm{wt}_{G,\trianglelefteq}(v_m)$. Since $u_1, \ldots, u_k$ occur in that order in $v_1, \ldots, v_m$, it follows that $\mathrm{wt}_{G,\trianglelefteq}(u_1) \trianglelefteq \ldots \trianglelefteq \mathrm{wt}_{G,\trianglelefteq}(u_k)$. By Lemma 5.20 we obtain $\widehat{\mu} \circ \mathit{toDeriv} \circ \mathrm{hom}_{G_{\mathbb{B}}}(u_1) \trianglelefteq \ldots \trianglelefteq \widehat{\mu} \circ \mathit{toDeriv} \circ \mathrm{hom}_{G_{\mathbb{B}}}(u_k)$ and by the definition of map we get $\widehat{\mu}(d_1) \trianglelefteq \ldots \trianglelefteq \widehat{\mu}(d_k)$.

(iv) There is no $d \in D_G(w)$ with $\widehat{\mu}(d) \triangleleft \widehat{\mu}(d_k)$ since, by definition of sort, there is no $u \in R(G_{\mathbb{B}}) \cap mD_{\mathbf{c}}(G_{\mathbb{B}})$ with $\mathrm{wt}_{G,\trianglelefteq}(u) \triangleleft \mathrm{wt}_{G,\trianglelefteq}(u_k)$. ■

In addition to just implementing CS-parse$(G, n, \trianglelefteq)$, we add a threshold θ for the weight of the found derivations. Then our algorithm will only consider the candidates whose weight is less than or equal to θ. Without such a threshold, the parsing algorithm would not terminate if the intersection of $R'(G_{\mathbb{B}})$ and $R_{h,w}$ had an infinite language and w had less than n derivations in G. On the other hand, if w has a derivation in G with a weight above the threshold, our algorithm will not find it, however large we choose n. Thus the algorithm only approximates CS-parse$(G, n, \trianglelefteq)$.

Algorithm 3: Approximation of CS-parse using a threshold θ

Input: a restricted $\mathscr{A}$-weighted MCFG $G = (N, \Delta, S, P, \mu)$, a number $n \in \mathbb{N}$, a partial order $\trianglelefteq$ on $\mathscr{A}$ where $\trianglelefteq$ respects $\mathscr{A}$ and $\mathscr{A}$ is $\trianglelefteq$-factorisable, a threshold $\theta \in \mathscr{A}$ with $0 \trianglelefteq \theta$, and a word $w \in \Delta^*$
Output: the subsequence of derivations d in CS-parse$(G, n, \trianglelefteq)(w)$ with $\mu(d) \trianglelefteq \theta$

```
1  function CS-Parse(G, n, ⊴, θ, w)
2    let mD_c(Σ, 𝔓) = mD_c(G_𝔹)
3    let h = weights_G ∘ hom_{G_𝔹}
4    let parses = ε
5    while hasNextCandicate ∧ |parses| < n do
6      let u = nextCandidate
7      if isMember(Σ, 𝔓, u) then
8        append toDeriv(u) to the end of parses
9      end if
10   end while
11   return parses
12 end function

13 procedure hasNextCandicate
14   return whether some u ∈ R'(G_𝔹) ∩ R_{h,w} with wt_{G,⊴}(u) ⊴ θ was
     not yet considered
15 end procedure

16 procedure nextCandidate
17   from the elements of R'(G_𝔹) ∩ R_{h,w} previously not considered,
     return an element whose image under wt_{G,⊴} is smallest with
     respect to ⊴
18 end procedure
```

To achieve that our algorithm terminates, we require for our input that G is restricted and that $\mathscr{A}$ is $\trianglelefteq$-factorisable. Let us take a closer look at how Algorithm 3 works. Since $\mathrm{sort}(\mathrm{wt}_{G,\trianglelefteq}, \trianglelefteq)$ returns an infinite list, we have a procedure *nextCandidate* that computes only the next element in that list, *hasNextCandicate* returns whether there is such a next element. We utilise those two procedures to only compute the prefix of $\mathrm{sort}(\mathrm{wt}_{G,\trianglelefteq}, \trianglelefteq)(R'(G_{\mathbb{B}}) \cap R_{h,w})$ we need to obtain the first n parses. Since there are deterministic FSA to recognise both $R'(G_{\mathbb{B}})$ and $R_{h,w}$, the computation of $\mathrm{sort}(\mathrm{wt}_{G,\trianglelefteq}, \trianglelefteq)(R'(G_{\mathbb{B}}) \cap R_{h,w})$ amounts to enumerating the paths in a weighted labelled graph ordered by their weights[8] and then replacing each path by the unique word recognised along it, where we take the graph with its labels from a deterministic FSA recognising $R'(G_{\mathbb{B}}) \cap R_{h,w}$ and the weights from $\mathrm{wt}_{G,\trianglelefteq}$.

Proof of termination for Algorithm 3. Let $h = \mathrm{weights}_G \circ \mathrm{hom}_{G_{\mathbb{B}}}$ and M be the set of all weights assigned by $\mathrm{wt}_{G,\trianglelefteq}$ to each of the words read in the transitions along one pass of every loop in the product of $\mathscr{M}'(G_{\mathbb{B}})$ and $\mathscr{M}_{h,w}$. Then M contains neither 0 nor 1 because $\mathscr{M}'(G_{\mathbb{B}})$ has no harmful loops. Since $\cdot$ has arbitrarily large powers, we know that for every element $a \in M$, there is a natural number k such that $\theta \trianglelefteq a^k$. Let $\hat{k}$ be the maximum of all such k's. There are only finitely many runs in the product of $\mathscr{M}'(G_{\mathbb{B}})$ and $\mathscr{M}_{h,w}$ that contain every loop less than $\hat{k}$ times. Hence *hasNextCandicate* is false after a finite number of iterations of the **while**-loop (lines 6 to 11), and Algorithm 3 terminates. ■

Example 5.23 (Examples 2.5, 3.7, and 5.7 continued)**.** Consider the word $w = ac$. The product of $\mathscr{M}'(G_{\mathbb{B}})$ and $\mathscr{M}_{h,w}$ together with $\mathrm{wt}_{G,\trianglelefteq}$ is shown in Figure 8 (for the product construction see Hopcroft and Ullman 1979, after Theorem 3.3). It suffices to consider at most 8 candidates to find the (only) derivation of w in G, as is shown in Table 4. The candidates themselves are not shown, instead we see their weight, their corresponding path in the graphical representation of the product of $\mathscr{M}'(G_{\mathbb{B}})$ and $\mathscr{M}_{h,w}$, and whether they are in $mD_{\mathbf{c}}(G_{\mathbb{B}})$. Candidate 7 is exactly $\mathrm{toBrackets}(\rho'_1(\rho'_2(\rho'_4), \rho'_5))$. □

[8] This is described for example in Hoffman and Pavley 1959 for graphs with edge weights from the real numbers. However, their algorithm works also for complete commutative strong bimonoids $\mathscr{A}$ with a partial order on $\mathscr{A}$ that respects $\mathscr{A}$.

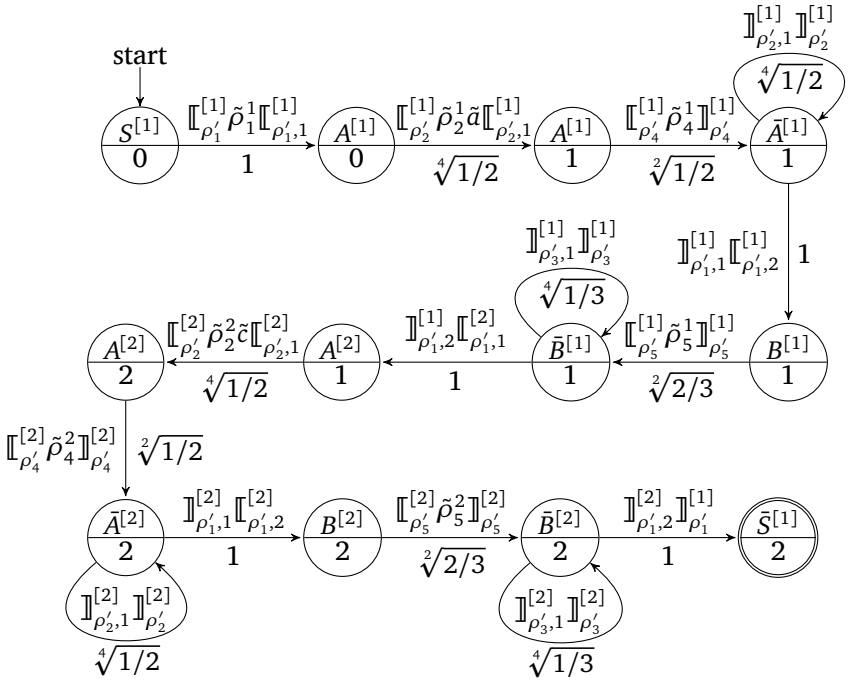

Figure 8: Product of $\mathscr{M}'(G_{\mathbb{B}})$ and $\mathscr{M}_{h,w}$ with the weight assigned by $\mathrm{wt}_{G,\trianglelefteq}$ to words read in each transition

Table 4: First eight paths (sorted by their image under $\mathrm{wt}_{G,\trianglelefteq}$) in the product of $\mathscr{M}'(G_{\mathbb{B}})$ and $\mathscr{M}_{h,w}$ and whether the corresponding candidate u_i is in $mD_{\mathrm{c}}(G_{\mathbb{B}})$

i	$\mathrm{wt}_{G,\trianglelefteq}(u_i)$	path corresponding to u_i	$u_i \in mD_{\mathrm{c}}(G_{\mathbb{B}})$?
1	$\frac{1}{3\sqrt{2}}$	without using any loops	no
2	$\frac{1}{3\sqrt[4]{8}}$	use the loop of $(\bar{A}^{[1]}, 1)$	no
3		use the loop of $(\bar{A}^{[2]}, 2)$	no
4	$\frac{1}{3\sqrt[4]{12}}$	use the loop of $(\bar{B}^{[1]}, 1)$	no
5		use the loop of $(\bar{B}^{[2]}, 2)$	no
6	$\frac{1}{6}$	use the loop of $(\bar{A}^{[1]}, 1)$ twice	no
7		use the loops of $(\bar{A}^{[1]}, 1)$ and $(\bar{A}^{[2]}, 2)$	yes
8		use the loop of $(\bar{A}^{[2]}, 2)$ twice	no

From the proof of termination of Algorithm 3 we can gather that the complexity of the algorithm depends on how many candidates are considered. The upper bound for the number of considered candidates is determined by n, θ, and the number $\widehat{k}$. In particular, it does not depend on the input word w. Therefore we cannot expect to get a meaningful time complexity for Algorithm 3.

Instead we will only determine the time complexity of the evaluation of *isMember* on line 7.

Lemma 5.24. Let G be an MCFG and $mD_{\mathfrak{c}}(\Sigma, \mathfrak{P}) = mD_{\mathfrak{c}}(G)$. The every element of $R'(G) \cap mD_{\mathfrak{c}}$ is $\mathfrak{P}$-simple.

Proof. Let $G = (N, \Gamma, S, P)$ and Σ and $\mathfrak{P}$ be defined as in Definition 4.9.

We prove the following statement by induction on the length of $w_1 \cdots w_\ell$: If $B \in N$ and $w_1, \ldots, w_\ell \in D(\Sigma)$ such that $w_1 \cdots w_\ell \in mD_{\mathfrak{c}}$ and w_κ is recognised along a path from $B^{[\kappa]}$ to $\overline{B}^{[\kappa]}$ in $\mathscr{M}'(G)$ for every $\kappa \in [\ell]$, then $w_1 \cdots w_\ell$ is $\mathfrak{P}$-simple.

By setting $\ell = 1$ and $B = S$, this statement implies our claim. Now let $B \in N$ and $w_1, \ldots, w_k \in D(\Sigma)$ such that $w_1 \cdots w_\ell \in mD_{\mathfrak{c}}$ and w_κ is recognised along a path from $B^{[\kappa]}$ to $\overline{B}^{[\kappa]}$ in $\mathscr{M}'(G)$ for every $\kappa \in [\ell]$. By the definitions of $\mathscr{M}'(G)$ and $mD_{\mathfrak{c}}(G)$, we know that there is some production $\rho = B \to f(B_1, \ldots, B_k) \in P$ with

$$f = [u_{1,0} x_{i(1,1)}^{j(1,1)} u_{1,1} \cdots x_{i(1,p_1)}^{j(1,p_1)} u_{1,p_1}, \ldots, u_{\ell,0} x_{i(\ell,1)}^{j(\ell,1)} u_{\ell,1} \cdots x_{i(\ell,p_\ell)}^{j(\ell,p_\ell)} u_{\ell,p_\ell}]$$

such that for every $\kappa \in [\ell]$ either

(i) $w_\kappa = [\![_\rho^{[\kappa]} \widetilde{u}_{\kappa,0}]\!]_\rho^{[\kappa]}$ or

(ii) $w_\kappa = [\![_\rho^{[\kappa]} \widetilde{u}_{\kappa,0} \; [\![_{\rho,i(\kappa,1)}^{[j(\kappa,1)]} \; v_{i(\kappa,1)}^{j(\kappa,1)} \;]\!]_{\rho,i(\kappa,1)}^{[j(\kappa,1)]} \; \widetilde{u}_{\kappa,1} \; \cdots$
$[\![_{\rho,i(\kappa,p_\kappa)}^{[j(\kappa,p_\kappa)]} \; v_{i(\kappa,p_\kappa)}^{j(\kappa,p_\kappa)} \;]\!]_{\rho,i(\kappa,p_\kappa)}^{[j(\kappa,p_\kappa)]} \; \widetilde{u}_{\kappa,p_\kappa} \;]\!]_\rho^{[\kappa]}$,

and $v_i^j \in D(\Sigma)$ is recognised along a path from $B_i^{[j]}$ to $\overline{B}_i^{[j]}$ in $\mathscr{M}'(G)$ for every $i \in [k]$ and $j \in [\mathrm{sort}(B_i)]$, and $v_i^1 \cdots v_i^{\mathrm{sort}(B_i)} \in mD_{\mathfrak{c}}(G)$ for every $i \in [k]$. Set $w'_1, \ldots, w'_\ell \in D(\Sigma)$ such that $w_\kappa = [\![_\rho^{[\kappa]} w'_\kappa]\!]_\rho^{[\kappa]}$ for each $\kappa \in [\ell]$. Then by induction hypothesis the string $v_i^1 \cdots v_i^{\mathrm{sort}(B_i)}$ is $\mathfrak{P}$-simple for every $i \in [k]$. By analysing the form of w_κ for each $\kappa \in [\ell]$, we observe that $\mathrm{occ}_{\mathfrak{p}}\, w'_1 \cdots w'_\ell \leq 1$ and $\mathrm{occ}_{\mathfrak{p}}\, w_1 \cdots w_\ell \leq 1$ for each cell $\mathfrak{p} \in \{\mathfrak{p}' \in \mathfrak{P} \mid |\mathfrak{p}'| \geq 2\}$. Hence, $w_1 \cdots w_\ell$ is $\mathfrak{P}$-simple. ∎

Let G be an MCFG and $mD_c(\Sigma, \mathfrak{P} = mD_c(G)$. Since every element of $R'(G) \cap mD_c(G)$ is $\mathfrak{P}$-simple, we can use ***isMember'*** on line 7 instead of ***isMember***. Then line 7 can be done in time quadratic in the length of the candidate u.

6 RELATED PARSING APPROACHES

An established approach to speed up the parsing of MCFGs for practical applications is to use a formalism with lower parsing complexity than MCFGs to guide the exploration of the search space. In the following, we will focus on four such approaches.

The parsers in Barthélemy *et al.* (2001); Burden and Ljunglöf (2005); van Cranenburgh (2012) work as follows: Suppose that we want to parse a given word w with a grammar G of a formalism A. We first construct a grammar (or automaton) G' in a formalism B that has a lower parsing complexity than A. This can be done offline. Then, we parse w with G'. Lastly, we parse w with G, but while doing so, we consult the parses of w in G' to *guide* the exploration of the search space (of possible parses). The three papers differ in their choice of formalisms for G and G', and in their use of the parses of w in G' while parsing w in G:

(i) Barthélemy *et al.* (2001) have a positive range concatenation grammar (short: PRCG) (Boullier 1998) of arbitrary arity for G and use a PRCG of arity 1 for G'. They extract from the parse forest F of w in G' a so-called *guiding structure* and query this structure while parsing w in G. The guiding structure can range from a set of instantiated clauses that occur in F to F itself. In their experiments they used as a guiding structure the function that assigns for each instantiated clause the number of its occurrences in F.

(ii) Burden and Ljunglöf (2005, Section 4) have a linear context-free rewriting system (short: LCFRS) (Vijay-Shanker *et al.* 1987) for G and a context-free grammar for G'. They use deductive parsing. The *parse chart* C' of w in G' is created. While creating the parse chart of w in G, only items are created that are consistent with the items in C'. The algorithm is therefore an instance of *coarse-to-fine parsing* (Charniak *et al.* 2006).

(iii) Van Cranenburgh (2012) has a probabilistic LCFRS (of arbitrary fan-out) for G and a probabilistic LCFRS of fan-out 1 for G'. As

Burden and Ljunglöf (2005), he uses deductive parsing: First, a parse chart C' of w in G' is created. Then the probabilities of G' are used to restrict C' to the n best parses, obtaining a new parse chart $\widehat{C}$, this step is called *pruning*. A value of $n = 50$ was used in the experiments. Then, while creating the parse chart of w in G, only items are created that are consistent with the items in $\widehat{C}$. The algorithm is an instance of coarse-to-fine parsing.

Kallmeyer and Maier (2015) present a different approach:

(iv) They construct an FSA G' as the predict/resume-closure of *thread automaton thread stores* (Villemonte de la Clergerie 2002) where the corresponding thread automaton is constructed from the given LCFRS G. The addresses in the thread stores are represented by regular expressions to keep the set of states of G' finite. Then, a *parse table* is read off of G'. As opposed to Items (i) to (iii), w is *not* parsed with G'. Instead, while parsing w with G using a shift-reduce parser, the parse table is consulted directly at each shift or reduce operation to determine the successor state. Their algorithm is an instance of LR-parsing.

With the Chomsky-Schützenberger parsing presented in this article, we construct from the given weighted LCFRS G *three* devices (instead of just one): the deterministic FSA $\mathscr{M}'(G_{\mathbb{B}})$ together with the weight assignment $\mathrm{wt}_{G,\trianglelefteq}$, the congruence multiple Dyck language $mD_{\mathsf{c}}(G_{\mathbb{B}})$, and the alphabetic homomorphism $\mathrm{hom}_{G_{\mathbb{B}}}$. For the given word w we construct a deterministic FSA, let us call it $\mathscr{M}$, that recognises $\mathrm{hom}^{-1}_{G_{\mathbb{B}}}(w) \cap L(\mathscr{M}'_{G_{\mathbb{B}}})$. Constructing $\mathscr{M}$ is an additional preprocessing step in comparison to Items (i) to (iv). In contrast to Item (iii), we do not use the weight assignment $\mathrm{wt}_{G,\trianglelefteq}$ for pruning. We instead use it to enumerate the elements of $L(\mathscr{M})$ in increasing order of their costs. Finally, we filter the list of those elements with $mD_{\mathsf{c}}(G_{\mathbb{B}})$. Note that w is never actually parsed with G as in Items (i) to (iv).

7 CONCLUSION AND OUTLOOK

We obtained a weighted version of the Chomsky-Schützenberger characterisation of MCFLs for complete commutative strong bimonoids (Theorem 3.12) by separating the weights from the weighted MCFG and using Yoshinaka *et al.* (2010, Theorem 3) for the unweighted part.

We defined a variant of multiple Dyck languages that uses congruence relations (Definition 4.2), gave an algorithm to decide whether a word is in a given congruence multiple Dyck language in Algorithm 2, and derived a CS representation using congruence multiple Dyck languages.

Following the idea of Hulden (2011), we used this CS representation for weighted MCFL to describe a parsing algorithm (Algorithm 3) that approximates n-best parsing for restricted weighted MCFGs with a weight structure that is partially ordered and factorisable.

The utility of Algorithm 3 can *not* be judged based on theoretical considerations alone. The author therefore plans to implement it and evaluate it empirically.

ACKNOWLEDGEMENTS

The author is very grateful to the anonymous reviewers, as well as Mark-Jan Nederhof and Toni Dietze for their insightful comments, which helped improve this article.

REFERENCES

Alfred Vaino AHO and Jeffrey D. ULLMAN (1972), *The Theory of Parsing, Translation, and Compiling*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, ISBN 0-13-914556-7, `http://dl.acm.org/citation.cfm?id=SERIES11430.578789`.

Krasimir ANGELOV (2009), Incremental parsing with parallel multiple context-free grammars, in *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 69–76, Association for Computational Linguistics, `http://dl.acm.org/citation.cfm?id=1609074`.

François BARTHÉLEMY, Pierre BOULLIER, Philippe DESCHAMP, and Éric DE LA CLERGERIE (2001), Guided parsing of range concatenation languages, in *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics - ACL '01*, Association for Computational Linguistics (ACL), doi:10.3115/1073012.1073019.

Pierre BOULLIER (1998), A generalization of mildly context-sensitive formalisms, in *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 4)*, pp. 17–20, Citeseer.

Sabine BRANTS, Stefanie DIPPER, Peter EISENBERG, Silvia HANSEN-SCHIRRA, Esther KÖNIG, Wolfgang LEZIUS, Christian ROHRER, George SMITH, and Hans

USZKOREIT (2004), TIGER: Linguistic Interpretation of a German Corpus, *Research on Language and Computation*, 2(4):597–620, doi:10.1007/s11168-004-7431-3.

Håkan BURDEN and Peter LJUNGLÖF (2005), Parsing Linear Context-free Rewriting Systems, in Harry BUNT, Robert MALOUF, and Alon LAVIE, editors, *Proceedings of the Ninth International Workshop on Parsing Technology*, Parsing '05, pp. 11–17, Association for Computational Linguistics, Stroudsburg, PA, USA, http://dl.acm.org/citation.cfm?id=1654494.1654496.

Matthias BÜCHSE, Daniel GEISLER, Torsten STÜBER, and Heiko VOGLER (2010), n-Best Parsing Revisited, in Frank DREWES and Marco KUHLMANN, editors, *Proceedings of the 2010 Workshop on Applications of Tree Automata in Natural Language Processing*, pp. 46–54, Association for Computational Linguistics, http://www.aclweb.org/anthology/W10-2506.

Eugene CHARNIAK, Michael POZAR, Theresa VU, Mark JOHNSON, Micha ELSNER, Joseph AUSTERWEIL, David ELLIS, Isaac HAXTON, Catherine HILL, R. SHRIVATHS, and Jeremy MOORE (2006), Multilevel coarse-to-fine PCFG parsing, in *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics* -, Association for Computational Linguistics (ACL), doi:10.3115/1220835.1220857.

Noam CHOMSKY and Marcel Paul SCHÜTZENBERGER (1963), The algebraic theory of context-free languages, pp. 118–161, doi:10.1016/S0049-237X(09)70104-1.

Tobias DENKINGER (2015), A Chomsky-Schützenberger representation for weighted multiple context-free languages, in *Proceedings of the 12th International Conference on Finite-State Methods and Natural Language Processing (FSMNLP 2015)*, http://aclweb.org/anthology/W15-4803.

Manfred DROSTE, Torsten STÜBER, and Heiko VOGLER (2010), Weighted finite automata over strong bimonoids, *Information Sciences*, 180(1):156–166, doi:10.1016/j.ins.2009.09.003.

Manfred DROSTE and Heiko VOGLER (2013), The Chomsky-Schützenberger Theorem for Quantitative Context-Free Languages, in Marie-Pierre BÉAL and Olivier CARTON, editors, *Developments in Language Theory*, volume 7907 of *Lecture Notes in Computer Science*, pp. 203–214, Springer Berlin Heidelberg, ISBN 978-3-642-38770-8, doi:10.1007/978-3-642-38771-5_19.

Jürgen DUSKE, Rainer PARCHMANN, and Johann SPECHT (1979), A Homomorphic Characterization of Indexed Languages, 15(4):187–195.

Séverine FRATANI and El Makki VOUNDY (2015), Context-free characterization of Indexed Languages, abs/1409.6112, http://arxiv.org/abs/1409.6112.

Séverine FRATANI and El Makki VOUNDY (2016), Homomorphic Characterizations of Indexed Languages, in Adrian-Horia DEDIU, Jan

JANOUŠEK, Carlos MARTÍN-VIDE, and Bianca TRUTHE, editors, *Proceedings of the 10th International Conference on Language and Automata Theory and Applications (LATA 2015)*, pp. 359–370, Springer Science + Business Media, doi:10.1007/978-3-319-30000-9_28.

Luisa HERRMANN and Heiko VOGLER (2015), A Chomsky-Schützenberger Theorem for Weighted Automata with Storage, in Andreas MALETTI, editor, *Proceedings of the 6th International Conference on Algebraic Informatics (CAI 2015)*, volume 9270, pp. 90–102, Springer International Publishing, ISBN 978-3-319-23021-4, doi:10.1007/978-3-319-23021-4_11.

Walter HOFFMAN and Richard PAVLEY (1959), A Method for the Solution of the Nth Best Path Problem, 6(4):506–514, doi:10.1145/320998.321004.

John Edward HOPCROFT and Jeffrey David ULLMAN (1969), *Formal languages and their relation to automata*, Addison-Wesley Longman Publishing Co., Inc.

John Edward HOPCROFT and Jeffrey David ULLMAN (1979), *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1st edition.

Liang HUANG and David CHIANG (2005), Better k-best Parsing, in *Proceedings of the Ninth International Workshop on Parsing Technology*, Parsing '05, pp. 53–64, Association for Computational Linguistics, Stroudsburg, PA, USA, `http://dl.acm.org/citation.cfm?id=1654494.1654500`.

Mans HULDEN (2011), Parsing CFGs and PCFGs with a Chomsky-Schützenberger Representation, in Zygmunt VETULANI, editor, *Human Language Technology. Challenges for Computer Science and Linguistics*, volume 6562 of *Lecture Notes in Computer Science*, pp. 151–160, Springer Berlin Heidelberg, ISBN 978-3-642-20094-6, doi:10.1007/978-3-642-20095-3_14.

Laura KALLMEYER and Wolfgang MAIER (2013), Data-driven parsing using probabilistic linear context-free rewriting systems, *Computational Linguistics*, 39(1):87–119, doi:10.1162/COLI_a_00136.

Laura KALLMEYER and Wolfgang MAIER (2015), LR Parsing for LCFRS, in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics (ACL), doi:10.3115/v1/n15-1134.

Makoto KANAZAWA (2014), Multidimensional trees and a Chomsky-Schützenberger-Weir representation theorem for simple context-free tree grammars, ISSN 1465-363X, doi:10.1093/logcom/exu043.

Marco KUHLMANN and Giorgio SATTA (2009), Treebank Grammar Techniques for Non-projective Dependency Parsing, in *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '09, pp. 478–486, Association for Computational Linguistics, Stroudsburg, PA, USA, `http://dl.acm.org/citation.cfm?id=1609067.1609120`.

Wolfgang MAIER and Anders SØGAARD (2008), Treebanks and mild context-sensitivity, in *Proceedings of Formal Grammar*, p. 61, http://web.stanford.edu/group/cslipublications/cslipublications/FG/2008/maier.pdf.

Jens MICHAELIS (2001a), Derivational Minimalism Is Mildly Context-Sensitive, in Michael MOORTGAT, editor, *Logical Aspects of Computational Linguistics*, volume 2014 of *Lecture Notes in Computer Science*, pp. 179–198, Springer Berlin Heidelberg, ISBN 978-3-540-42251-8, doi:10.1007/3-540-45738-0_11.

Jens MICHAELIS (2001b), Transforming Linear Context-Free Rewriting Systems into Minimalist Grammars, in Philippe GROOTE, Glyn MORRILL, and Christian RETORÉ, editors, *Logical Aspects of Computational Linguistics*, volume 2099 of *Lecture Notes in Computer Science*, pp. 228–244, Springer Berlin Heidelberg, ISBN 978-3-540-42273-0, doi:10.1007/3-540-48199-0_14.

Mehryar MOHRI (2000), Minimization algorithms for sequential transducers, 234(1–2):177–201, ISSN 0304-3975, doi:10.1016/s0304-3975(98)00115-7.

Arto SALOMAA (1973), *Formal languages*, Academic Press, ISBN 978-0126157505.

Arto SALOMAA and Matti SOITTOLA (1978), *Automata-Theoretic Aspects of Formal Power Series*, Springer New York, ISBN 978-1-4612-6266-4, doi:10.1007/978-1-4612-6264-0.

Hiroyuki SEKI, Takashi MATSUMURA, Mamoru FUJII, and Tadao KASAMI (1991), On multiple context-free grammars, 88(2):191–229, ISSN 0304-3975, doi:10.1016/0304-3975(91)90374-B.

Hiroyuki SEKI, Ryuichi NAKANISHI, Yuichi KAJI, Sachiko ANDO, and Tadao KASAMI (1993), Parallel Multiple Context-free Grammars, Finite-state Translation Systems, and Polynomial-time Recognizable Subclasses of Lexical-functional Grammars, in *Proceedings of the 31st Annual Meeting on Association for Computational Linguistics*, ACL '93, pp. 130–139, Association for Computational Linguistics, Stroudsburg, PA, USA, doi:10.3115/981574.981592.

Andreas VAN CRANENBURGH (2012), Efficient Parsing with Linear Context-free Rewriting Systems, in Walter DAELEMANS, editor, *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '12, pp. 460–470, Association for Computational Linguistics, Stroudsburg, PA, USA, ISBN 978-1-937284-19-0, http://dl.acm.org/citation.cfm?id=2380816.2380873.

Krishnamurti VIJAY-SHANKER (1988), *A study of tree adjoining grammars*, Ph.D. thesis, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.401.1695&rep=rep1&type=pdf.

Krishnamurti VIJAY-SHANKER, David Jeremy WEIR, and Aravind K. JOSHI (1986), Tree adjoining and head wrapping, in *Proceedings of the 11th Conference*

on Computational Linguistics, pp. 202–207, Association for Computational Linguistics, doi:10.3115/991365.991425.

Krishnamurti VIJAY-SHANKER, David Jeremy WEIR, and Aravind K. JOSHI (1987), Characterizing Structural Descriptions Produced by Various Grammatical Formalisms, in *Proceedings of the 25th Annual Meeting on Association for Computational Linguistics*, ACL '87, pp. 104–111, Association for Computational Linguistics, Stroudsburg, PA, USA, doi:10.3115/981175.981190.

Éric VILLEMONTE DE LA CLERGERIE (2002), Parsing Mildly Context-Sensitive Languages with Thread Automata, in *Proceedings of the 19th International Conference on Computational Linguistics (COLING '02)*, volume 1, pp. 1–7, Association for Computational Linguistics, Stroudsburg, PA, USA, doi:10.3115/1072228.1072256.

David Jeremy WEIR (1988), *Characterizing mildly context-sensitive grammar formalisms*, Ph.D. thesis, `http://repository.upenn.edu/dissertations/AAI8908403`.

David Jeremy WEIR and Aravind K. JOSHI (1988), Combinatory categorial grammars: Generative power and relationship to linear context-free rewriting systems, in *Proceedings of the 26th annual meeting on Association for Computational Linguistics*, pp. 278–285, Association for Computational Linguistics, doi:10.3115/982023.982057.

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*, pp. 596–607, Springer, ISBN 978-3-642-13088-5, doi:10.1007/978-3-642-13089-2_50.