A Chomsky-Schützenberger Theorem for Weighted Automata with Storage

Luisa Herrmann and Heiko Vogler(✉)

Department of Computer Science,
Technische Universität Dresden, D-01062 Dresden, Germany
{Luisa.Herrmann,Heiko.Vogler}@tu-dresden.de

Abstract. We enrich the concept of automata with storage by weights taken from any unital valuation monoid. We prove a Chomsky-Schützenberger theorem for the class of weighted languages recognizable by such weighted automata with storage.

1 Introduction

The classical Chomsky-Schützenberger theorem [3, Prop. 2] (for short: CS theorem) states that each context-free language is the homomorphic image of the intersection of a Dyck-language and a regular language. In [28] it was shown under which conditions the homomorphism can be non-erasing. In [23] the CS theorem was employed to specify a parser for context-free languages. The CS theorem has been extended to string languages generated by tree-adjoining grammars [32], multiple context-free languages [33], indexed languages [17][1], and yield images of simple context-free tree languages [25].

Already in [3] the CS theorem for context-free languages was proved in a special weight setting: each word in the language is associated with the number of its derivations. In [29] the CS theorem was shown for algebraic (formal) power series over commutative semirings. In [9] this result was generalized to algebraic power series over unital valuation monoids, called quantitative context-free languages; (unital) valuation monoids allow to describe, e.g., average consumption of energy. Also in [9] quantitative context-free languages were characterized by weighted pushdown automata over unital valuation monoids. Recently, the CS theorem has been proved for weighted multiple context-free languages over complete commutative strong bimonoids [6].

In the classical CS theorem, the set Y of letters occurring in the Dyck-language depends on the given context-free grammar or pushdown automaton. An alternative is to code Y by a homomorphism g over a two-letter alphabet and to obtain the following CS theorem [22, Thm. 10.4.3]: each context-free language L can be represented in the form $L = h(g^{-1}(D_2) \cap R)$ for some homomorphisms h and g and a regular language R; D_2 denotes the Dyck-language over a two letter alphabet. In the sequel we call this alternative the CS theorem.

[1] We are grateful to one of the reviewers for pointing out this reference to us.

A. Maletti (Ed.): CAI 2015, LNCS 9270, pp. 115–127, 2015.
DOI: 10.1007/978-3-319-23021-4_11

In this paper we prove a CS theorem for the class of weighted languages recognizable by weighted iterated pushdown automata over unital valuation monoids. A weighted language[2] is a mapping from Σ^* to some weight algebra. Intuitively, an iterated pushdown is a pushdown in which each square contains a pushdown in which each square contains a pushdown ... (and so on). The idea of iterated pushdowns goes back to [21,26,27]. It was proved in [11, Thm. 6] that the classes of languages accepted by iterated pushdown automata form a strict, infinite hierarchy with increasing nesting of pushdowns. In [5] it was proved that n-iterated pushdown automata characterize the n-th level of the OI-string language hierarchy [4,13,31] which starts at its first three levels with the regular, context-free, and indexed languages [1] (equivalently, OI-macro languages [16]).

We obtain the CS theorem for weighted iterated pushdown automata as application of the even more general, main result of our paper: the CS theorem for K-weighted automata with storage where K is an arbitrary unital valuation monoid. An automaton with storage S [30,19,12][3] is a one-way nondeterministic finite-state automaton with an additional storage of type S; a successful computation starts with the initial state and an initial configuration of S; in each transition the automaton can test the current storage configuration and apply an instruction to it. For instance, pushdown automata, n-iterated pushdown automata, stack automata [20], and nested stack automata [2] can be formulated as automata with storage. For a number of examples of storages we refer to [12] where these automata were called $\mathrm{REG}(S)$ r-acceptors. The concept of automata with storage is quite flexible: for instance, we can also express M-automata [24] where M is a (multiplicative) monoid, in a straightforward way as such automata with storage (cf. Ex. 4).

We extend the concept of automata with storage to that of K-weighted automata with storage where K is a unital valuation monoid; this extension is done in the same way as pushdown automata have been extended in [9] to weighted pushdown automata over unital valuation monoids. Then our main result states the following (cf. Thm. 11). Let $r\colon \Sigma^* \to K$ be recognizable by some K-weighted automaton over storage type S. Then there are a regular language R, a finite set Ω of pairs (each consisting of a predicate and an instruction), a configuration c of S, a letter-to-letter morphism g, and a (weighted) alphabetic morphism h such that $r = h(g^{-1}(\mathrm{B}(\Omega, c)) \cap R)$ where $\mathrm{B}(\Omega, c)$ is the set of all Ω-behaviours of c.

2 Preliminaries

Notations and Notions. The set of non-negative integers (including 0) is denoted by $\mathbb{N}$. Let $n \in \mathbb{N}$. Then $[n]$ denotes the set $\{i \in \mathbb{N} \mid 1 \leq i \leq n\}$. Thus $[0] = \emptyset$. Let A and B be sets. The set of all subsets (finite subsets) of A is denoted by $\mathcal{P}(A)$ ($\mathcal{P}_{\mathrm{fin}}(A)$, resp.). We denote the identity mapping on A by id_A. Let $f\colon A \to B$ be a mapping. We denote by $\mathrm{im}(f)$ the set $\{b \in B \mid \exists a \in A : f(a) = b\}$.

[2] or, equivalently, formal power series

[3] If we cite notions or definitions from [12], then we always refer to the version of 2014.

We fix a countably infinite set Λ and call its elements symbols. We call each finite subset Σ of Λ an *alphabet. In the rest of this paper, we let* Σ *and* Δ *denote alphabets unless specified otherwise.*

Unital Valuation Monoids. The concept of valuation monoid was introduced in [7,8] and extended in [9] to unital valuation monoid. A *unital valuation monoid* is a tuple $(K, +, \mathrm{val}, 0, 1)$ such that $(K, +, 0)$ is a commutative monoid and $\mathrm{val}\colon K^* \to K$ is a mapping such that (i) $\mathrm{val}(a) = a$ for each $a \in K$, (ii) $\mathrm{val}(a_1, \ldots, a_n) = 0$ whenever $a_i = 0$ for some $i \in [n]$, (iii) $\mathrm{val}(a_1, \ldots, a_{i-1}, 1, a_{i+1}, \ldots, a_n) = \mathrm{val}(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$ for any $i \in [n]$, and (iv) $\mathrm{val}(\varepsilon) = 1$.

A monoid $(K, +, 0)$ is *complete* if it has an infinitary sum operation $\sum_I\colon K^I \to K$ for each enumerable set I (for the axioms cf. [10]). We call a unital valuation monoid $(K, +, \mathrm{val}, 0, 1)$ complete if $(K, +, 0)$ has this property. We write $\sum_{i\in I} a_i$ instead of $\sum_I (a_i \mid i \in I)$.

We refer the reader to [9, Ex. 1 and 2] for a number of examples of unital valuation monoids. For instance, each complete semiring (in particular, the *Boolean semiring* $\mathbb{B} = (\{0, 1\}, \vee, \wedge, 0, 1))$ and each complete lattice is a complete unital valuation monoid. *In the rest of this paper, we let K denote an arbitrary unital valuation monoid* $(K, +, \mathrm{val}, 0, 1)$ *unless specified otherwise.*

Weighted Languages. A *K-weighted language over Σ* is a mapping of the form $r\colon \Sigma^* \to K$. We denote the set of all such mappings by $K\langle\langle\Sigma^*\rangle\rangle$. For every $r \in K\langle\langle\Sigma^*\rangle\rangle$, we denote the set $\{w \in \Sigma^* \mid r(w) \neq 0\}$ by $\mathrm{supp}(r)$.

A family $(r_i \mid i \in I)$ of K-weighted languages $r_i \in K\langle\langle\Sigma^*\rangle\rangle$ is *locally finite* if for each $w \in \Sigma^*$ the set $I_w = \{i \in I \mid r_i(w) \neq 0\}$ is finite. In this case or if K is complete, we define $\sum_{i\in I} s_i \in K\langle\langle\Sigma^*\rangle\rangle$ by $\big(\sum_{i\in I} s_i\big)(w) = \sum_{i\in I_w} s_i(w)$ for each $w \in \Sigma^*$.

Each $L \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle$ determines the set $\mathrm{supp}(L) \subseteq \Sigma^*$. Vice versa, each set $L \subseteq \Sigma^*$ determines the $\mathbb{B}$-weighted language $\chi_L \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle$ with $\chi_L(w) = 1$ if and only if $w \in L$. Thus, for every $L \subseteq \Sigma^*$, we have $\mathrm{supp}(\chi_L) = L$; and for every $L \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle$ we have $\chi_{\mathrm{supp}(L)} = L$. In the sequel we will not distinguish between these two points of view.

3 Weighted Automata with Storage

We take up the concept of automata with storage [30] and present it in the style of [12] (cf. [14,15] for further investigations). Moreover, we add weights to the transitions of the automaton where the weights are taken from some unital valuation monoid.

Storage Types: We recall the definition of storage type from [12,30] with a slight modification. A *storage type* S is a tuple (C, P, F, C_0) where C is a set (*configurations*), P is a set of total functions each having the type $p\colon C \to \{\mathrm{true}, \mathrm{false}\}$ (*predicates*), F is a set of partial functions each having the type $f\colon C \to C$ (*instructions*), and $C_0 \subseteq C$ (*initial configurations*).

Example 1. Let c be an arbitrary but fixed symbol. The *trivial storage type* is the storage type $\mathrm{TRIV} = (\{\mathrm{c}\}, \{\mathrm{p_{true}}\}, \{\mathrm{f_{id}}\}, \{\mathrm{c}\})$ where $\mathrm{p_{true}}(\mathrm{c}) = \mathrm{true}$ and $\mathrm{f_{id}}(\mathrm{c}) = \mathrm{c}$. □

Next we recall the pushdown operator P from [12, Def. 5.1] and [14, Def. 3.28]: if S is a storage type, then $\mathrm{P}(S)$ is a storage type of which the configurations have the form of a pushdown; each cell contains a pushdown symbol and a configuration of S. Formally, let Γ be a fixed infinite set (*pushdown symbols*). Also, let $S = (C, P, F, C_0)$ be a storage type. The *pushdown of* S is the storage type $\mathrm{P}(S) = (C', P', F', C'_0)$ where

- $C' = (\Gamma \times C)^+$ and $C'_0 = \{(\gamma_0, c_0) \mid \gamma_0 \in \Gamma, c_0 \in C_0\}$,
- $P' = \{\mathrm{bottom}\} \cup \{(\mathrm{top} = \gamma) \mid \gamma \in \Gamma\} \cup \{\mathrm{test}(p) \mid p \in P\}$ such that for every $(\delta, c) \in \Gamma \times C$ and $\alpha \in (\Gamma \times C)^*$ we have
$$\begin{aligned} &\mathrm{bottom}\big((\delta,c)\alpha\big) &&= \mathrm{true} \text{ if and only if } \alpha = \varepsilon\\ &(\mathrm{top}=\gamma)\big((\delta,c)\alpha\big) &&= \mathrm{true} \text{ if and only if } \gamma = \delta\\ &\mathrm{test}(p)\big((\delta,c)\alpha\big) &&= p(c)\end{aligned}$$
- $F' = \{\mathrm{pop}\} \cup \{\mathrm{stay}(\gamma) \mid \gamma \in \Gamma\} \cup \{\mathrm{push}(\gamma, f) \mid \gamma \in \Gamma, f \in F\}$ such that for every $(\delta, c) \in \Gamma \times C$ and $\alpha \in (\Gamma \times C)^*$ we have
$$\begin{aligned} &\mathrm{pop}\big((\delta,c)\alpha\big) &&= \alpha \text{ if } \alpha \neq \varepsilon\\ &\mathrm{stay}(\gamma)\big((\delta,c)\alpha\big) &&= (\gamma,c)\alpha\\ &\mathrm{push}(\gamma,f)\big((\delta,c)\alpha\big) &&= (\gamma,f(c))(\delta,c)\alpha \text{ if } f(c) \text{ is defined}\end{aligned}$$
  and undefined in all other situations.

For each $n \geq 0$ we define $\mathrm{P}^n(S)$ inductively as follows: $\mathrm{P}^0(S) = S$ and $\mathrm{P}^n(S) = \mathrm{P}(\mathrm{P}^{n-1}(S))$ for each $n \geq 1$.

Example 2. Intuitively, $\mathrm{P}(\mathrm{TRIV})$ corresponds to the usual pushdown storage except that there is no empty pushdown. For $n \geq 0$, we abbreviate $\mathrm{P}^n(\mathrm{TRIV})$ by P^n and call it the *n-iterated pushdown storage.* □

Throughout this paper we let S denote an arbitrary storage type (C, P, F, C_0) unless specified otherwise.

Automata with Storage: An (S, Σ)-*automaton* is a tuple $\mathcal{A} = (Q, \Sigma, c_0, q_0, Q_f, T)$ where Q is a finite set (*states*), Σ is an alphabet (*terminal symbols*), $c_0 \in C_0$ (*initial configuration*), $q_0 \in Q$ (*initial state*), $Q_f \subseteq Q$ (*final states*), and $T \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times P \times Q \times F$ is a finite set (*transitions*). If $T \subseteq Q \times \Sigma \times P \times Q \times F$, then we call $\mathcal{A}$ *ε-free.*

The computation relation of $\mathcal{A}$ is the binary relation on the set $Q \times \Sigma^* \times C$ of $\mathcal{A}$-*configurations* defined as follows. For every transition $\tau = (q, x, p, q', f)$ in T we define the binary relation $\vdash^\tau$ on the set of $\mathcal{A}$-configurations: for every $w \in \Sigma^*$ and $c \in C$, we let $(q, xw, c) \vdash^\tau (q', w, f(c))$ if $p(c)$ is true and $f(c)$ is defined. The *computation relation of* $\mathcal{A}$ is the binary relation $\vdash = \bigcup_{\tau \in T} \vdash^\tau$. The *language recognized by* $\mathcal{A}$ is the set $L(\mathcal{A}) = \{w \in \Sigma^* \mid (q_0, w, c_0) \vdash^* (q_f, \varepsilon, c) \text{ for some } q_f \in Q_f, c \in C\}$.

A *computation* is a sequence $\theta = \tau_1 \dots \tau_n$ of transitions τ_i ($i \in [n]$) such that there are $\mathcal{A}$-configurations $c_0, \dots, c_n$ with $c_{i-1} \vdash^{\tau_i} c_i$. We abbreviate this computation by $c_0 \vdash^\theta c_n$. Let $q \in Q$, $w \in \Sigma^*$, and $c \in C$. A *q-computation on w and c* is a computation θ such that $(q, w, c) \vdash^\theta (q_f, \varepsilon, c')$ for some $q_f \in Q_f$, $c' \in C$. We denote the set of all q-computations on w and c by $\Theta_{\mathcal{A}}(q, w, c)$. Furthermore, we denote the set of all q_0-computations on w and c_0 by $\Theta_{\mathcal{A}}(w)$. Thus we have $L(\mathcal{A}) = \{w \in \Sigma^* \mid \Theta_{\mathcal{A}}(w) \neq \emptyset\}$.

We say that $\mathcal{A}$ is *ambiguous* if there is a $w \in \Sigma^*$ such that $|\Theta_{\mathcal{A}}(w)| \geq 2$. Otherwise $\mathcal{A}$ is *unambiguous*. A language $L \subseteq \Sigma^*$ is *(S, Σ)-recognizable* if there is an (S, Σ)-automaton $\mathcal{A}$ with $L(\mathcal{A}) = L$.

Example 3. (1) The TRIV-automata are (usual) finite-state automata, and P^1-automata are essentially pushdown automata. (2) For each $n \geq 1$, P^n-automata correspond to n-iterated pushdown automata of [26,27,11,5]. (3) Nested stack automata [2] correspond to NS(TRIV)-automata where NS is an operator on storage types (cf. [14, Def. 7.1]). In [14, Thm. 7.4] it was proved that, for every S, the storage types $\mathrm{P}^2(S)$ and $\mathrm{NS}(S)$ are equivalent (cf. [14, Def. 4.6] for the definition of equivalence), which implies that the acceptance power of automata using these storage types is the same (cf. [14, Thm. 4.18] for this implication). □

Example 4. We indicate how to embed the concept of M-automata [24] where $(M, \cdot, 1)$ is a multiplicative monoid, into the setting of automata with storage. For this we define the storage type *monoid M*, denoted by $\mathrm{MON}(M)$, by (C, P, F, C_0) where $C = M$ and $C_0 = \{1\}$, $P = \{\text{true?}\} \cup \{1?\}$ with $\text{true?}(m) = \text{true}$, and $1?(m) = \text{true}$ if and only if $m = 1$, $F = \{[m] \mid m \in M\}$ and $[m]\colon M \to M$ is defined by $[m](m') = m' \cdot m$.

For a given M-automaton $\mathcal{A}$, we construct an equivalent $\mathrm{MON}(M)$-automaton $\mathcal{B}$ as follows. If (q, x, q', m) is a transition of $\mathcal{A}$ (with states q, q', input symbol x, and $m \in M$), then $(q, x, \text{true?}, q', [m])$ is a transition of $\mathcal{B}$. Moreover, for each final state q of $\mathcal{A}$, the transition $(q, \varepsilon, 1?, q_f, [1])$ is in $\mathcal{B}$ where q_f is the only final state of $\mathcal{B}$. □

Weighted Automata with Storage: Next we define the weighted version of (S, Σ)-automata. The line of our definitions follows the definition of weighted pushdown automata in [9].

An *(S, Σ)-automaton with weights in K* is a tuple $\mathcal{A} = (Q, \Sigma, c_0, q_0, Q_f, T, \mathrm{wt})$ where $(Q, \Sigma, c_0, q_0, Q_f, T)$ is an (S, Σ)-automaton (*underlying (S, Σ)-automaton*) and $\mathrm{wt}\colon T \to K$ (*weight assignment*). If the underlying (S, Σ)-automaton is ε-free, then we call $\mathcal{A}$ *ε-free*. Let $\theta = \tau_1 \dots \tau_n$ be a computation of $\mathcal{A}$. The *weight of θ* is the element in K defined by $\mathrm{wt}(\theta) = \mathrm{val}(\mathrm{wt}(\tau_1), \dots, \mathrm{wt}(\tau_n))$.

An *(S,Σ,K)-automaton* is an (S, Σ)-automaton $\mathcal{A}$ with weights in K such that (i) $\Theta_{\mathcal{A}}(w)$ is finite for every $w \in \Sigma^*$ or (ii) K is complete. In this case the *weighted language recognized by* $\mathcal{A}$ is the K-weighted language $\|\mathcal{A}\|\colon \Sigma^* \to K$ defined for every $w \in \Sigma^*$ by $\|\mathcal{A}\|(w) = \sum_{\theta \in \Theta_{\mathcal{A}}(w)} \mathrm{wt}(\theta)$.

A weighted language $r\colon \Sigma^* \to K$ is *(S, Σ, K)-recognizable* if there is an (S, Σ, K)-automaton $\mathcal{A}$ such that $r = \|\mathcal{A}\|$.

Example 5. (1) Each $(S, \Sigma, \mathbb{B})$-automaton $\mathcal{A}$ can be considered as an (S, Σ)-automaton which recognizes $\mathrm{supp}(\|\mathcal{A}\|)$. (2) Apart from ε-moves, $(\mathrm{TRIV}, \Sigma, K)$-automata are the same as weighted finite automata over Σ and the valuation monoid K [9]. (3) The $(\mathrm{P}^1, \Sigma, K)$-automata are essentially the same as weighted pushdown automata over Σ and K [9] where acceptance with empty pushdown can be simulated in the usual way. Thus, for every $r\colon \Sigma^* \to K$ we have: r is the quantitative behaviour of a WPDA as defined in [9] if and only if r is $(\mathrm{P}^1, \Sigma, K)$-recognizable. $\square$

For $n \geq 0$, a *weighted n-iterated pushdown language over Σ and K* is a $(\mathrm{P}^n, \Sigma, K)$-recognizable weighted language.

4 Separating the Weights from an (S, Σ, K)-Automaton

In this section we will represent an (S, Σ, K)-recognizable weighted language as the homomorphic image of an (S, Δ)-recognizable language.

We recall from [9] the concept of (weighted) alphabetic morphism. First, we introduce monomes and alphabetic morphisms. A mapping $r\colon \Sigma^* \to K$ is called a *monome* if $\mathrm{supp}(r)$ is empty or a singleton. If $\mathrm{supp}(r) = \{w\}$, then we also write $r(w).w$ instead of r. We let $K[\Sigma \cup \{\varepsilon\}]$ denote the set of all monomes with support in $\Sigma \cup \{\varepsilon\}$.

Let Δ be an alphabet and $h\colon \Delta \to K[\Sigma \cup \{\varepsilon\}]$ be a mapping. The *alphabetic morphism induced by h* is the mapping $h'\colon \Delta^* \to K\langle\langle \Sigma^* \rangle\rangle$ such that for every $n \geq 0$, $\delta_1, \dots, \delta_n \in \Delta$ with $h(\delta_i) = a_i.y_i$ we have $h'(\delta_1 \dots \delta_n) = \mathrm{val}(a_1, \dots, a_n).y_1 \dots y_n$. Note that $h'(v)$ is a monome for every $v \in \Delta^*$, and $h'(\varepsilon) = 1.\varepsilon$. If $L \subseteq \Delta^*$ such that the family $(h'(v) \mid v \in L)$ is locally finite or if K is complete, we let $h'(L) = \sum_{v \in L} h'(v)$. In the sequel we will use the following convention. If we write "alphabetic morphism $h\colon \Delta \to K[\Sigma \cup \{\varepsilon\}]$", then we mean the alphabetic morphism induced by h.

We define a special case of alphabetic morphisms in which $K = \mathbb{B}$. If for every $\delta \in \Delta$ the support of $h(\delta)$ is $\{\sigma\}$ for some $\sigma \in \Sigma$, then we call h' a *letter-to-letter morphism*. Note that in this case the alphabetic morphism induced by h has the property that for every $v \in \Delta^*$, $\mathrm{supp}(h'(v))$ contains at most one element and if $\mathrm{supp}(h'(v)) = \{w\}$ for some $w \in \Sigma^*$, then the lengths of w and v are equal.

Theorem 6. *For every $r \in K\langle\langle \Sigma^* \rangle\rangle$ the following two statements are equivalent:*
(1) r is (S, Σ, K)-recognizable.
(2) There are an alphabet Δ, an unambiguous ε-free (S, Δ)-automaton $\mathcal{A}$, and an alphabetic morphism $h\colon \Delta \to K[\Sigma \cup \{\varepsilon\}]$ such that $r = h(L(\mathcal{A}))$.

Proof. (1) $\Rightarrow$ (2): This generalizes [9, Lm. 3] in a straightforward way. Let $\mathcal{B} = (Q, \Sigma, c_0, q_0, Q_f, T, \mathrm{wt})$ be an (S, Σ, K)-automaton. We construct the (S, T)-automaton $\mathcal{A} = (Q, T, c_0, q_0, Q_f, T')$ and the mapping $h\colon T \to K[\Sigma \cup \{\varepsilon\}]$ such that, if $\tau = (q, x, p, q', f)$ is in T, then (q, τ, p, q', f) is in T' and we define $h(\tau) = \mathrm{wt}(\tau).x$. Obviously, $\mathcal{A}$ is unambiguous and ε-free.

Let $w \in \Sigma^*$ and $\theta = \tau_1 \dots \tau_n \in \Theta_{\mathcal{B}}(w)$. By definition of h, we have that $h(\theta) = \mathrm{val}(\mathrm{wt}(\tau_1), \dots, \mathrm{wt}(\tau_n)).w$. Hence $\mathrm{wt}(\theta) = \big(h(\theta)\big)(w)$. Also, by definition of (S, Σ, K)-automata, the set $\Theta_{\mathcal{B}}(w)$ is finite if K is not complete. Thus the family $(h(\theta) \mid \theta \in L(\mathcal{A}))$ is locally finite if K is not complete. Then, for every $w \in \Sigma^*$, we have $\|\mathcal{B}\|(w) = \sum_{\theta \in \Theta_{\mathcal{B}}(w)} \mathrm{wt}(\theta) = \sum_{\theta \in \Theta_{\mathcal{B}}(w)} \big(h(\theta)\big)(w) \overset{(*)}{=} \sum_{\theta \in L(\mathcal{A})} \big(h(\theta)\big)(w) = \big(\sum_{\theta \in L(\mathcal{A})} h(\theta)\big)(w) = \big(h(L(\mathcal{A}))\big)(w)$ where $(*)$ holds because for every $\theta \in L(\mathcal{A})$ with $\theta \notin \Theta_{\mathcal{B}}(w)$, we have $\big(h(\theta)\big)(w) = 0$ and due to the fact that $\sum_{\theta \in L(\mathcal{A}),\, \theta \notin \Theta_{\mathcal{B}}(w)} 0 = 0$. Thus $\|\mathcal{B}\| = h(L(\mathcal{A}))$.

(2) $\Rightarrow$ (1): Let $\mathcal{A} = (Q, \Delta, c_0, q_0, Q_f, T)$ be an unambiguous ε-free (S, Δ)-automaton and $h \colon \Delta \to K[\Sigma \cup \{\varepsilon\}]$ an alphabetic morphism. Moreover, we assume that the family $(h(v) \mid v \in L(\mathcal{A}))$ is locally finite if K is not complete. We will construct an (S, Σ, K)-automaton $\mathcal{B}$ such that $\|\mathcal{B}\| = h(L(\mathcal{A}))$.

Our construction employs a similar technique of coding the preimage of h into the set of states as in [9, Lm. 4] in order to handle non-injectivity of h appropriately. However, we have to modify the construction slightly, because the straightforward generalization would require that S has an identity instruction (needed in the first step of the computation), which in general we do not assume. In our constructed automaton, the target state (and not, as in [9, Lm. 4], the source state) of each transition encodes a preimage of the symbol which is read by this transition.

Formally, we construct the (S, Σ, K)-automaton $\mathcal{B} = (Q', \Sigma, c_0, q_0', Q_f', T', \mathrm{wt})$ where $Q' = \{q_0'\} \cup \Delta \times Q$ with some element q_0' with $q_0' \notin \Delta \times Q$, $Q_f' = \Delta \times Q_f$, and T' and wt are defined as follows. Let $\delta \in \Delta$ and $h(\delta) = a.y$.

- If (q_0, δ, p, q, f) is in T, then $(q_0', y, p, (\delta, q), f)$ is in T', and its weight is a.
- If (q, δ, p, q', f) is in T, then $((\delta', q), y, p, (\delta, q'), f)$ is in T' for each $\delta' \in \Delta$, and its weight is a.

Let $w \in \Sigma^*$. First, let $v \in \Delta^*$ with $h(v) = z.w$ for some $z \in K$. We write $v = \delta_1 \dots \delta_n \in \Delta^*$ with $n \geq 0$ and $\delta_i \in \Delta$. Let $h(\delta_i) = a_i.y_i$ for every $1 \leq i \leq n$. Thus $h(v) = \mathrm{val}(a_1, \dots, a_n).y_1 \dots y_n$ and $w = y_1 \dots y_n$ and $z = \mathrm{val}(a_1, \dots, a_n)$.

Let $\theta = \tau_1 \dots \tau_n$ be a q_0-computation in $\Theta_{\mathcal{A}}(v)$. Clearly, for each $i \in [n]$, the second component of τ_i is δ_i. Then we construct the q_0'-computation $\theta' = \tau_1' \dots \tau_n'$ in $\Theta_{\mathcal{B}}(y_1 \dots y_n)$ inductively as follows:

- If $\tau_1 = (q_0, \delta_1, p_1, q_1, f_1)$, then we let $\tau_1' = (q_0', y_1, p_1, (\delta_1, q_1), f_1)$.
- If $1 < i \leq n$ and $\tau_i = (q_{i-1}, \delta_i, p_i, q_i, f_i)$, then we let $\tau_i' = ((\delta_{i-1}, q_{i-1}), y_i, p_i, (\delta_i, q_i), f_i)$.

Note that $\big(h(v)\big)(w) = \mathrm{val}(a_1, \dots, a_n) = \mathrm{val}(\mathrm{wt}(\tau_1'), \dots, \mathrm{wt}(\tau_n')) = \mathrm{wt}(\theta')$.

Conversely, for every q_0'-computation $\theta' = \tau_1' \dots \tau_n'$ in $\Theta_{\mathcal{B}}(w)$ by definition of T' there are a uniquely determined $v \in \Delta^*$ and a uniquely determined q_0-computation $\theta = \tau_1 \dots \tau_n$ in $\Theta_{\mathcal{A}}(v)$ such that θ' is the computation constructed above. Hence, for every $v \in \Delta^*$ and $w \in \Sigma^*$, if $h(v) = z.w$ for some $z \in K$, then $\Theta_{\mathcal{A}}(v)$ and $\Theta_{\mathcal{B}}(w)$ are in a one-to-one correspondence.

Thus, for every $w \in \Sigma^*$, we obtain $\big(h(L(\mathcal{A}))\big)(w) = \sum_{v \in L(\mathcal{A})} \big(h(v)\big)(w) = \sum_{\substack{v \in L(\mathcal{A}): \\ (h(v))(w) \neq 0}} \big(h(v)\big)(w)$. Since $\mathcal{A}$ is unambiguous this is equal to

$\sum_{\substack{v\in L(\mathcal{A}),\theta\in\Theta_{\mathcal{A}}(v):\\ (h(v))(w)\neq 0}} \mathrm{wt}(\theta')$. Since there is a one-to-one correspondence between $\Theta_{\mathcal{A}}(v)$ and $\Theta_{\mathcal{B}}(w)$, this is equal to $\sum_{\theta'\in\Theta_{\mathcal{B}}(w)} \mathrm{wt}(\theta') = \|\mathcal{B}\|(w)$. Thus $h(L(\mathcal{A})) = \|\mathcal{B}\|$. $\square$

We could strengthen Theorem 6 by proving $(2') \Rightarrow (1)$ where $(2')$ is obtained from (2) by dropping the ε-freeness of $\mathcal{A}$.

5 Separating the Storage from an (S, Δ)-Automaton

In this section we will characterize the language recognized by an ε-free (S,Δ)-automaton $\mathcal{A}$ as the image of the set of behaviours of the initial configuration of $\mathcal{A}$ under a simple transducer mapping. Note that $\mathcal{A}$ need not be unambiguous. Our proof follows closely the technique in the proof of [14, Thm. 3.26].

Let c_0 be the initial configuration of $\mathcal{A}$ and θ a computation of $\mathcal{A}$, i.e., $\theta \in \Theta_{\mathcal{A}}(q_0, w, c_0)$ for some w. By dropping from θ all references to states and to the input, a sequence of pairs remains where each pair consists of a predicate and an instruction. This sequence might be called a behaviour of c_0. Formally, let Ω be a finite subset of $P \times F$,[4] $c \in C$, and $v = (p_1, f_1)\dots(p_n, f_n) \in \Omega^*$. We say that v is an *Ω-behaviour of* c if for every i with $i \in [n]$ we have (i) $p_i(c') = \mathrm{true}$ and (ii) $f_i(c')$ is defined where $c' = f_{i-1}(\dots f_1(c)\dots)$ (note that $c' = c$ for $i = 1$). We denote the set of all Ω-behaviours of c by $\mathrm{B}(\Omega, c)$. Note that each behaviour of c is a path in the approximation of c according to [14, Def. 3.23].

An *a-transducer* [19] is a machine $\mathcal{M} = (Q, \Omega, \Delta, \delta, q_0, Q_f)$ where Q, Ω, and Δ are alphabets (*states*, *input/output symbols*, resp.), $q_0 \in Q$ (*initial state*), $Q_f \subseteq Q$ (*final states*), and δ is a finite subset of $Q \times \Omega^* \times Q \times \Delta^*$. We say that $\mathcal{M}$ is a *simple transducer (from Ω to Δ)* if $\delta \subseteq Q \times \Omega \times Q \times \Delta$. The binary relation $\vdash_{\mathcal{M}}$ on $Q \times \Omega^* \times \Delta^*$ is defined as follows: let $(q, ww', v) \vdash_{\mathcal{M}} (q', w', vv')$ if $(q, w, q', v') \in \delta$. The *mapping induced by* $\mathcal{M}$, also denoted by $\mathcal{M}$, is the mapping $\mathcal{M}\colon \Omega^* \to \mathcal{P}(\Delta^*)$ defined by $\mathcal{M}(w) = \{v \in \Delta^* \mid (q_0, w, \varepsilon) \vdash^*_{\mathcal{M}} (q, \varepsilon, v), q \in Q_f\}$. If $\mathcal{M}$ is a simple transducer, then $\mathcal{M}(w)$ is finite for every w. For every $L \subseteq \Omega^*$ we define $\mathcal{M}(L) = \bigcup_{v\in L} \mathcal{M}(v)$.

Our goal is to prove the following theorem.

Theorem 7. *Let $S = (C, P, F, C_0)$ be a storage type. Moreover, let $L \subseteq \Delta^*$. Then the following are equivalent:*

(1) L is recognizable by some ε-free (S, Δ)-automaton.

(2) There are $c \in C$, a finite set $\Omega \subseteq P \times F$, and a simple transducer $\mathcal{M}$ from Ω to Δ such that $L = \mathcal{M}(\mathrm{B}(\Omega, c))$.

We note that (1)⇒(2) of Theorem 7 is similar to [19, Lm. 2.3] (after decomposing the simple transducer $\mathcal{M}$ from Ω to Δ according to Theorem 9).

For the proof of this theorem, we define the concept of relatedness between an ε-free (S, Δ)-automaton $\mathcal{A}$ and a simple transducer $\mathcal{M}$ with the following intention:

[4] We recall that $S = (C, P, F, C_0)$ is an arbitrary storage type.

$\mathcal{A}$ allows a computation

$$(q_0, x_1, p_1, q_1, f_1)(q_1, x_2, p_2, q_2, f_2) \dots (q_{n-1}, x_n, p_n, q_n, f_n) \ ,$$

for some states $q_1, \dots, q_{n-1}$ if and only if

$$(q_0, (p_1, f_1) \dots (p_n, f_n), \varepsilon) \vdash^*_{\mathcal{M}} (q_n, \varepsilon, x_1 \dots x_n) \ .$$

That is, while reading a behaviour of the initial configuration of $\mathcal{A}$, the simple transducer $\mathcal{M}$ produces a string which is recognized by $\mathcal{A}$. Formally, let $\mathcal{A} = (Q, \Delta, c, q_0, Q_f, T)$ be an ε-free (S, Δ)-automaton and $\mathcal{M} = (Q', \Omega, \Delta', \delta, q'_0, Q'_f)$ be a simple transducer. Then $\mathcal{A}$ *is related to* $\mathcal{M}$ if

- $Q = Q'$, $q_0 = q'_0$, $Q_f = Q'_f$,
- $\Delta = \Delta'$ and Ω is the set of all pairs (p, f) such that T contains a transition of the form (q, x, p, q', f) for some q,q', and x, and
- for every $q, q' \in Q$, $x \in \Delta$, $p \in P$, and $f \in F$ we have: $(q, x, p, q', f) \in T$ if and only if $(q, (p, f), q', x) \in \delta$.

Lemma 8. Let $\mathcal{A}$ be an ε-free (S, Δ)-automaton with initial configuration c and let $\mathcal{M}$ be a simple transducer from Ω to Δ. If $\mathcal{A}$ is related to $\mathcal{M}$, then $L(\mathcal{A}) = \mathcal{M}(\mathrm{B}(\Omega, c))$.

Proof. Let $\mathcal{A} = (Q, \Delta, c, q_0, Q_f, T)$ and $\mathcal{M} = (Q, \Omega, \Delta, \delta, q_0, Q_f)$. First we prove that $L(\mathcal{A}) \subseteq \mathcal{M}(\mathrm{B}(\Omega, c))$. Let $v \in L(\mathcal{A})$. Then $v = x_1 ... x_n$ for some $n \geq 0$ and $x_i \in \Delta$ for every $1 \leq i \leq n$. Moreover, there is a q_0-computation θ in $\Theta_{\mathcal{A}}(v)$ with $\theta = \tau_1 ... \tau_n$, such that $\tau_i \in T$ where $\tau_1 = (q_0, x_1, p_1, q_1, f_1)$, for every $2 \leq i \leq n$ we have $\tau_i = (q_{i-1}, x_i, p_i, q_i, f_i)$, and $q_n \in Q_f$. Since $\mathcal{A}$ is related to $\mathcal{M}$, we have $(q_{i-1}, (p_i, f_i), q_i, x_i) \in \delta$ for every $1 \leq i \leq n$. Hence $(q_0, w, \varepsilon) \vdash^*_{\mathcal{M}} (q_n, \varepsilon, x_1 \dots x_n)$ with $w = (p_1, f_1) \dots (p_n, f_n)$. Since $w \in \mathrm{B}(\Omega, c)$ is a behaviour of c, $v = x_1 \dots x_n$, and $q_n \in Q_f$, we obtain that $v \in \mathcal{M}(\mathrm{B}(\Omega, c))$.

Next we prove that $\mathcal{M}(\mathrm{B}(\Omega, c)) \subseteq L(\mathcal{A})$. Let $v \in \mathcal{M}(\mathrm{B}(\Omega, c))$ with $v = x_1 ... x_n$ for some $n \geq 0$ and $x_i \in \Delta$ for every $1 \leq i \leq n$. Then there is a behaviour $w \in \mathrm{B}(\Omega, c)$ of c such that $v \in \mathcal{M}(w)$. Then there are $(p_i, f_i) \in \Omega$ with $1 \leq i \leq n$ such that $w = (p_1, f_1) \dots (p_n, f_n)$. Moreover, there are $q_0, \dots, q_n \in Q$ such that $(q_0, (p_1, f_1), q_1, x_1) \in \delta$, for every $2 \leq i \leq n$: $(q_{i-1}, (p_i, f_i), q_i, x_i) \in \delta$, and $q_n \in Q_f$. Since $\mathcal{A}$ is related to $\mathcal{M}$, we have $\tau_i = (q_{i-1}, x_i, p_i, q_i, f_i) \in T$. Since $w \in \mathrm{B}(\Omega, c)$, q_0 is the initial state of $\mathcal{A}$, and $q_n \in Q_f$, we have that $\tau_1 \dots \tau_n \in \Theta_{\mathcal{A}}(v)$ and thus $v \in L(\mathcal{A})$. $\square$

Proof (of Theorem 7). (1) $\Rightarrow$ (2): Let L be recognizable by some ε-free (S, Δ)-automaton $\mathcal{A} = (Q, \Delta, c, q_0, Q_f, T)$. Let Ω be the set of all pairs (p, f) such that T contains a transition of the form (q, x, p, q', f) for some q,q', and x. We construct the simple transducer $\mathcal{M} = (Q, \Omega, \Delta, \delta, q_0, Q_f)$ by defining $(q, (p, f), q', x) \in \delta$ if and only if $(q, x, p, q', f) \in T$ for every $q, q' \in Q$, $x \in \Delta$, and $(p, f) \in \Omega$. Clearly, $\mathcal{A}$ is related to $\mathcal{M}$ and thus, by Lemma 8, we have that $L(\mathcal{A}) = \mathcal{M}(\mathrm{B}(\Omega, c))$.

(2) $\Rightarrow$ (1): Let $c \in C$, Ω a finite subset of $P \times F$, and $\mathcal{M} = (Q, \Omega, \Delta, \delta, q_0, Q_f)$ a simple transducer. First we reduce $\mathcal{M}$ to the simple transducer $\mathcal{M}' = (Q, \Omega', \Delta, \delta, q_0, Q_f)$ where Ω' is the set of all pairs (p, f) such that $(q, (p, f), q', x) \in \delta$ for some $q, q' \in Q$ and $x \in \Delta$. Obviously, $\delta \subseteq Q \times \Omega' \times Q \times \Delta$ and $\mathcal{M}(\mathrm{B}(\Omega, c)) = \mathcal{M}'(\mathrm{B}(\Omega', c))$.

Next we construct the ε-free (S, Δ)-automaton $\mathcal{A} = (Q, \Delta, c, q_0, Q_f, T)$ by defining $T = \{(q, x, p, q', f) \mid (q, (p, f), q', x) \in \delta\}$. Since $\mathcal{A}$ is related to $\mathcal{M}'$, we have that $L(\mathcal{A}) = \mathcal{M}'(\mathrm{B}(\Omega', c)) = \mathcal{M}(\mathrm{B}(\Omega, c))$ by Lemma 8. □

6 The Main Result and Its Applications

For the proof of our CS theorem for weighted automata with storage, we first recall a result for simple transducers [18, proof of Thm. 2.1].

Theorem 9. *Let Ω be an alphabet and $L \subseteq \Omega^*$ and let $\mathcal{M}\colon \Omega^* \to \mathcal{P}_{fin}(\Delta^*)$ be induced by a simple transducer $\mathcal{M}$. Then there are an alphabet Φ, two letter-to-letter morphisms $h_1\colon \Phi \to \mathbb{B}[\Omega]$ and $h_2\colon \Phi \to \mathbb{B}[\Delta]$, and a regular language $R \subseteq \Phi^*$ such that $\mathcal{M}(L) = h_2(h_1^{-1}(L) \cap R)$.*

Next we show that a letter-to-letter morphism $h_2\colon \Phi \to \mathbb{B}[\Delta]$ and an alphabetic morphism $h\colon \Delta \to K[\Sigma \cup \{\varepsilon\}]$ can be combined smoothly. We define the alphabetic morphism $(h \circ h_2)\colon \Phi \to K[\Sigma \cup \{\varepsilon\}]$ for every $x \in \Phi$ by $(h \circ h_2)(x) = h(\delta)$ if $h_2(x) = 1.\delta$ for some $\delta \in \Delta$ (recall that $|\mathrm{supp}(h_2(x))| = 1$).

Lemma 10. *Let $h_2\colon \Phi \to \mathbb{B}[\Delta]$ be a letter-to-letter morphism and $h\colon \Delta \to K[\Sigma \cup \{\varepsilon\}]$ an alphabetic morphism. Moreover, let $H \subseteq \Phi^*$ be a language. If $(h(v) \mid v \in h_2(H))$ is locally finite, then $((h \circ h_2)(w) \mid w \in H)$ is locally finite.*

Proof. Let $u \in \Sigma^*$. By assumption, we have that $\{v \in h_2(H) \mid u \in \mathrm{supp}(h(v))\}$ is finite; let us denote this set by C_u. Since h_2 is letter-to-letter, we have that $\{y \in H \mid v \in h_2(y)\}$ is finite for each $v \in h_2(H)$. Then we have: $|\{w \in H \mid u \in \mathrm{supp}((h \circ h_2)(w)\}| = \sum_{v \in C_u} |\{y \in H \mid v \in h_2(y)\}|$. Hence, $\{w \in H \mid u \in \mathrm{supp}((h \circ h_2)(w)\}$ is finite. □

Now we can prove the CS theorem for (S, Σ, K)-automata (cf. Fig.1).

Theorem 11. *Let $S = (C, P, F, C_0)$ be a storage type, Σ an alphabet, and K a unital valuation monoid. If $r \in K\langle\langle \Sigma^* \rangle\rangle$ is (S, Σ, K)-recognizable, then there are*
- *an alphabet Φ and a regular language $R \subseteq \Phi^*$,*
- *a finite set $\Omega \subseteq P \times F$ and a configuration $c \in C$,*
- *a letter-to-letter morphism $h_1\colon \Phi \to \mathbb{B}[\Omega]$, and*
- *an alphabetic morphism $h'\colon \Phi \to K[\Sigma \cup \{\varepsilon\}]$*

such that $r = h'(h_1^{-1}(\mathrm{B}(\Omega, c)) \cap R)$.

Proof. By Theorem 6 there are an alphabet Δ, an ε-free (S, Δ)-automaton $\mathcal{A}$, and an alphabetic morphism $h\colon \Delta \to K[\Sigma \cup \{\varepsilon\}]$ such that $r = h(L(\mathcal{A}))$. Hence, if K is not complete, then $\Theta_{\mathcal{A}}(w)$ is finite for every $w \in \Sigma^*$, and $(h(v) \mid v \in L(\mathcal{A}))$ is locally finite. According to Theorem 7, there are $c \in C$, a finite set $\Omega \subseteq P \times F$, and a simple transducer $\mathcal{M}$ from Ω to Δ such that $L(\mathcal{A}) = \mathcal{M}(\mathrm{B}(\Omega, c))$. Due to Theorem 9, there are an alphabet Φ, two letter-to-letter morphisms $h_1\colon \Phi \to \mathbb{B}[\Omega]$ and $h_2\colon \Phi \to \mathbb{B}[\Delta]$, and a regular language $R \subseteq \Phi^*$ such that

$$
\begin{tikzcd}
 & R \subseteq \Phi^* \arrow[dl, "h_1"'] \arrow[dr, "h_2"] \arrow[ddr, bend left, "h'"] & \\
\mathrm{B}(\Omega,c) \subseteq \Omega^* \arrow[rr, "\mathcal{M}"] & & \Delta^* \supseteq L(\mathcal{A}) \arrow[d, "h"] \\
 & & r \in K\langle\langle \Sigma^* \rangle\rangle
\end{tikzcd}
$$

Fig. 1. An illustration of the proof of Theorem 11

$\mathcal{M}(\mathrm{B}(\Omega, c)) = h_2(h_1^{-1}(\mathrm{B}(\Omega, c)) \cap R)$. Let us denote the language $h_1^{-1}(\mathrm{B}(\Omega, c)) \cap R$ by H. Thus $L(\mathcal{A}) = h_2(H)$.

Since $(h(v) \mid v \in L(\mathcal{A}))$ is locally finite if K is not complete, we have by Lemma 10 that also $((h \circ h_2)(w) \mid w \in H)$ is locally finite if K is not complete. Thus $r = (h \circ h_2)(h_1^{-1}(\mathrm{B}(\Omega, c)) \cap R)$ and we can take $h' = (h \circ h_2)$. $\square$

Finally we instantiate the storage type S in Theorem 11 in several ways and obtain the CS theorem for the corresponding class of (S, Σ, K)-recognizable weighted languages: (1) $S = \mathrm{P}^n$: K-weighted n-iterated pushdown languages. (2) $S = \mathrm{NS(TRIV)}$ where NS is the nested stack operator defined in [14, Def. 7.1]: K-weighted nested stack automata (cf. Ex. 3). (3) $S = \mathrm{SC(TRIV)}$ where SC is obtained from NS by forbidding instructions for creating and destructing nested stacks: K-weighted stack automata (weighted version of stack automata [20]). (4) $S = \mathrm{MON}(M)$ for some monoid M (cf. Ex. 4): K-weighted M-automata (weighted version of M-automata [24]).

In future investigations we will compare formally the CS theorem for quantitative context-free languages over Σ and K [9, Thm. 2(1) $\Leftrightarrow$ (2)] with our Theorem 11 for (P^1, Σ, K)-recognizable weighted languages.

References

1. Aho, A.V.: Indexed grammars – an extension of context-free grammars. J. ACM **15**, 647–671 (1968)
2. Aho, A.V.: Nested stack automata. JACM **16**, 383–406 (1969)
3. Chomsky, N., Schützenberger, M.P.: The algebraic theory of context-free languages. In: Computer Programming and Formal Systems, pp. 118–161. North-Holland, Amsterdam (1963)
4. Damm, W.: The IO- and OI-hierarchies. Theoret. Comput. Sci. **20**, 95–207 (1982)
5. Damm, W., Goerdt, A.: An automata-theoretical characterization of the OI-hierarchy. Inform. Control **71**, 1–32 (1986)
6. Denkinger, T.: A Chomsky-Schützenberger representation for weighted multiple context-free languages. In: The 12th International Conference on Finite-State Methods and Natural Language Processing (FSMNLP 2015) (2015). (accepted for publication)

7. Droste, M., Meinecke, I.: Describing average- and longtime-behavior by weighted MSO logics. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 537–548. Springer, Heidelberg (2010)
8. Droste, M., Meinecke, I.: Weighted automata and regular expressions over valuation monoids. Intern. J. of Found. of Comp. Science **22**(8), 1829–1844 (2011)
9. Droste, M., Vogler, H.: The Chomsky-Schützenberger theorem for quantitative context-free languages. In: Béal, M.-P., Carton, O. (eds.) DLT 2013. LNCS, vol. 7907, pp. 203–214. Springer, Heidelberg (2013)
10. Eilenberg, S.: Automata, Languages, and Machines - Volume A. Pure and Applied Mathematics, vol. 59. Academic Press (1974)
11. Engelfriet, J.: Iterated pushdown automata and complexity classes. In: Proc. of STOCS 1983, pp. 365–373. ACM, New York (1983)
12. Engelfriet, J.: Context-free grammars with storage. Technical Report 86–11, University of Leiden (1986). see also: arXiv:1408.0683 [cs.FL] (2014)
13. Engelfriet, J., Schmidt, E.M.: IO and OI.I. J. Comput. System Sci. **15**(3), 328–353 (1977)
14. Engelfriet, J., Vogler, H.: Pushdown machines for the macro tree transducer. Theoret. Comput. Sci. **42**(3), 251–368 (1986)
15. Engelfriet, J., Vogler, H.: High level tree transducers and iterated pushdown tree transducers. Acta Inform. **26**, 131–192 (1988)
16. Fischer, M.J.: Grammars with macro-like productions. Ph.D. thesis, Harvard University, Massachusetts (1968)
17. Fratani, S., Voundy, E.M.: Dyck-based characterizations of indexed languages. published on arXiv http://arxiv.org/abs/1409.6112 (March 13, 2015)
18. Ginsburg, S., Greibach, S.A.: Abstract families of languages. Memoirs of the American Math. Soc. **87**, 1–32 (1969)
19. Ginsburg, S., Greibach, S.A.: Principal AFL. J. Comput. Syst. Sci. **4**, 308–338 (1970)
20. Greibach, S.A.: Checking automata and one-way stack languages. J. Comput. System Sci. **3**, 196–217 (1969)
21. Greibach, S.A.: Full AFLs and nested iterated substitution. Inform. Control **16**, 7–35 (1970)
22. Harrison, M.A.: Introduction to Formal Language Theory, 1st edn. Addison-Wesley Longman Publishing Co., Inc, Boston (1978)
23. Hulden, M.: Parsing CFGs and PCFGs with a Chomsky-Schützenberger representation. In: Vetulani, Z. (ed.) LTC 2009. LNCS, vol. 6562, pp. 151–160. Springer, Heidelberg (2011)
24. Kambites, M.: Formal languages and groups as memory. arXiv:math/0601061v2 [math.GR] (October 19, 2007)
25. Kanazawa, M.: Multidimensional trees and a Chomsky-Schützenberger-Weir representation theorem for simple context-free tree grammars. J. Logic Computation (2014)
26. Maslov, A.N.: The hierarchy of indexed languages of an arbitrary level. Soviet Math. Dokl. **15**, 1170–1174 (1974)
27. Maslov, A.N.: Multilevel stack automata. Probl. Inform. Transm. **12**, 38–42 (1976)
28. Okhotin, A.: Non-erasing variants of the Chomsky–Schützenberger theorem. In: Yen, H.-C., Ibarra, O.H. (eds.) DLT 2012. LNCS, vol. 7410, pp. 121–129. Springer, Heidelberg (2012)
29. Salomaa, A., Soittola, M.: Automata-Theoretic Aspects of Formal Power Series. Texts and Monographs in Computer Science. Springer-Verlag (1978)
30. Scott, D.: Some definitional suggestions for automata theory. J. Comput. System Sci. **1**, 187–212 (1967)

31. Wand, M.: An algebraic formulation of the Chomsky hierarchy. In: Manes, E.G. (ed.) Category Theory Applied to Computation and Control. LNCS, vol. 25, pp. 209–213. Springer, Heidelberg (1975)
32. Weir, D.J.: Characterizing Mildly Context-Sensitive Grammar Formalisms. Ph.D. thesis, University of Pennsylvania (1988)
33. Yoshinaka, R., Kaji, Y., Seki, H.: Chomsky-Schützenberger-type characterization of multiple context-free languages. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 596–607. Springer, Heidelberg (2010)