Theory Comput Syst (2008) 42: 157–186
DOI 10.1007/s00224-007-9053-2

Storage Products and Linear Control of Derivations

Christian Wartena

Published online: 3 October 2007

Abstract Various automata using certain kinds of tuples of storages are defined in the literature. In this paper we investigate some possibilities to define such storages independently of automata by a restricted type of product on storages, called concatenation. It is shown that there is a strong relation between automata with concatenated pushdowns and restricted classes of linear controlled grammars. Using this result, some relations between hierarchies of automata with an ascending number of concatenated pushdowns and some well-known hierarchies of controlled grammars follow naturally.

Keywords Formal language theory · Automata theory · Linguistics · Pushdown storage

1 Introduction

In [19, 25, 26] it is argued that automata and certain types of grammars using tuples of pushdowns provide interesting models for the description of natural languages. The interest in such systems, however, is not limited to linguistics. For instance, [1] defines multi-pushdown automata and multi-pushdown grammars and [7] defines multistack pushdown automata. Tuples of pushdowns are very powerful. An automaton with two pushdowns already can simulate a Turing machine. Thus [1] uses a restriction on the accessibility of the stacks. This paper investigates the possibilities for defining such restrictions on tuples of arbitrary storages, independently of the concrete systems that may use them. Two different products are introduced, called concatenation w.r.t. reading (popping) and concatenation w.r.t. writing (pushing), respectively. If the first operation is iteratively applied to concatenate pushdowns we get exactly the storages employed by the multi-pushdown automata of [1].

C. Wartena (✉)
Telematica Instituut, Postbus 589, 7500 AN, Enschede, The Netherlands
e-mail: Christian.Wartena@telin.nl

In [24] it is shown that the classes of languages that are accepted by multi-pushdown automata of either type are *mildly context-sensitive*. Mildly context-sensitivity was defined by Aravind Joshi (see, e.g., [14]) to characterize classes of languages that are only a bit stronger than context-free languages, in the sense that they still have a number of nice properties of the context-free languages, but at the same time are strong enough to describe natural languages in an adequate manner. Though storages consisting of sequences of pushdowns formed by concatenation w.r.t. reading as well as those formed by concatenation w.r.t. writing both give rise to mildly context-sensitive classes of languages, it can be argued that especially concatenation w.r.t. writing is of interest for the study of formal properties of natural languages (cf. [24, 26]). The following two paragraphs sketch the linguistic argumentation and are not essential for the understanding of the rest of the paper, but might give the interested reader a short impression how composite storages can be used to describe structures in natural languages and why concatenation w.r.t. writing is to be preferred to the other type of concatenation.

In the first place, automata with two pushdowns concatenated w.r.t. writing accept exactly the class of languages generated by extended right-linear indexed grammars (ERLIGs). ERLIGs arise from a restriction on the rule format of linear indexed grammars (LIGs) that we have argued to be appropriate for the description of natural languages (cf. [17, 18]). In an LIG a stack of indices is passed on through a derivation from a parent node to one of its children in each rewriting step. At each point an index can be pushed onto the stack or popped from the stack. Thus a dependency between two nodes, that are not in one local subtree, can be accounted for by pushing an index and popping the same index symbol again. For linguistic description this mechanism can, e.g., be used to code the dependency between a fronted element and its "original" context, like that between a question word and its "original position", speaking in terms of generative transformational grammar. Looking at linguistic structures we found that the stack of indices is always passed to the rightmost nonterminal child. Inheriting the stack to the left always results in an ungrammatical sentence (marked with * in the examples below). The contrast is illustrated in the following example sentences, in which the underlying or original position of the question word is marked by an underscore:

1. (a) [Who did [Mary think [that John loves __]]]?
   (b) *[Who did [[that John loves __] annoy Mary]]?

More examples and an extensive discussion of the issue is found in [17, 18].

In the second place, it can be argued that a concatenation of pushdowns is a more appropriate storage to keep information about nonlocal dependencies during a derivation than the stack of indices used in LIGs: there are various types of nonlocal dependencies that we need to keep apart. This is, e.g., clear in Dutch sentences in which there is both, question word fronting and so called verb-raising (sentence 2) or in (marginally acceptable) Hungarian sentence 3 in which there is question word fronting of multiple question words and topicalization at the same time:

2. [Wie$_t$ heeft$_k$ [Gijs [[de buurman$_{_t}$ muizen $_{\_i}$ $_{\_j}$] zien laten$_j$ vangen$_i$] $_{\_k}$]]?
   Who has Gijs the neighbor mice see let catch
   "*Whom has Gijs seen the neighbor let catch mice?*"

3. A háboru$_t$, kinek$_i$ és miért$_j$ mondta $\__i$ János, hogy $\__t$ $\__j$ kitört?
   The war, whom and why said János, that has broken out.

For both examples it is easy to check that a simple stack will not suffice to account for the dependencies if it is used as sketched above, while a concatenation of two pushdowns can be used straightforwardly. In this case, the restrictions imposed on the accessibility of the storage components can be argued to be consonant with linguistic conditions on non-local dependencies if the storage is formed by concatenation w.r.t. writing but not if it is formed using the other operation (for discussion see [24]).

The classes of languages accepted by automata using a concatenation of n pushdowns constitute an infinite hierarchy in case the concatenation was done with respect to reading as well as in the case the storages were concatenated with respect to writing. Thus two hierarchies are defined, the former of which was established by [1]. The two hierarchies can be related by *inversion* of storages, defined below, or by reversal of languages. The main result of this paper shows that there is a strong connection between storage concatenation and linear control of context-free grammars. Especially it is shown that an automaton with the concatenation of a storage S and a pushdown is equivalent to a subclass of context-free grammars which are linearly controlled by an automaton using S as its storage. Using this result we can redefine both hierarchies in terms of controlled grammars. Furthermore, we directly get the inclusion of the multi-pushdown hierarchy in a hierarchy of controlled grammars established by Weir [27]. Finally, a new proof will be given for the fact that Weir's hierarchy can be embedded in the multi-pushdown hierarchy as well. The last two results were first proved in [5].

2 Concatenation of Storages

A system with two pushdowns in general can simulate a Turing machine. In [20] it was noted that the power of automata with two (or more) stacks can be reduced by imposing a restriction on the possibilities for reading from the pushdowns. This led to the definition of *multi-pushdown automata* in [1, 20]. Here, the definition of the used storages is implicit in the definition of the multi-pushdown automata. It is however possible to define storages independently of the devices that use them. The idea of defining storages without reference to automata (or grammars) was exploited in, e.g., [2, 9, 11, 21, 22]. Using the concept of abstract storages we will define two concatenation operations by explicitly putting restrictions on a tuple of two storages. These operations in turn can be used to define the kind of storage that is used by multi-pushdown automata.

In the following we assume the reader to be familiar with the fundamental concepts of formal language and automata theory, particularly with context-free grammars and pushdown automata.

2.1 Storages

Definition 1 A *storage*[1] is a quintuple $S = (C, c_\epsilon, F, id, m)$, where C is a set of configurations, $c_\epsilon \in C$ is the empty configuration, F a set of function symbols, $id \in F$ the identity symbol and m is the meaning function, which associates every $f \in F$ with a partial function $m(f) : C \to C$, such that $m(id)$ is the identity on C.

A *trivial storage*, denoted S_{triv}, is defined as $S_{\text{triv}} = (\{c\}, c, \{\mathsf{id}\}, \mathsf{id}, m)$, where c is an arbitrary object. A *pushdown* over a finite alphabet Γ is defined by[2] $S_{\text{pd}}(\Gamma) = (\Gamma^*, \epsilon, F, \mathsf{id}, m)$ with $F = \{\mathsf{id}\} \cup \{\mathsf{push}(a) \mid a \in \Gamma\} \cup \{\mathsf{pop}(a) \mid a \in \Gamma\}$ and for every $a \in \Gamma$ and $\beta \in \Gamma^*$,

$$m(\mathsf{push}(a))(\beta) = a\beta,$$
$$m(\mathsf{pop}(a))(a\beta) = \beta.$$

Usually pushdowns are defined with only one function for erasing the topmost symbol. In our definition there is for each $a \in \Gamma$ a function to erase a. The advantage of this definition is that each (partial) function $m(f)$ (with $f \in F$) is injective, which will be important for the inversion of storages. The classes of all trivial and all pushdown storages are denoted $\mathcal{S}_{\text{triv}}$ and $\mathcal{S}_{\text{pd}}$, respectively.

Definition 2 An *S-automaton* is a tuple $M = (Q, \Sigma, S, \delta, Q_I, Q_F)$, where Q is a finite set of states, Σ is the input alphabet, $S = (C, c_\epsilon, F, id, m)$ is a storage, $Q_I \subseteq Q$ and $Q_F \subseteq Q$ the set of initial and final states, respectively, and δ, the transition relation, a finite subset of $Q \times \Sigma^* \times Q \times F$.

The set $\mathrm{ID}(M) = Q \times \Sigma^* \times C$ is called the set of instantaneous descriptions. For $(q_1, vw, c_1), (q_2, w, c_2) \in \mathrm{ID}(M)$ with $w \in \Sigma^*$ we write $(q_1, vw, c_1) \vdash_M (q_2, w, c_2)$ if there exists $(q_1, v, q_2, f) \in \delta$ such that $m(f)$ is defined for c_1 and $m(f)(c_1) = c_2$. The transitive and reflexive closure $\vdash_M^*$ of $\vdash_M$ is defined as usual. We write $I \vdash_M^n K$ for $I, K \in ID(M)$ if K can be derived from I in n steps. Usually, automata accept languages either by final state or by empty configuration. Here it is convenient to require both, i.e. to accept by *empty configuration and final state* (cf. [11, Sect. 4.1]). Thus the *language accepted by M* is defined as $L(M) = \{w \in \Sigma^* \mid (q_0, w, c_\epsilon) \vdash_M^* (q, \epsilon, c_\epsilon)$ for some $q_0 \in Q_I$ and $q \in Q_F\}$.

Note that S-automata are defined here as one-way, nondeterministic automata. Sometimes we will use the "sugared notation" $(p, v, q, f_1 \,\&\, f_2) \in \delta$ as an abbreviation for $(p, v, p', f_1) \in \delta$ and $(p', \epsilon, q, f_2) \in \delta$ where p' is a new state. Alternatively, we can see $f_1 \,\&\, f_2$ as a new function symbol for which $m(f_1 \,\&\, f_2)(c) = m(f_2)(m(f_1)(c))$ for each $c \in C$ for which $m(f_1)(c))$ and $(m(f_2)(m(f_1)(c))$ are defined. Finally, for pushdowns and $v \in \Gamma^*$ we will write $\mathsf{push}(v)$ for $\mathsf{push}(a_n)$

[1] Often the definition of a storage is more elaborate, using, e.g., predicates and sets of initial and final configurations.

[2] Throughout the paper the following notational conventions are used. The empty string is denoted by ϵ. For each set V the notation V_ϵ is used as an abbreviation for $V \cup \{\epsilon\}$. As usual V^* and V^+ denote Kleene closure and positive Kleene closure of V, resp.

$\&\,\mathsf{push}(a_{n-1})\cdots\&\,\mathsf{push}(a_1)$ if $v = a_1\cdots a_{n-1}a_n$, with $a_1,\ldots,a_n \in \Gamma$, and we will take $\mathsf{push}(\epsilon)$ to be identical with id.

For each storage S we set $L(S) = \{L(M) \mid M \text{ is an } S\text{-automaton}\}$, the class of languages accepted by S-automata. For any class of storages $\mathcal{S}$ we define $L(\mathcal{S}) = \bigcup_{S\in\mathcal{S}} L(S)$.

Definition 3 For storages $S_1 = (C_1, c_\epsilon^1, F_1, id_1, m_1)$ and $S_2 = (C_2, c_\epsilon^2, F_2, id_2, m_2)$, S_1 and S_2 are *isomorphic*, written $S_1 \cong S_2$, if there exist bijections $h_{\mathrm{cn}} : C_1 \to C_2$ and $h_{\mathrm{fn}} : F_1 \to F_2$ such that:

1. $h_{\mathrm{cn}}(c_\epsilon^1) = c_\epsilon^2$;
2. $h_{\mathrm{cn}}(m_1(f)(c)) = m_2(h_{\mathrm{fn}}(f))(h_{\mathrm{cn}}(c))$ for each $c \in C_1$ and $f \in F_1$.

Proposition 1 *For storages S_1 and S_2, $S_1 \cong S_2$ implies $L(S_1) = L(S_2)$.*

2.2 Operations on Storages

The storage that the multi-pushdown automata of [1] use can be considered as a tuple of pushdowns if writing (pushing) is concerned, but with respect to reading (popping) the storage behaves like a single pushdown. A possible configuration of such a storage is shown in Fig. 1. In [24–26] it is argued that it is more appropriate for the description of non-local dependencies in natural languages to leave reading unrestrained and restrict the writing operations. Both storages can be defined by iterative application of an appropriate *concatenation* operation. Concatenation will be defined below on the basis of the product of storages by restricting the domain of the functions. The two types of concatenation we consider can be related by *inversion* of storages.

Definition 4 For storages $S_1 = (C_1, c_\epsilon^1, F_1, id_1, m_1)$ and $S_2 = (C_2, c_\epsilon^2, F_2, id_2, m_2)$, the *product* of S_1 and S_2 is $S_1 \circ S_2 = (C_1 \times C_2, (c_\epsilon^1, c_\epsilon^2), F, \triangleleft id_1, m)$ where $F = \{\triangleleft f \mid f \in F_1\} \cup \{\triangleright f \mid f \in F_2\}$, and for every $c_1 \in C_1$ and $c_2 \in C_2$

$$m(\triangleleft f)((c_1,c_2)) = (m_1(f)(c_1), c_2) \quad \text{if } m_1(f)(c_1) \text{ is defined,}$$
$$\text{otherwise it is undefined,}$$
$$m(\triangleright f)((c_1,c_2)) = (c_1, m_2(f)(c_2)) \quad \text{if } m_2(f)(c_2) \text{ is defined,}$$
$$\text{otherwise it is undefined.}$$

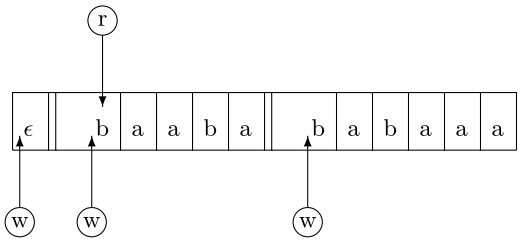


Fig. 1 Example of a configuration of a $S_{\mathrm{pd}} \circ_r S_{\mathrm{pd}} \circ_r S_{\mathrm{pd}}$ storage with marking of possibilities for reading and writing

The set of *semi-empty* configurations of $S_1 \circ S_2$ is defined as $C_{SE} = \{c_\epsilon^1\} \times C_2$. For two classes of storages $\mathcal{S}_1$ and $\mathcal{S}_2$ the product is defined straightforwardly as $\mathcal{S}_1 \circ \mathcal{S}_2 = \{S_1 \circ S_2 \mid S_1 \in \mathcal{S}_1 \text{ and } S_2 \in \mathcal{S}_2\}$.

We assume that $\lhd$ is a fixed injective mapping on symbols, i.e., if f and f' are distinct symbols then so are $\lhd f$ and $\lhd f'$ (and similarly for $\rhd$). Given a set of (function) symbols F we define the *extension* of F as the smallest set F' of symbols containing F and closed under $\lhd$ and $\rhd$.

As mentioned above, the product of two storages is mostly too powerful for our purposes. Following an idea of [1] we can reduce this power by restricting the operations that can be applied to the second component. A rather general definition was suggested by Budach [3].

Definition 5 For storages $S_1 = (C_1, c_\epsilon^1, F_1, id_1, m_1)$ and $S_2 = (C_2, c_\epsilon^2, F_2, id_2, m_2)$ such that $S_1 \circ S_2 = (C, c_\epsilon, F, id, m)$, and for a set of symbols K, the *K-product* of S_1 and S_2 is $S_1 \circ_K S_2 = (C, c_\epsilon, F, id, m')$ with[3]

$$\begin{array}{ll} m'(\rhd f) = m(\rhd f)|_{C_{SE}} & \text{if } f \in K', \\ m'(\rhd f) = m(\rhd f) & \text{if } f \notin K', \\ m'(\lhd f) = m(\lhd f) & \end{array}$$

where K' denotes the extension of K. For classes of storages $\mathcal{S}_1$ and $\mathcal{S}_2$ and a set of symbols K, the K-product is $\mathcal{S}_1 \circ_K \mathcal{S}_2 = \{S_1 \circ_K S_2 \mid S_1 \in \mathcal{S}_1 \text{ and } S_2 \in \mathcal{S}_2\}$.

Note that $m(\rhd f)((c_1, c_2))$ is undefined if $f \in K'$ and $c_1 \neq c_\epsilon^1$. The restricted product still has some of the usual properties of a product. Every K-product is associative, i.e., for any storages S_1, S_2, S_3 and any set K, it is easy to see that $(S_1 \circ_K S_2) \circ_K S_3 \cong S_1 \circ_K (S_2 \circ_K S_3)$ using the obvious bijections h_{cn} and h_{fn}. The trivial storage serves as a neutral element for all K-products, i.e., for any storage S and any set K, we have $S \circ_K S_{\text{triv}} \cong S_{\text{triv}} \circ_K S \cong S$.

In the following the K-products for two sets K will be of special interest. First we will consider the set r, which determines what operations (for pushdowns and similar storages) are considered as reading operations. For pushdowns and products of pushdowns r is defined as $\{\mathsf{pop}(a) \mid a \text{ a symbol}\}$. Below we sometimes will call the r-product of two stores the *concatenation with respect to reading*. The counterpart of the set r is w which, for pushdowns and products of pushdowns, is defined as $\{\mathsf{push}(a) \mid a \text{ a symbol}\}$. The product $\circ_w$ will be called *concatenation with respect to writing*. Concatenation of pushdowns w.r.t. reading yields exactly the storages that were (implicitly) studied in [1]. The other kind of concatenation yields storages that can be viewed as a stack with several reading heads, but only one writing head. Since this type is interesting for the description of natural languages it would be useful if we could transfer the results concerning complexity and parsability of [1] to concatenations of pushdowns w.r.t. writing. Below we will see that this is possible, since

[3]For any (partial) function $f : A \to B$ and any $U \subseteq A$ the *restriction of f to U*, denoted $f|_U$, is defined as $f|_U(u) = f(u)$ if $u \in U$ and $f(u)$ is defined and undefined otherwise.

w- and r-products with a pushdown as the second argument can be related by *inversion* of storages.

Definition 6 Let $S = (C, c_\epsilon, F, id, m)$ be a storage such that for each $f \in F$, $m(f)$ is a (partial) injective function. Then S is an *invertible* storage. The *inverse* of an invertible storage S is $S^{\text{inv}} = (C, c_\epsilon, F, id, m')$ where m' is defined by[4]

$$m'(f) = m(f)^{-1} \quad \text{for each } f \in F.$$

For any class of invertible storages $\mathcal{S}$ we define $\mathcal{S}^{\text{inv}} = \{S^{\text{inv}} \mid S \in \mathcal{S}\}$.

Note that $(S^{\text{inv}})^{\text{inv}} = S$. Furthermore, for any finite alphabet Γ it can be shown that $S_{\text{pd}}(\Gamma)^{\text{inv}} \cong S_{\text{pd}}(\Gamma)$. In order to do so we have to define bijections h_{cn} between the configurations and h_{fn} between the function symbols that will constitute the isomorphism. We let h_{cn} be the identity on Γ^*. Thus the first condition of Definition 3, $h_{\text{cn}}(\epsilon) = \epsilon$, is fulfilled. We define h_{fn} by setting $h_{\text{fn}}(\mathsf{push}(a)) = \mathsf{pop}(a)$, $h_{\text{fn}}(\mathsf{pop}(a)) = \mathsf{push}(a)$ and $h_{\text{fn}}(\mathsf{id}) = \mathsf{id}$ for each $a \in \Gamma$. Now it is easy to see that for each $a \in \Gamma$ and $\alpha \in \Gamma^*$, $m(\mathsf{push}(a))(\alpha) = m(\mathsf{pop}(a))^{-1}(\alpha)$ and $m(\mathsf{pop}(a))(\alpha) = m(\mathsf{push}(a))^{-1}(\alpha)$ if m is the meaning function of the pushdown. Thus the second condition of Definition 3 is fulfilled as well.

However, in the context of a restricted product we cannot simply interchange $S_{\text{pd}}(\Gamma)^{\text{inv}}$ and $S_{\text{pd}}(\Gamma)$. For instance, $S \circ_r S_{\text{pd}}(\Gamma)$ and $S \circ_r S_{\text{pd}}(\Gamma)^{\text{inv}}$ are not isomorphic. This is due to the fact that the concatenation operator restricts in both cases the application of $\text{pop}(a)$ (for each $a \in \Gamma$) while $\text{pop}(a)$ in the latter storage corresponds to $\text{push}(a)$ in the former.

Proposition 2 *Let S_1 and S_2 be invertible storages, and let K be a set of symbols. Then $(S_1 \circ_K S_2)^{\text{inv}} = S_1^{\text{inv}} \circ_K S_2^{\text{inv}}$.*

Proof Let $S_1 = (C_1, c_\epsilon^1, F_1, id_1, m_1)$ and $S_2 = (C_2, c_\epsilon^2, F_2, id_2, m_2)$ be two storages. Furthermore, let m_A and m_B be the meaning functions of $(S_1 \circ_K S_2)^{\text{inv}}$ and $S_1^{\text{inv}} \circ_K S_2^{\text{inv}}$, respectively, and let $F = \{\lhd f \mid f \in F_1\} \cup \{\rhd f \mid f \in F_2\}$.

Both composite storages have the set $C_1 \times C_2$ as set of configurations, $(c_\epsilon^1, c_\epsilon^2)$ as empty configuration, F as set of function symbols and $\lhd id_1$ as identity symbol. Thus, it remains to show that $m_A = m_B$, i.e., that $m_A(f)(c_1, c_2) = m_B(f)(c_1, c_2)$ for each $c_1 \in C_1$, $c_2 \in C_2$ and $f \in F$. If $f = \lhd f'$ for some $f' \in F_1$ this is indeed the case since we find by the definitions of product and inversion $m_A(\lhd f')(c_1, c_2) = m_B(\lhd f')(c_1, c_2) = (m_1(f')^{-1}(c_1), c_2)$. In case $f = \rhd f'$ we find $m_A(\rhd f')(c_1, c_2) = m_B(\rhd f')(c_1, c_2) = (c_1, m_2(f')^{-1}(c_2))$. Note, moreover, that operations on the second component in both cases are restricted to configurations with c_ϵ^1 as the first component for the same function symbols. □

Proposition 3 *Let S_1 and S_2 be isomorphic storages with sets of function symbols F_1 and F_2, resp., and let K_1 and K_2 be two sets such that $f \in K_1$ iff $h(f) \in K_2$ for each*

[4]For each injective function f the *inverse of* f, denoted f^{-1} is defined by $f^{-1}(x) = y$ if $f(y) = x$.

$f \in F_1$, where h is the bijection between F_1 and F_2 that constitutes the isomorphism of S_1 and S_2. Then for each storage S, $S \circ_{K_1} S_1 \cong S \circ_{K_2} S_2$.

Proposition 4 *For any invertible storage S and any finite set of symbols Γ the following holds*:

$$\text{(a)} \quad S \circ_r S_{\mathrm{pd}}(\Gamma) \cong (S^{\mathrm{inv}} \circ_w S_{\mathrm{pd}}(\Gamma))^{\mathrm{inv}},$$

$$\text{(b)} \quad S \circ_w S_{\mathrm{pd}}(\Gamma) \cong (S^{\mathrm{inv}} \circ_r S_{\mathrm{pd}}(\Gamma))^{\mathrm{inv}}.$$

Proof Let S be an invertible storage, let Γ be a finite alphabet and let h_{fn} be the permutation of the function symbols that constitutes the isomorphism of $S_{\mathrm{pd}}(\Gamma)$ and $S_{\mathrm{pd}}(\Gamma)^{\mathrm{inv}}$. Note that $f \in w$ iff $h_{\mathrm{fn}}(f) \in r$, since $h_{\mathrm{fn}}(\mathsf{pop}(a)) = \mathsf{push}(a)$ and $h_{\mathrm{fn}}(\mathsf{push}(a)) = \mathsf{pop}(a)$ for each stack symbol a. By Proposition 3 we find: $S \circ_r S_{\mathrm{pd}}(\Gamma) \cong S \circ_w S_{\mathrm{pd}}(\Gamma)^{\mathrm{inv}}$ and by Proposition 2: $S \circ_w S_{\mathrm{pd}}(\Gamma)^{\mathrm{inv}} = (S^{\mathrm{inv}} \circ_w S_{\mathrm{pd}}(\Gamma))^{\mathrm{inv}}$.

The proof of part (b) can be done analogously to the proof of the first equation. □

Inversion of a storage corresponds to reversal[5] of the accepted language:

Proposition 5 *$L(S^{\mathrm{inv}}) = L(S)^R$ for any invertible storage S.*

Proof It clearly suffices to prove that $L(S)^R \subseteq L(S^{\mathrm{inv}})$ (by the idempotency of inversion and reversal). Let $S = (C, c_\epsilon, F, id, m)$ be a storage and let $M = (Q, \Sigma, S, \delta, Q_I, Q_F)$ be an S-automaton. Now construct an S^{inv}-automaton M' such that $L(M') = L(M)^R$. We define $M' = (Q, \Sigma, S^{\mathrm{inv}}, \delta', Q_F, Q_I)$ with $S^{\mathrm{inv}} = (C, c_\epsilon, F, id, m')$ and

$$\delta' = \{(q_2, x, q_1, f) \mid (q_1, x, q_2, f) \in \delta\}.$$

Now it can be shown by induction on the number of transitions that $(q_1, w, c_1) \vdash_M^n (q_2, \epsilon, c_2)$ iff $(q_2, w^R, c_2) \vdash_{M'}^n (q_1, \epsilon, c_1)$. If $n = 0$ the assertion is trivially true. Assume the assertion is true for some $n \in \mathbb{N}$.[6]

Only if. Consider a computation of length $n + 1$ in M:

$$(q_1, aw, c_1) \vdash_M (q_2, w, c_2) \vdash_M^n (q_3, \epsilon, c_3).$$

The first transition is licensed by some $(q_1, a, q_2, f) \in \delta$ such that $m(f)(c_1) = c_2$. From the induction hypothesis we know that there is a computation $(q_3, w^R, c_3) \vdash_{M'}^n (q_2, \epsilon, c_2)$. By the construction we have $(q_2, a, q_1, f) \in \delta'$ and $m'(f)(c_2) = m(f)^{-1}(c_2) = c_1$. Thus we find the following computation in M':

$$(q_3, w^R a, c_3) \vdash_{M'}^n (q_2, a, c_2) \vdash_{M'} (q_1, \epsilon, c_1).$$

[5] For an alphabet Σ and a string $w \in \Sigma^*$ the reversal w^R of w is defined by setting $\epsilon^R = \epsilon$ and $(av)^R = v^R a$ with $a \in \Sigma$ and $v \in \Sigma^*$. For a language L the reversal of L is defined as $L^R = \{w \mid w^R \in L\}$. For a class of languages $\mathcal{L}$ the reversal of $\mathcal{L}$ is defined as $\mathcal{L}^R = \{L \mid L^R \in \mathcal{L}\}$.

[6] $\mathbb{N}$ denotes the set of all non-negative integers.

If. Consider a computation of length $n+1$ in M':

$$(q_3, w^R a, c_3) \vdash_{M'}^{n} (q_2, a, c_2) \vdash_{M'} (q_1, \epsilon, c_1).$$

The last transition was licensed by some $(q_2, a, q_1, f) \in \delta'$. Consequently, there is a $(q_1, a, q_2, f) \in \delta$. Thus we find the following computation in M:

$$(q_1, aw, c_1) \vdash_{M} (q_2, w, c_2) \vdash_{M}^{n} (q_3, \epsilon, c_3).$$

From the assertion it easily follows that $L(M') = L(M)^R$. □

3 Linear Controlled Grammars

Linear control of context-free grammars (CFGs) is defined in [27]. The definition is twofold. The first part says which paths have to be controlled; the second part defines the control itself.

Definition 7 A *linear distinguished grammar (LDG)* is a quadruple $G = (N, \Sigma, R, A_{\text{in}})$, where N and Σ are disjoint finite sets of nonterminal and terminal symbols, respectively, $A_{\text{in}} \in N$ is the start symbol, and R is a finite set of production rules of the form: $A \to \beta_1 X ! \beta_2$ or $A \to \epsilon$ with $A \in N$, $X \in N \cup \Sigma$, called the *distinguished symbol*, $\beta_1, \beta_2 \in (N \cup \Sigma)^*$, and ! a special symbol not in $N \cup \Sigma$. For each LDG we assume that there is a finite alphabet P and a bijection $\rho : R \to P$. A *linear controlled grammar* (LCG) is a pair $K = (G, H)$, where G is an LDG and H is a language over P, called the control language.

The sets of nonterminal and terminal objects of K are respectively defined as $O_N(K) = N \times P^*$ and $O_\Sigma(K) = \Sigma_\epsilon \times P^*$. Let $O(K) = O_N(K) \cup O_\Sigma(K)$. A string $\sigma \in O(K)^*$ is said to derive a string $\tau \in O(K)^*$, written $\sigma \underset{G}{\Rightarrow} \tau$, if either case 1 or case 2:

1. $\sigma = \gamma(A, \omega)\delta$, $r = A \to \beta_1 X ! \beta_2 \in R$, $\tau = \gamma\beta_1'(X, \omega\rho(r))\beta_2'\delta$,
2. $\sigma = \gamma(A, \omega)\delta$, $r = A \to \epsilon \in R$, $\tau = \gamma(\epsilon, \omega\rho(r))\delta$

where $A \in N$, $X \in N \cup \Sigma$, $\beta_1, \beta_2 \in (N \cup \Sigma)^*$, $\gamma, \delta \in O(K)^*$, $\omega \in P^*$, and β_1' and β_2' are obtained from β_1 and β_2 resp. by replacing every symbol $Y \in N \cup \Sigma$ by (Y, ϵ). In case 1 $(X, \omega\rho(r))$ and in case 2 $(\epsilon, \omega\rho(r))$ is called the *distinguished child* of (A, ω). The reflexive and transitive closure of $\underset{G}{\Rightarrow}$, denoted by $\underset{G}{\Rightarrow}^*$, is defined as usual. The language generated by K is defined as $L(K) = \{a_1 a_2 \cdots a_n \mid (A_{\text{in}}, \epsilon) \underset{G}{\Rightarrow}^* (a_1, \omega_1)(a_2, \omega_2) \cdots (a_n, \omega_n)$ and $a_i \in \Sigma_\epsilon,\ \omega_i \in H_\epsilon$ for $1 \le i \le n, n \in \mathbb{N}\}$. In the following we will not distinguish between R and P and write r if $\rho(r)$ is intended in order to obtain better readability.

Example 1 Let $G = (\{S, T\}, \{a, b, c, d\}, R, S)$ be an LDG with $R = \{r_1 = S \to \epsilon,\ r_2 = S \to aS!d, r_3 = S \to ST!S, r_4 = T \to bT!c,\ r_5 = T \to \epsilon\}$ and let $H = \{r_1\} \cup \{r_2^n r_3 r_4^n r_5 \mid n \in \mathbb{N}\}$ be a (context-free) control language.

It can be shown that the LCG $K = (G, H)$ generates the (obviously non-context-free) language L that can be defined as follows.

$$\begin{aligned} L_0 &= \{\epsilon\}, \\ L_i &= \{a^n v b^n c^n w d^n \mid n \in \mathbb{N} \text{ and } v, w \in L_{i-1}\}, \\ L &= \bigcup_{i=0}^{\infty} L_i. \end{aligned}$$

In order to refer to objects in a derivation it is sometimes assumed that the objects have addresses in $\mathbb{N}^*$. In the following we will use two different address assignments, *leftmost* and *inside-out* address assignment. In each case the address of $(A_{\text{in}}, \epsilon)$ is ϵ. Suppose a string $\sigma = \alpha X \beta \in O(K)^*$ derives a string τ rewriting the object X with address ξ into new objects $Y_1 Y_2 \cdots Y_i \cdots Y_n$ with Y_i the distinguished child of X. If the address assignment is leftmost then the address of each Y_k is ξk for each $1 \leq k \leq n$. In the case of inside-out assignment the address of Y_k is $\xi(i - k + 1)$ for $1 \leq k \leq i$ and ξk for $i < k \leq n$. For each object in α and β that is not rewritten, the address in τ is the same as in σ. A sequence of strings of objects $\sigma_1, \ldots, \sigma_n$ such that $\sigma_i \Rightarrow \sigma_{i+1}$ is called a derivation (of σ_n from σ_1). If in each step the nonterminal object with the lexicographically[7] smallest address is rewritten then the derivation is called leftmost in case the address assignment is leftmost and inside-out in case the address assignment is inside-out.

Let K be an LCG. A derivation tree in K is defined as for a normal CFG. A *spine* is a sequence of nodes $\chi_1, \ldots, \chi_n$ such that χ_{i+1} is the distinguished child of χ_i and χ_n, the *foot* of the spine, is a leaf. Since these concepts are used only informally we do not have to make them more precise.

The class of all LDGs is denoted by $\mathcal{G}_{\text{LD}}$. Furthermore, for any class of grammars $\mathcal{G}$ for which control by a control language is defined and for any class of languages $\mathcal{L}$, let $\mathcal{G}/\mathcal{L} = \{(G, H) \mid G \in \mathcal{G} \text{ and } H \in \mathcal{L}\}$ and for any class of grammars $\mathcal{G}$ let $L(\mathcal{G}) = \{L(G) \mid G \in \mathcal{G}\}$.

In [17, 18] restrictions on linear indexed grammars (LIGs)[8] were studied that apply to LDGs as well. The two restrictions that will play an important role in the following can be verbalized as follows: either the distinguished symbol in each rule has always to be the leftmost nonterminal child or it has to be the rightmost nonterminal child in each rule. Following the terminology for the LIG restrictions we will call LDGs obeying these conditions *extended left LDGs* and *extended right LDGs*, respectively.

[7] The lexicographic ordering relation $<_{\text{lex}}$ on $\mathbb{N}^*$ is defined by: $\chi <_{\text{lex}} \chi j \omega$ and $\chi i \psi <_{\text{lex}} \chi j \omega$ for all $\chi, \psi, \omega \in \mathbb{N}^*$ and $i, j \in \mathbb{N}$ with $i < j$.

[8] In this paper the term LIG always denotes the formalism defined in [10], which should not be confused with the grammars defined by [8] with the same name.

Definition 8 An LDG $G = (N, \Sigma, R, A_{\text{in}})$ is an *extended left LDG (ELLDG)* if each production is of one of the forms (1a), (2) or (3):

$$\begin{array}{ll} (1a) & A \to wB!\beta, \\ (1b) & A \to \beta B!w, \\ (2) & A \to va!w, \\ (3) & A \to \epsilon \end{array}$$

with $A, B \in N$, $\beta \in (N \cup \Sigma)^*$, $a \in \Sigma$ and $v, w \in \Sigma^*$. If each production is of the form (1b), (2) or (3), G is an *extended right LDG (ERLDG)*. The class of ELLDGs (ERLDGs) is denoted by $\mathcal{G}_{\text{ELLD}}$ ($\mathcal{G}_{\text{ERLD}}$).[9]

If a grammar is an ELLDG or an ERLDG it is clear for each rule which nonterminal symbol on the right hand side is the distinguished one, even if the !-symbol is not present. Therefore, we usually drop this symbol. Similarly, we sometimes leave out this marker from unary rules in LDGs.

By symmetry it is easy to verify the following proposition.

Proposition 6 *For any class of languages* $\mathcal{L}$, $L(\mathcal{G}_{\text{LD}}/\mathcal{L}) = L(\mathcal{G}_{\text{LD}}/\mathcal{L})^R$ *and* $(L(\mathcal{G}_{\text{ERLD}}/\mathcal{L}))^R = L(\mathcal{G}_{\text{ELLD}}/\mathcal{L})$.

Proof Let $K = (G, H)$ be a linear controlled grammar with $G = (N, \Sigma, R, A_{\text{in}})$ an LDG. Now we construct an LDG $G' = (N, \Sigma, R', A_{\text{in}})$ with $R' = \{A \to \beta_2^R X!\beta_1^R \mid A \to \beta_1 X!\beta_2 \in R\} \cup \{A \to \epsilon \mid A \to \epsilon \in R\}$ where $A \in N$, $X \in N \cup \Sigma$ and $\beta_1, \beta_2 \in (N \cup \Sigma)^*$. We set $K' = (G', H)$. Obviously, $L(K') = (L(K))^R$. Finally, note that G' is an ELLDG if G is an ERLDG and vice versa. □

Another useful property that is easy to show is the following proposition which in fact is a variation of Theorem 2.2 of [12].

Proposition 7 *For each class of languages* $\mathcal{L}$ *closed under homomorphism and right concatenation with a symbol*

$$\begin{array}{ll} (a) & \mathcal{L} \subseteq L(\mathcal{G}_{\text{ERLD}}/\mathcal{L}), \\ (b) & \mathcal{L}^R \subseteq L(\mathcal{G}_{\text{ELLD}}/\mathcal{L}). \end{array}$$

Proof Let $L \in \mathcal{L}$ with $L \subseteq \Sigma^*$ for some alphabet Σ. We construct an LCG $K = (G, H)$ with G an ERLDG such that $L(K) = L$ in the following way: $G = (\{S\}, \Sigma, R, \{S\})$ with $R = \{S \to aS! \mid a \in \Sigma\} \cup \{S \to \epsilon\}$. Define a homomorphism $h : \Sigma^* \to R^*$ by setting $h(a) = S \to aS!$ and extending it to strings in the usual way. Now H can be defined as $H = \{h(w) \cdot r \mid w \in L \text{ and } r = S \to \epsilon\}$. The proof of (b) is almost the same and can be done using productions of the form $S \to S!a$. □

[9]Following this terminology an LLDG is an LDG in which each production has the form $A \to B!\beta$, $A \to a!w$ or $A \to \epsilon$. RLDGs can be defined symmetrically. RLDGs and LLDGs were investigated in [17, 18], but do not play a role in the present paper.

In the following a further restricted class of controlled grammars called *controlled linear context-free grammars* is used.

Definition 9 Let $G = (N, \Sigma, R, A_{\text{in}})$ be a CFG. G is a *linear grammar* if each production is of one of the following two forms: $A \to vBw$ or $A \to w$, where $A, B \in N$ and $v, w \in \Sigma^*$. The class of linear grammars is denoted by $\mathcal{G}_{\text{lin}}$.

We finish the section on controlled grammars with some useful closure properties of controlled languages.

Proposition 8 *For any class of languages $\mathcal{L}$ containing all finite languages and closed under union and right concatenation with a symbol, $L(\mathcal{G}_{\text{LD}}/\mathcal{L})$ is closed under substitution, concatenation and Kleene+.*

Proof We first show closure under substitution. With regard to the underlying LDGs of the original and the substituting languages the proof is almost the same as for the corresponding proposition for context-free languages. Thus if $K = (G, H)$ is an LCG with $G = (N, \Sigma, R, A_{\text{in}})$ an LDG and for each $a \in \Sigma$ we have an LCG $K_a = (G_a, H_a)$ with G_a an LDG and S_a the start symbol of G_a, we construct a new LCG $K' = (G', H')$ in the following way: we assume w.l.o.g. that the symbols (and thus the rules) of the involved grammars are distinct. As usual the rules of G' are obtained by replacing every symbol $a \in \Sigma$ in the rules of G by S_a, with the following exceptions. If the symbol a in the original rule is followed by an exclamation mark, the substituted symbol S_a will not be distinguished by an exclamation mark. Instead, a new nonterminal symbol X (neither used in G nor in any of the grammars G_a) followed by the exclamation mark is added to the right-hand side of the rule and a rule $X \to \epsilon$ is added to the set of rules. Similarly, each rule $A \to \epsilon$ of G is changed into $A \to X$, for that same nonterminal X.

The control language is defined as $H' = \{w \cdot r \mid w \in H \text{ and } r = X \to \epsilon\} \cup \bigcup_{a \in \Sigma} H_a$. Since $\mathcal{L}$ is closed under union and right concatenation with a symbol, $H' \in \mathcal{L}$.

For closure under concatenation consider two languages L_1 and L_2, respectively generated by the LCGs $K_1 = (G_1, H_1)$ and $K_2 = (G_2, H_2)$ with $G_1 = (N_1, \Sigma_1, R_1, A^1_{\text{in}})$ and $G_2 = (N_2, \Sigma_2, R_2, A^2_{\text{in}})$ two LDGs. We again assume that the symbols (and thus the rules) of both grammars are distinct. We construct a grammar $K = (G, H)$ with $G = (N, \Sigma, R, A_{\text{in}})$ where $N = \{X, A_{\text{in}}\} \cup N_1 \cup N_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$, $R = \{X \to \epsilon, A_{\text{in}} \to A^1_{\text{in}} X! A^2_{\text{in}}\} \cup R_1 \cup R_2$ and X, $A_{\text{in}} \notin N_1 \cup N_2$ are new symbols. The control language is defined as $H = \{r_1 r_2 \mid r_1 = A_{\text{in}} \to A^1_{\text{in}} X! A^2_{\text{in}} \text{ and } r_2 = X \to \epsilon\} \cup H_1 \cup H_2$.

Finally, consider a language $L = L(K)$ with $K = (G, H)$ is an LCG with $G = (N, \Sigma, R, A_{\text{in}})$. The language L^+ is generated by $K' = (G', H')$ with $G' = (N', \Sigma, R', A'_{\text{in}})$ where $N' = \{X, A'_{\text{in}}\} \cup N$, $R = \{X \to \epsilon, A'_{\text{in}} \to X! A_{\text{in}} A'_{\text{in}}, A'_{\text{in}} \to X! A_{\text{in}}\} \cup R$ and X, $A'_{\text{in}} \notin N$ are new symbols. The control language is finally defined as $H' = \{r_1 r_3, r_2 r_3 \mid r_1 = A'_{\text{in}} \to X! A_{\text{in}} A'_{\text{in}} \text{ and } r_2 = A'_{\text{in}} \to X! A_{\text{in}} \text{ and } r_3 = X \to \epsilon\} \cup H_1 \cup H_2$.

For all three assertions it is a standard exercise to show that the constructed grammars indeed generate the required languages. Finally, note that for the proof of the

closure under concatenation and Kleene+ we did not need the closure of $\mathcal{L}$ under right concatenation with a symbol, whereas for the proof of closure under substitution it is not required that $\mathfrak{L}$ contains all finite languages. □

Proposition 9 *For any class of languages* $\mathcal{L}$, $L(\mathcal{G}_{\mathrm{LD}}/\mathcal{L})$, $L(\mathcal{G}_{\mathrm{ELLD}}/\mathcal{L})$ *and* $L(\mathcal{G}_{\mathrm{ERLD}}/\mathcal{L})$ *are closed under homomorphism. If* $\mathcal{L}$ *is closed under union and left concatenation with a symbol,* $L(\mathcal{G}_{\mathrm{LD}}/\mathcal{L})$, $L(\mathcal{G}_{\mathrm{ELLD}}/\mathcal{L})$ *and* $L(\mathcal{G}_{\mathrm{ERLD}}/\mathcal{L})$ *are closed under union and left and right concatenation with a symbol.*

Proof Let $K = (G, H)$ be an LCG with $G = (N, \Sigma, R, A_{\mathrm{in}})$ an LDG, ELLDG or ERLDG and let $h : \Sigma^* \to \Delta^*$ be a homomorphism. We assume w.l.o.g. that $N \cap \Delta = \emptyset$ and define a homomorphism $h' : (\Sigma \cup N)^* \to (\Delta \cup N)^*$ by setting $h'(X) = h(X)$ if $X \in \Sigma$ and $h'(X) = X$ if $X \in N$ and extending it to strings in the usual way. Now construct a grammar $G' = (N, \Delta, R', A_{\mathrm{in}})$ with $R' = \{A \to h'(\beta) \mid A \to \beta \in R\}$ and let $K' = (G', H)$. Obviously $L(K') = h(L(K))$.

For the closure under union consider two LCGs $K_1 = (G_1, H_1)$ and $K_2 = (G_2, H_2)$ with $G = (N_1, \Sigma_1, R_1, A^1_{\mathrm{in}})$ and $G = (N_2, \Sigma_2, R_2, A^2_{\mathrm{in}})$ LDGs, ELLDGs or ERLDGs and $H_1, H_2 \in \mathcal{L}$. Now construct a grammar $G = (N_1 \cup N_2 \cup \{A'_{\mathrm{in}}\}, \Sigma \cup \Sigma_2, R', A'_{\mathrm{in}})$ with A'_{in} a new symbol and $R' = \{r_1 = A'_{\mathrm{in}} \to A^1_{\mathrm{in}}, r_2 = A'_{\mathrm{in}} \to A^2_{\mathrm{in}}\} \cup R_1 \cup R_2$. We define a control language $H' = \{r_1 w \mid w \in H_1\} \cup \{r_2 w \mid w \in H_2\}$ and an LCG $K' = (G', H')$. Again it is easy to see that $L(K') = L(K_1) \cup L(K_2)$. Since $\mathcal{L}$ is closed under union and left concatenation with a symbol, $H' \in \mathcal{L}$.

For the closure under right concatenation with a symbol consider an LCG $K = (G, H)$ with $G = (N, \Sigma, R, A_{\mathrm{in}})$ an LDG, ELLDG or ERLDG and $H \in \mathcal{L}$. For a symbol a construct a grammar $G = (N \cup \{A'_{\mathrm{in}}\}, \Sigma \cup \{a\}, R', A'_{\mathrm{in}})$ with A'_{in} a new symbol and $R' = \{r = A'_{\mathrm{in}} \to A_{\mathrm{in}}a\} \cup R$. We define a control language $H' = \{rw \mid w \in H\}$ and an LCG $K' = (G', H')$. Again it is easy to see that $L(K') = L(K) \cdot \{a\}$. Since $\mathcal{L}$ is closed under left concatenation with a symbol, $H' \in \mathcal{L}$. For closure under left concatenation with a symbol we can use a rule $A'_{\mathrm{in}} \to aA_{\mathrm{in}}$ and leave the rest of the proof unchanged. □

4 The Relation between Concatenation and Linear Control

In this section we will show that there is a strong relation between linear control and concatenation of storages. It is shown that each language that is accepted by an automaton with a storage S and a pushdown concatenated w.r.t. reading, is generated by an ELLDG controlled by an S-automaton, and vice versa. Using the inversion in the relation between both types of concatenation (Proposition 4) we show a symmetrical result for concatenation w.r.t. writing and ERLDGs.

Similar relations between controlled grammars and storage concatenation are shown in [23] for linear grammars and one-turn pushdowns.

The languages that are accepted by an $\mathcal{S} \circ_r \mathcal{S}_{\mathrm{pd}}$-automaton can be characterized by linear control of an ELLDG G by some $L \in L(\mathcal{S})$. In order to construct an $\mathcal{S} \circ_r \mathcal{S}_{\mathrm{pd}}$-automaton accepting the language generated by a controlled ELLDG, we let the pushdown component of the automaton's storage store the grammar symbols, simulating

a leftmost derivation. The first component corresponds to the storage of an automaton accepting the control language. Each time a symbol of the control word, i.e., a rule of the grammar, is produced, this rule is carried out: the initial terminals are read from the input, the first nonterminal is coded in a new state and all other symbols are pushed onto the second component of the store. As long as the expansion of distinguished symbols is controlled, reading from the second component is not necessary since the controlled path leads from leftmost nonterminal to leftmost nonterminal child. Since each control word has to be computed on the first component of the storage, the simulation is only possible if the configuration of that component after recognizing a control word is again the empty configuration, which is the case because recognition is by final state and empty configuration.

Lemma 1 *For each class of storages* $\mathcal{S}$,

$$L(\mathcal{G}_{\mathrm{ELLD}}/L(\mathcal{S})) \subseteq L(\mathcal{S} \circ_r \mathcal{S}_{\mathrm{pd}}).$$

Proof Let $\mathcal{S}$ be a class of storages, let $S = (C, c_\epsilon, F, id, m) \in \mathcal{S}$ and let $K = (G, L(M))$ be an LCG, with $G = (N, \Sigma, R, A_{\mathrm{in}})$ an ELLDG and $M = (Q, R, S, \delta, Q_I, Q_F)$ an automaton. Construct an automaton $M' = (Q \times N_\epsilon, \Sigma, S \circ_r S_{\mathrm{pd}}(N \cup \Sigma), \delta', Q'_I, Q'_F)$, with $Q'_I = \{(q, A_{\mathrm{in}}) \mid q \in Q_I\}$ and $Q'_F = \{(q, \epsilon) \mid q \in Q_F\}$, such that $L(M') = L(K)$ by setting

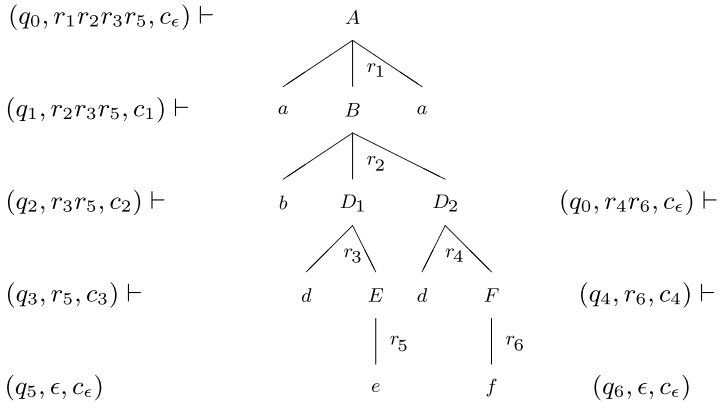
$$\begin{aligned}
\delta' = \{&((q_1, A), v, (q_2, B), \lhd f \,\&\, \rhd \mathsf{push}(\beta)) \mid \\
&\quad r = A \to vB!\beta \in R \text{ and } (q_1, r, q_2, f) \in \delta \\
&\quad \text{with } A, B \in N, v \in \Sigma^* \text{ and } \beta \in (N \cup \Sigma)^*\} && (1)\\
\cup\ \{&((q_1, A), v, (q_2, \epsilon), \lhd f \,\&\, \rhd \mathsf{push}(w)) \mid \\
&\quad r = A \to v!w \in R, v \in \Sigma^+, w \in \Sigma^* \text{ and } (q_1, r, q_2, f) \in \delta\} && (2)\\
\cup\ \{&((q_1, A), \epsilon, (q_2, \epsilon), \lhd f) \mid \\
&\quad r = A \to \epsilon \in R, \text{ and } (q_1, r, q_2, f) \in \delta\} && (3)\\
\cup\ \{&((q_1, A), \epsilon, (q_2, A), \lhd f) \mid (q_1, \epsilon, q_2, f) \in \delta \text{ and } A \in N_\epsilon\} && (4)\\
\cup\ \{&((q, \epsilon), \epsilon, (q_0, A), \rhd \mathsf{pop}(A)) \mid\ A \in N, q_0 \in Q_I, q \in Q_F\} && (5)\\
\cup\ \{&((q, \epsilon), a, (q, \epsilon), \rhd \mathsf{pop}(a)) \mid\ a \in \Sigma, q \in Q_F\}. && (6)
\end{aligned}$$

An example for the construction is given in Fig. 2.

In the following we will consider only leftmost derivations for G. In order to show that $L(M') = L(K)$ we first show that the derivation of each spine and the computation of its control word corresponds to a computation of M'. This is formally expressed in the following assertion.

For all $q_1 \in Q$, $A \in N$, $x \in \Sigma^*$, $c_1 \in C$, $q_2 \in Q_F$ and $X_1, \ldots, X_n \in N \cup \Sigma$:

$$\begin{aligned}
&((q_1, A), x, (c_1, \epsilon)) \vdash_{M'}^+ ((q_2, \epsilon), \epsilon, (c_\epsilon, X_n \cdots X_1)) \quad \text{and} \\
&M' \text{ does not use transitions of type (5) and (6) in this computation} \\
&\qquad\qquad \text{iff} \\
&\text{there exist } \omega \in R^+, a_1, \ldots, a_{k-1} \in \Sigma \text{ and } a_k \in \Sigma_\epsilon \text{ such that} \\
&(A, \epsilon) \Rightarrow_G^+ (a_1, \epsilon) \cdots (a_{k-1}, \epsilon)(a_k, \omega)(X_n, \epsilon) \cdots (X_1, \epsilon), \\
&(q_1, \omega, c_1) \vdash_M^+ (q_2, \epsilon, c_\epsilon) \text{ and } x = a_1 \cdots a_k. && (7)
\end{aligned}$$



$$(a,\epsilon)(b,\epsilon)(d,\epsilon)(e,r_1r_2r_3r_5)(d,\epsilon)(f,r_4r_6)(a,\epsilon)$$

a) Derivation in an ELLDG controlled by an $\mathcal{S}$–automaton.

$$\begin{aligned}
((q_0, A), abdedfa, (c_\epsilon, \epsilon)) &\vdash \\
((q_1, B), bdedfa, (c_1, a)) &\vdash \\
((q_2, D_1), dedfa, (c_2, D_2a)) &\vdash \\
((q_3, E), edfa, (c_3, D_2a)) &\vdash \\
((q_5, \epsilon), dfa, (c_\epsilon, D_2a)) &\vdash \\
((q_0, D_2), dfa, (c_\epsilon, a)) &\vdash \\
((q_4, F), fa, (c_4, a)) &\vdash \\
((q_6, \epsilon), a, (c_\epsilon, a)) &\vdash \\
((q_6, \epsilon), \epsilon, (c_\epsilon, \epsilon))
\end{aligned}$$

b) Simulation by an $\mathcal{S} \circ_r \mathcal{S}_{\mathrm{pd}}$ automaton.

Fig. 2 Example of the simulation of a controlled ELLDG derivation

This assertion can be proved straightforwardly by induction on the length of the derivation and the length of the computation, respectively:

If. Suppose we have a derivation $(A,\epsilon) \overset{i}{\underset{G}{\Rightarrow}} (a_1,\epsilon)\cdots(a_{k-1},\epsilon)(a_k,\omega)(X_n,\epsilon)\cdots(X_1,\epsilon)$ and a corresponding computation $(q_1,\omega,c_1) \overset{*}{\underset{M}{\vdash}} (q_2,\epsilon,c_\epsilon)$. In case $i=1$ the applied production $\omega \in R$ cannot be of the form mentioned in (1) since we know that the object (a_k,ω) must be the distinguished child of (A,ϵ) by the definition of ELLDG. Thus the production must be one like those treated in (2) and (3). Using in addition the necessary number of transitions corresponding to the computation of M for accepting ω, introduced by (4), a computation for M' can be found, in which no pops on the second component are used.

For the induction suppose the assertion is true for some $i \in \mathbb{N}$. A derivation of length $i+1$ has the following shape:

$$\begin{aligned}
(A,\epsilon) &\underset{G}{\Rightarrow} \\
(a_1,\epsilon)\cdots(a_g,\epsilon)(B,r)(X_m,\epsilon)\cdots(X_1,\epsilon) &\overset{*}{\underset{G}{\Rightarrow}} \\
(a_1,\epsilon)\cdots(a_{k-1},\epsilon)(a_k,r\omega)(X_n,\epsilon)\cdots(X_1,\epsilon)&
\end{aligned}$$

with $0 \leq g \leq k$ and $0 \leq m \leq n$, $A, B \in N$, $a_1, \ldots, a_{k-1} \in \Sigma$, $a_k \in \Sigma_\epsilon$, $X_1, \ldots, X_n \in N \cup \Sigma$, $r \in R$ and $\omega \in R^+$. For M we have $(q_1, r\omega, c_1) \vdash_M^* (q_2, r\omega, c_2) \vdash_M (q_3, \omega, c_3) \vdash_M^* (q_4, \epsilon, c_\epsilon)$. For M' we find correspondingly:

$$\begin{aligned} &((q_1, A), a_1 \cdots a_k, (c_1, \epsilon)) \vdash_{M'}^* && \text{(by (4))} \\ &((q_2, A), a_1 \cdots a_k, (c_2, \epsilon)) \vdash_{M'} && \text{(by (1))} \\ &((q_3, B), a_{g+1} \cdots a_k, (c_3, X_m \cdots X_1)) \vdash_{M'} && \text{(by induction)} \\ &((q_4, \epsilon), \epsilon, (c_\epsilon, X_n \cdots X_1)). \end{aligned}$$

Only if. Consider a computation $((q_1, A), x, (c_1, \epsilon)) \vdash_{M'}^i ((q_2, \epsilon), \epsilon, (c_\epsilon, X_n \cdots X_1))$ with $q_2 \in Q_F$ that does not use transitions of type (5) or (6). First consider the case in which the computation starts with a transition introduced by (2) or (3) of the construction. In this case the (possibly) remaining transitions must be of type (4) with $A = \epsilon$, and it is easy to see that the implication is true. If $i = 1$ a transition introduced by (2) or (3) of the construction must have been applied and the implication thus is true.

If the implication is true for some $i \in \mathbb{N}$, for a computation of length $i + 1$ we have now only to pay attention to those cases in which the first transition applied was introduced by (1) or (4). The interesting case is, of course, the one in which (1) was used. Here, we have for M':

$$((q_1, A), vx, (c_1, \epsilon)) \vdash_{M'} ((q_2, B), x, (c_2, X_m \cdots X_1)) \vdash_{M'}^i ((q_3, \epsilon), \epsilon, (c_\epsilon, X_n \cdots X_1))$$

with $0 \leq m \leq n$, $A, B \in N$, $v, x \in \Sigma^*$, $X_1, \ldots, X_n \in N \cup \Sigma$. By (1) and by induction we can be sure that there is a derivation

$$\begin{aligned} &(A, \epsilon) \Rightarrow_G \\ &(a_1, \epsilon) \cdots (a_g, \epsilon)(B, r)(X_m, \epsilon) \cdots (X_1, \epsilon) \Rightarrow_G^i \\ &(a_1, \epsilon) \cdots (a_{k-1}, \epsilon)(a_k, r\omega)(X_n, \epsilon) \cdots (X_1, \epsilon) \end{aligned}$$

where $0 \leq g \leq k$, $a_1 \cdots a_g = v$ and $a_{g+1} \cdots a_k = x$. An associated computation in M with input $r\omega$ is easily found as well.

After we have used induction on the length of spines above, in the second part of the proof we will in some sense do an induction on the number of spines in a derivation. We show that for all $A \in N$, $x \in \Sigma^*$ and $X_1, \ldots, X_k \in N \cup \Sigma$:

$$\begin{aligned} &\text{there exist } q_1 \in Q_I \text{ and } q_2 \in Q_F \text{ such that} \\ &((q_1, A), x, (c_\epsilon, X_k \cdots X_1)) \vdash_{M'}^* ((q_2, \epsilon), \epsilon, (c_\epsilon, \epsilon)) \\ &\qquad\qquad \text{iff} \\ &\text{there exist } \omega_1, \ldots, \omega_n \in L(M)_\epsilon \text{ and } a_1, \ldots, a_n \in \Sigma_\epsilon \text{ such that} \\ &(A, \epsilon)(X_k, \epsilon) \cdots (X_1, \epsilon) \Rightarrow_G^* (a_1, \omega_1) \cdots (a_n, \omega_n) \text{ and } x = a_1 \cdots a_n. \end{aligned} \tag{8}$$

If. Consider a (leftmost) derivation of length i, assuming that the implication holds for all (leftmost) derivations of length $j < i$. Since the derivation is leftmost, it starts by rewriting (A, ϵ). We split up the derivation into two parts. The first part corresponds to a derivation of (A, ϵ) as in (7) and the second part is the remainder of the

derivation. Hence the derivation has one of the following two forms (where the first form corresponds to the case that the remainder of the derivation is empty): either

$$\begin{aligned}(A,\epsilon)(X_k,\epsilon)\cdots(X_1,\epsilon) &\overset{*}{\underset{G}{\Rightarrow}}\\ (a_1,\epsilon)\cdots(a_{m-1},\epsilon)(a_m,\omega)(a_{m+1},\epsilon)\cdots(a_p,\epsilon)(X_k,\epsilon)\cdots(X_1,\epsilon) &= \\ (a_1,\omega_1)\cdots(a_n,\omega_n)&\end{aligned}$$

or

$$\begin{aligned}(A,\epsilon)(X_k,\epsilon)\cdots(X_1,\epsilon) &\overset{*}{\underset{G}{\Rightarrow}}\\ (a_1,\epsilon)\cdots(a_{m-1},\epsilon)(a_m,\omega)(a_{m+1},\epsilon)\cdots(a_p,\epsilon)(B,\epsilon)(X_l,\epsilon)\cdots(X_1,\epsilon) &\overset{*}{\underset{G}{\Rightarrow}}\\ (a_1,\omega_1)\cdots(a_n,\omega_n)&\end{aligned}$$

with $m \le p \le n$, $A, B \in N$, $X_1, \ldots, X_k \in N \cup \Sigma$, $\omega \in R^+$ and (for the second form) either $l < k$ and $B = X_{l+1}$ or $l \ge k$. Since $\omega \in L(M)$, there exist $q_1 \in Q_I$ and $q_2 \in Q_F$ such that $(q_1, \omega, c_\epsilon) \underset{M}{\vdash}^* (q_2, \epsilon, c_\epsilon)$.

For the first form of derivation (in which $X_k \cdots X_1 = a_{p+1} \cdots a_n$) we find for M':

$$\begin{aligned}((q_1, A), a_1 \cdots a_n, (c_\epsilon, X_k \cdots X_1)) &\underset{M'}{\vdash}^* && \text{(by (7))}\\ ((q_2, \epsilon), a_{m+1} \cdots a_n, (c_\epsilon, a_{m+1} \cdots a_n)) &\underset{M'}{\vdash}^* && \text{(by (6))}\\ ((q_2, \epsilon), \epsilon, (c_\epsilon, \epsilon)).&&&\end{aligned}$$

Note that the transitions from (6) are applicable because the configuration of the first component is c_ϵ.

For the second form of derivation we find for M':

$$\begin{aligned}((q_1, A), a_1 \cdots a_n, (c_\epsilon, X_k \cdots X_1)) &\underset{M'}{\vdash}^* && \text{(by (7))}\\ ((q_2, \epsilon), a_{m+1} \cdots a_n, (c_\epsilon, a_{m+1} \cdots a_p B X_l \cdots X_1)) &\underset{M'}{\vdash}^* && \text{(by (6))}\\ ((q_2, \epsilon), a_{p+1} \cdots a_n, (c_\epsilon, B X_l \cdots X_1)) &\underset{M'}{\vdash} && \text{(by (5))}\\ ((q_3, B), a_{p+1} \cdots a_n, (c_\epsilon, X_l \cdots X_1)) &\underset{M'}{\vdash}^* && \text{(by induction)}\\ ((q_4, \epsilon), \epsilon, (c_\epsilon, \epsilon))&&&\end{aligned}$$

with $q_3 \in Q_I$ and $q_4 \in Q_F$.

Only if. Suppose the implication is true for each $j < i$, and consider a computation of M' of length i. We split up the computation into three parts. In the first part M' does not pop from the second component, i.e., does not use transitions of type (5) or (6). In the second part it uses transitions of type (6) only. The third part is the remainder of the computation, starting with a transition of type (5). Note that at the end of the first part the first component of the configuration must be c_ϵ, because otherwise the pops are not defined; for the same reason the state must be (q_3, ϵ) for some $q_3 \in Q$. Hence, the computation has one of the following two forms (where the first corresponds to the case that the third part of the computation is empty): either

$$\begin{aligned}((q_1, A), x, (c_\epsilon, X_k \cdots X_1)) &\underset{M'}{\vdash}^* && \text{(using (1–4))}\\ ((q_3, \epsilon), y, (c_\epsilon, u X_k \cdots X_1)) &\underset{M'}{\vdash}^* && \text{(using (6))}\\ ((q_2, \epsilon), \epsilon, (c_\epsilon, \epsilon))&&&\end{aligned}$$

where $y = uX_k \cdots X_1$ is a postfix of x and $q_3 = q_2$, or

$$\begin{aligned} ((q_1, A), x, (c_\epsilon, X_k \cdots X_1)) &\vdash_{M'}^* && \text{(using (1–4))} \\ ((q_3, \epsilon), y, (c_\epsilon, uBX_l \cdots X_1)) &\vdash_{M'}^* && \text{(using (6))} \\ ((q_3, \epsilon), v, (c_\epsilon, BX_l \cdots X_1)) &\vdash_{M'} && \text{(using (5))} \\ ((q_4, B), v, (c_\epsilon, X_l \cdots X_1)) &\vdash_{M'}^* \\ ((q_2, \epsilon), \epsilon, (c_\epsilon, \epsilon)) \end{aligned}$$

where $y = uv$ is a postfix of x, $q_1 \in Q_I$, $q_2 \in Q_F$, $B \in N$, $X_i \in N \cup \Sigma$ and (for the second form) either $l < k$ and $B = X_{l+1}$ or $l \geq k$; moreover, by (5), $q_3 \in Q_F$ and $q_4 \in Q_I$.

The corresponding derivations of G are:

$$\begin{aligned} (A, \epsilon)(X_k, \epsilon) \cdots (X_1, \epsilon) &\overset{*}{\underset{G}{\Rightarrow}} && \text{(by (7))} \\ (a_1, \epsilon) \cdots (a_{m-1}, \epsilon)(a_m, \omega)(a_{m+1}, \epsilon) \cdots (a_p, \epsilon)(X_k, \epsilon) \cdots (X_1, \epsilon) &= \\ (a_1, \omega_1) \cdots (a_n, \omega_n) \end{aligned}$$

with $x = a_1 \cdots a_n = a_1 \cdots a_m y$ and $u = a_{m+1} \cdots a_p$, and

$$\begin{aligned} (A, \epsilon)(X_k, \epsilon) \cdots (X_1, \epsilon) &\overset{*}{\underset{G}{\Rightarrow}} && \text{(by (7))} \\ (a_1, \epsilon) \cdots (a_{m-1}, \epsilon)(a_m, \omega)(a_{m+1}, \epsilon) \cdots & \\ (a_p, \epsilon)(B, \epsilon)(X_l, \epsilon) \cdots (X_1, \epsilon) &\overset{*}{\underset{G}{\Rightarrow}} && \text{(by induction)} \\ (a_1, \omega_1) \cdots (a_n, \omega_n) \end{aligned}$$

with $x = a_1 \cdots a_n$, $u = a_{m+1} \cdots a_p$ and $v = a_{p+1} \cdots a_n$, respectively. By (7) we moreover know that $(q_1, \omega, c_\epsilon) \vdash_M^+ (q_3, \epsilon, c_\epsilon)$. Since $q_1 \in Q_I$ and $q_3 \in Q_F$ we see that $\omega \in L(M)$.

Taking $A = A_{\text{in}}$ and $k = 0$ in assertion (8) we finally see that $L(M') = L(K)$. □

Before we proceed showing that the inclusion between controlled ELLDGs and automata with concatenated storages also holds in the other direction, we first define a kind of normal form for $\mathcal{S} \circ_r \mathcal{S}_{\text{pd}}$-automata.

Proposition 10 *For a storage S and a finite alphabet Γ, let M be an $S \circ_r S_{\text{pd}}(\Gamma)$-automaton. Then there exists an automaton $M' = (Q, \Sigma, S \circ_r S_{\text{pd}}(\Gamma \cup \{\#\}), \delta, \{q_0\}, \{q_f\})$ such that $L(M') = L(M)$ and* (i) *each computation of M' starts with pushing # onto the stack, ends with popping # and no intermediate transition pops or pushes #, where #, called the bottom of stack symbol, is a new symbol not in Γ and* (ii) *M reads at most one symbol in each derivation step* (*i.e., $\delta \subseteq Q \times \Sigma_\epsilon \times Q \times F$, where F denotes the set of function symbols of the composite storage*).

Proof First we show that given a storage $S = (C, c_\epsilon, F, id, m)$ and an S-automaton $M = (Q, \Sigma, S, \delta, Q_I, Q_F)$ we can construct an equivalent S-automaton $M' = (Q', \Sigma, S, \delta', Q_I, Q_F)$ with $\delta' \subseteq Q \times \Sigma_\epsilon \times Q \times F$. To do so, we set $Q' = Q \cup \{q_{w,p} \mid (q, vw, p, f) \in \delta \text{ with } v, w \in \Sigma^*\}$ and let

$$\delta' = \{(q, a, p, f) \mid (q, a, p, f) \in \delta \text{ and } a \in \Sigma_\epsilon\}$$

$$\cup\{(q, a, q_{v,p}, f) \mid (q, av, p, f) \in \delta \text{ and } a \in \Sigma, v \in \Sigma^+\}$$
$$\cup\{(q_{av,p}, a, q_{v,p}, id) \mid a \in \Sigma, v \in \Sigma^+\}$$
$$\cup\{(q_{a,p}, a, p, id) \mid a \in \Sigma\}.$$

Now let $M = (Q, \Sigma, S \circ_r S_{\mathrm{pd}}(\Gamma), \delta, Q_I, Q_F)$ be an automaton for a finite alphabet Γ and a storage S. We construct an automaton $M' = (Q \cup \{q_0, q_f\}, \Sigma, S \circ_r S_{\mathrm{pd}}(\Gamma \cup \{\#\}), \delta', \{q_0\}, \{q_f\})$, where q_0 and q_f are new, distinct, symbols not in Q and # a new symbol not in Γ, and where $\delta' = \delta \cup \{(q_0, \epsilon, q, \triangleright\mathsf{push}(\#)) \mid q \in Q_I\} \cup \{(q, \epsilon, q_f, \triangleright\mathsf{pop}(\#)) \mid q \in Q_F\}$. It is obvious that $L(M') = L(M)$, each computation of M' starts with pushing # and ends with popping that symbol. □

Given an $S \circ_r S_{\mathrm{pd}}(\Gamma)$-automaton M, the idea for constructing an equivalent ELLDG G controlled by (a language accepted by) an S-automaton M' is again straightforward, similar to the usual "triple construction" for simulating a pushdown automaton by a context-free grammar. The grammar G is used to code the configurations of the pushdown component and M' follows the configurations of the first component. The computation of a control word and hence the simulation of the first component of the $S \circ_r S_{\mathrm{pd}}(\Gamma)$ storage has to stop as soon as the foot of a spine in the ELLDG is reached. This is exactly the point at which M will pop an element from the second component and has to be in a semi-empty configuration, so to speak accepting the control word. The subsequent configurations of the first component are simulated along a new spine by a new computation of the S-automaton, starting with c_ϵ. Thus the final configuration that was reached by the first component has to be c_ϵ which is the case because recognition is by final state and empty configuration.

Lemma 2 *For each class of storages $\mathcal{S}$,*

$$L(\mathcal{S} \circ_r \mathcal{S}_{\mathrm{pd}}) \subseteq L(\mathcal{G}_{\mathrm{ELLD}}/L(\mathcal{S})).$$

Proof Let $\mathcal{S}$ be a class of storages, let $M = (Q, \Sigma, S \circ_r S_{\mathrm{pd}}(\Gamma), \delta, Q_I, Q_F)$ be an automaton for a finite alphabet Γ and $S = (C, c_\epsilon, F, id, m) \in \mathcal{S}$. Assume w.l.o.g. that M is in the form as described in Proposition 10. Thus $Q_I = \{q_0\}$ and $Q_F = \{q_z\}$ for some $q_0, q_z \in Q$, there is a symbol $\# \in \Gamma$ that serves as the bottom of stack symbol and M reads at most one input symbol at each transition.

Construct an ELLDG $G = (N, \Sigma, R, A_{\mathrm{in}})$ where A_{in} is a new symbol and $N = (Q \times \Gamma \times Q) \cup \{A_{\mathrm{in}}\}$ and construct an automaton $M' = (\{q_{\mathrm{in}}, q_{\mathrm{fn}}\}, R, S, \delta', \{q_{\mathrm{in}}\}, \{q_{\mathrm{fn}}\})$ where $q_{\mathrm{in}}, q_{\mathrm{fn}}$ are new, distinct, symbols and where R and δ' are determined by the following:

$$\begin{aligned}&\text{If } (q_1, a, q_2, \triangleleft f) \in \delta \quad \text{then,}\\ &\quad \text{for each } q \in Q \text{ and } A \in \Gamma,\\ &\quad r = (q_1, A, q) \to a(q_2, A, q) \in R \quad \text{and}\\ &\quad (q_{\mathrm{in}}, r, q_{\mathrm{in}}, f) \in \delta'. \qquad (1)\end{aligned}$$

If $(q_1, a, q_2, \triangleright\mathsf{id}) \in \delta$ then,
for each $q \in Q$ and $A \in \Gamma$,
$r = (q_1, A, q) \to a(q_2, A, q) \in R$ and
$(q_{\text{in}}, r, q_{\text{in}}, id) \in \delta'$. (2)

If $(q_1, a, q_2, \triangleright\mathsf{push}(B)) \in \delta$ with $B \neq \#$ then,
for each $q_3, q_4 \in Q$ and $A \in \Gamma$,
$r = (q_1, A, q_4) \to a(q_2, B, q_3)(q_3, A, q_4) \in R$ and
$(q_{\text{in}}, r, q_{\text{in}}, id) \in \delta'$. (3)

If $(q_0, \epsilon, q_1, \triangleright\mathsf{push}(\#)) \in \delta$ then
$r = A_{\text{in}} \to (q_1, \#, q_z) \in R$ and
$(q_{\text{in}}, r, q_{\text{in}}, id) \in \delta'$. (4)

If $(q_1, a, q_2, \triangleright\mathsf{pop}(A)) \in \delta$ then
$r = (q_1, A, q_2) \to a \in R$ and
$(q_{\text{in}}, r, q_{\text{fn}}, id) \in \delta'$. (5)

An example for the construction is given in Fig. 3. In the following, for the rules introduced by (1–3), we need to distinguish the case in which the symbol a mentioned in the construction is a terminal symbol and the case $a = \epsilon$. For convenience we identify (ϵ, ϵ) (which cannot be derived by a linear controlled grammar) with ϵ and subsume the latter case under the former.

To establish that $L(G, L(M')) = L(M)$ we will show that for all $p, q \in Q$, $x \in \Sigma^*$, $c_1 \in C$ and $A \in \Gamma$, the following two statements are equivalent:

(i) there exist $a_1, \ldots, a_n \in \Sigma_\epsilon$, $k \in \{1, \ldots, n\}$, $\omega_k \in R^+$ and $\omega_{k+1}, \ldots, \omega_n \in L(M')_\epsilon$ such that $(q_{\text{in}}, \omega_k, c_1) \vdash_{M'}^* (q_{\text{fn}}, \epsilon, c_\epsilon)$, $x = a_1 \cdots a_n$ and

$$((p, A, q), \epsilon) \Rightarrow_G^* (a_1, \epsilon) \cdots (a_{k-1}, \epsilon)(a_k, \omega_k)(a_{k+1}, \omega_{k+1}) \cdots (a_n, \omega_n).$$

(ii) $(p, x, (c_1, A)) \vdash_M^* (q, \epsilon, (c_\epsilon, \epsilon))$, no intermediate configuration in this computation equals (c_ϵ, ϵ) and no transition uses $\triangleright\mathsf{push}(\#)$.

If. The first part of the proof can be done by induction on the number of steps in a computation, i, starting with $i = 1$. Suppose $(p, x, (c_1, A)) \vdash_M (q, \epsilon, (c_\epsilon, \epsilon))$. In this case $x \in \Sigma_\epsilon$ and $c_1 = c_\epsilon$. By rule (5) of the construction we find $((p, A, q), \epsilon) \Rightarrow_G (x, r)$ and $(q_{\text{in}}, r, c_\epsilon) \vdash (q_{\text{fn}}, \epsilon, c_\epsilon)$.

Suppose that the assertion is true for each j, $1 \leq j \leq i$, for some $i \in \mathbb{N}$. Consider a computation of length $i + 1$.

If the first transition that is applied is of the form $(p, a_1, p_1, \triangleleft f)$ the computation has the following shape:

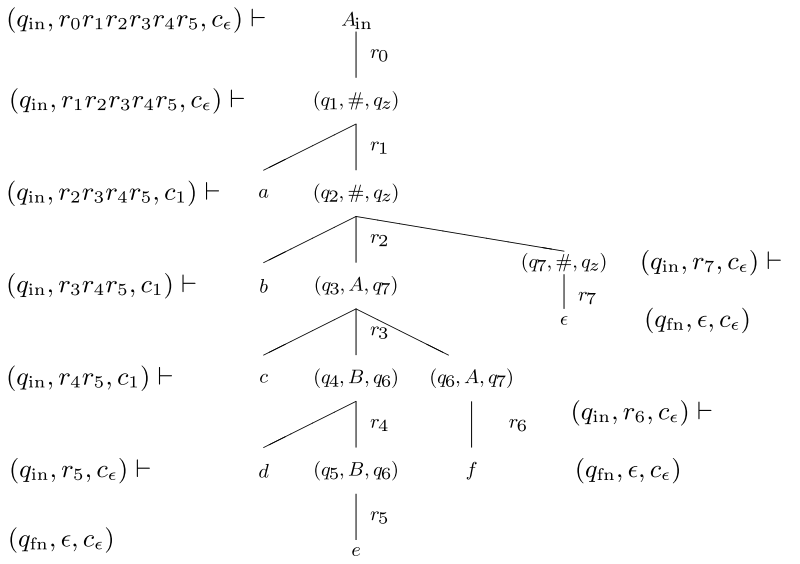
$$(p, x, (c_1, A)) \vdash_M (p_1, y, (m(f)(c_1), A)) \vdash_M^i (q, \epsilon, (c_\epsilon, \epsilon))$$

for some $y \in \Sigma^*$ with $x = a_1 y$. Using a production r in G corresponding to the first transition we have

$$\begin{aligned} ((p, A, q), \epsilon) &\Rightarrow_G && \text{(by (1))} \\ (a_1, \epsilon)((p_1, A, q), r) &\Rightarrow_G^* && \text{(by induction)} \\ (a_1, \epsilon)(a_2, \epsilon) \cdots (a_{k-1}, \epsilon)(a_k, r\omega_k) \cdots (a_n, \omega_n) & \end{aligned}$$

$$
\begin{aligned}
(q_0, abcdef, (c_\epsilon, \epsilon)) &\vdash \\
(q_1, abcdef, (c_\epsilon, \#)) &\vdash \\
(q_2, bcdef, (c_1, \#)) &\vdash \\
(q_3, cdef, (c_1, A\#)) &\vdash \\
(q_4, def, (c_1, BA\#)) &\vdash \\
(q_5, ef, (c_\epsilon, BA\#)) &\vdash \\
(q_6, f, (c_\epsilon, A\#)) &\vdash \\
(q_7, \epsilon, (c_\epsilon, \#)) &\vdash \\
(q_z, \epsilon, (c_\epsilon, \epsilon))
\end{aligned}
$$

a) Computation of an $\mathcal{S} \circ_r \mathcal{S}_{\mathrm{pd}}$–automaton



b) Simulation by an ELLDG and a controlling $\mathcal{S}$–automaton

Fig. 3 Example of the simulation of an $\mathcal{S} \circ_r \mathcal{S}_{\mathrm{pd}}$-automaton

with $a_2 \cdots a_n = y$. For M' we find again by (1) and by induction:

$$(q_{\mathrm{in}}, r\omega_k, c_1) \vdash_{M'} (q_{\mathrm{in}}, \omega_k, m(f)(c_1)) \vdash_{M'}^{*} (q_{\mathrm{fn}}, \epsilon, c_\epsilon).$$

By induction we can conclude that $\omega_{k+1}, \ldots, \omega_n \in L(M')_\epsilon$.

If the first transition is of the form $(p, a, p_1, \triangleright\mathsf{id})$ the situation is almost the same as in the case above.

Otherwise, if the first transition is $(p, a_1, p_1, \triangleright\mathsf{push}(B))$ with $B \neq \#$, we have

$$(p, x, (c_1, A)) \vdash_M (p_1, yz, (c_1, BA)) \vdash_M^{i-j} (p_2, z, (c_\epsilon, A)) \vdash_M^{j} (q, \epsilon, (c_\epsilon, \epsilon))$$

for some j such that $1 \leq j < i$, some $y, z \in \Sigma^*$ with $x = a_1 yz$ and some $p_2 \in Q$. By (3) and the induction hypothesis we may conclude that there is a production rule r

such that

$$\begin{aligned}((p,A,q),\epsilon) &\overrightarrow{\Rightarrow}_G && \text{(by (3))}\\ (a_1,\epsilon)((p_1,B,p_2),r)((p_2,A,q),\epsilon) &\overrightarrow{\Rightarrow}_G^* && \text{(by induction)}\\ (a_1,\epsilon)\cdots(a_{k-1},\epsilon)(a_k,r\omega_k)\cdots(a_l,\omega_l)((p_2,A,q),\epsilon) &\overrightarrow{\Rightarrow}_G^* && \text{(by induction)}\\ (a_1,\epsilon)\cdots(a_{k-1},\epsilon)(a_k,r\omega_k)\cdots(a_l,\omega_l)\cdots(a_n,\omega_n)\end{aligned}$$

with $2 \le k \le l \le n$, $y = a_2 \cdots a_l$ and $z = a_{l+1} \cdots a_n$ (and so $x = a_1 \cdots a_n$). For M' we find

$$(q_{\text{in}}, r\omega_k, c_1) \vdash_{M'} (q_{\text{in}}, \omega_k, c_1) \vdash_{M'}^* (q_{\text{fn}}, \epsilon, c_\epsilon).$$

Furthermore we find (by induction) that $\omega_i \in L(M')_\epsilon$ for each $k < i \le n$.

Finally, note that the first transition can neither be of type (4), which is excluded by the assertion, nor of type (5). In the latter case the computation has length 1 since no intermediate configuration is the empty one.

Only if. Suppose that $((p,A,q),\epsilon) \overrightarrow{\Rightarrow}_G^i v(a,\omega)w$ with $v \in (\Sigma \times \{\epsilon\})^*$, $a \in \Sigma_\epsilon$, $\omega \in R^+$, $w \in (\Sigma \times L(M')_\epsilon)^*$ and $(q_{\text{in}}, \omega, c_1) \vdash_{M'}^* (q_{\text{fn}}, \epsilon, c_\epsilon)$. If $i = 1$, the single applied production has to be one introduced by (5). Consequently, we know that $c_1 = c_\epsilon$, $\omega = r$, and $v = w = \epsilon$. Thus we have $(p, a, (c_\epsilon, A)) \vdash_M (q, \epsilon, (c_\epsilon, \epsilon))$ what was to be shown.

Suppose that for some $i \ge 1$ the assertion is true for each $1 \le j \le i$. Consider a derivation of length $i+1$. Since $i + 1 \ge 2$, the first rule applied is of type (1), (2) or (3). Suppose the first rule applied is of type (1). Then we have the following derivation:

$$\begin{aligned}((p,A,q),\epsilon) \overrightarrow{\Rightarrow}_G (a_1,\epsilon)((p_1,A,q),r) \overrightarrow{\Rightarrow}_G^i (a_1,\epsilon)\cdots(a_{k-1},\epsilon)(a_k,r\omega_k)\cdots(a_n,\omega_n)\\ \text{and}\quad (q_{\text{in}}, r\omega_k, c_1) \vdash_{M'} (q_{\text{in}}, \omega_k, m(f)(c_1)) \vdash_{M'}^* (q_{\text{fn}}, \epsilon, c_\epsilon)\end{aligned}$$

with $2 \le k \le n$, $a_1, \ldots, a_n \in \Sigma_\epsilon$, $p_1 \in Q$, $r \in R$ and $\omega_k \in R^*$. Using the transition which has allowed the introduction of the production rule of type (1) and the induction hypothesis we may conclude now:

$$(p, a_1a_2\cdots a_n, (c_1, A)) \vdash_M (p_1, a_2\cdots a_n, (m(f)(c_1), A)) \vdash_M^* (q, \epsilon, (c_\epsilon, \epsilon)).$$

In case the first applied rule was introduced by (2) the argumentation is almost the same and left to the reader. If otherwise a rule of type (3) is used we have a derivation of the following shape:

$$\begin{aligned}((p,A,q),\epsilon) &\overrightarrow{\Rightarrow}_G\\ (a_1,\epsilon)((p_1,B,q_1),r)((q_1,A,q),\epsilon) &\overrightarrow{\Rightarrow}_G^{i-j}\\ (a_1,\epsilon)\cdots(a_{k-1},\epsilon)(a_k,r\omega_k)\cdots(a_l,\omega_l)((q_1,A,q),\epsilon) &\overrightarrow{\Rightarrow}_G^{j}\\ (a_1,\epsilon)\cdots(a_{k-1},\epsilon)(a_k,r\omega_k)\cdots(a_l,\omega_l)\cdots(a_m,\omega_m)\cdots(a_n,\omega_n)&\\ \text{and}\quad (q_{\text{in}}, r\omega_k, c_1) \vdash_{M'} (q_{\text{in}}, \omega_k, c_1) \vdash_{M'}^* (q_{\text{fn}}, \epsilon, c_\epsilon)&\end{aligned}$$

for some $1 \leq j < i$ and with $2 \leq k \leq l \leq n$, $a_1, \ldots, a_n \in \Sigma_\epsilon$, $B \in \Gamma$, $p_1, q_1 \in Q$, $r \in R$ and $\omega_k \in R^*$. Now we may conclude

$$\begin{aligned}
&(p, a_1 a_2 \cdots a_n, (c_1, A)) \vdash_M && \text{(by the construction)}\\
&(p_1, a_2 \cdots a_n, (c_1, BA)) \vdash_M^* && \text{(by induction)}\\
&(q_1, a_{l+1} \cdots a_n, (c_\epsilon, A)) \vdash_M^* && \text{(by induction)}\\
&(q, \epsilon, (c_\epsilon, \epsilon)).
\end{aligned}$$

Note that the induction hypothesis can be applied to the derivation starting with $((q_1, A, q), \epsilon)$ because $\omega_m \in L(M')$ and thus $(q_{\text{in}}, \omega_m, c_\epsilon) \vdash_{M'}^* (q_{\text{fn}}, \epsilon, c_\epsilon)$.

To complete the proof we recall that M was assumed to be in the normal form of Proposition 10. Thus a computation of M has to start like $(q_0, x, (c_\epsilon, \epsilon)) \vdash_M (q_1, x, (c_\epsilon, \#))$ for some $q_1 \in Q$, no intermediate configuration can be the empty one and no intermediate transition uses $\triangleright\mathsf{push}(\#)$. By (4) of the construction we find the beginning of a derivation in G starting with the start symbol of G: $A_{\text{in}} \overset{*}{\underset{G}{\Rightarrow}} (q_1, \#, q_z)$. The implication holds the other way around as well. The correspondence of the remaining derivation and computation is established by the assertion just shown, that is applicable because of the normal form properties of M. Thus $(G, L(M'))$ generates exactly the strings that M accepts. □

From the last two lemmata we can conclude immediately the following theorem.

Theorem 1 *For each class of storages* $\mathcal{S}$,

$$L(\mathcal{S} \circ_r \mathcal{S}_{\text{pd}}) = L(\mathcal{G}_{\text{ELLD}}/L(\mathcal{S})).$$

Using the relation between concatenation w.r.t. reading and writing, the result of Theorem 1 can be translated directly to obtain a characterization of concatenation w.r.t. writing in terms of controlled grammars. A sketch of a direct proof is given in [26].

Theorem 2 *For each class of invertible storages* $\mathcal{S}$,

$$L(\mathcal{S} \circ_w \mathcal{S}_{\text{pd}}) = L(\mathcal{G}_{\text{ERLD}}/L(\mathcal{S})^R).$$

Proof

$$\begin{aligned}
L(\mathcal{S} \circ_w \mathcal{S}_{\text{pd}}) &= L((\mathcal{S}^{\text{inv}} \circ_r \mathcal{S}_{\text{pd}})^{\text{inv}}) && \text{(by Propositions 4 and 1)}\\
&= L(\mathcal{S}^{\text{inv}} \circ_r \mathcal{S}_{\text{pd}})^R && \text{(by Proposition 5)}\\
&= L(\mathcal{G}_{\text{ELLD}}/L(\mathcal{S}^{\text{inv}}))^R && \text{(by Theorem 1)}\\
&= L(\mathcal{G}_{\text{ERLD}}/L(\mathcal{S}^{\text{inv}})) && \text{(by Proposition 6)}\\
&= L(\mathcal{G}_{\text{ERLD}}/L(\mathcal{S})^R) && \text{(by Proposition 5).}
\end{aligned}$$

□

The main result of the paper, the relation between control and composition of storages, is captured by the two theorems above. In the following final section we will try to place this result in a somewhat broader context and consider a number of related results from the literature.

5 Some Hierarchies of Controlled Languages

In [1] a hierarchy of languages accepted by multi-pushdown automata is established. By Theorem 1 this hierarchy can be defined in terms of controlled ELLDGs. This representation lends itself to a comparison with other hierarchies defined by iterated control. In the following, let $\mathcal{L}_{\mathrm{REG}}$ denote the class of regular and let $\mathcal{L}_{\mathrm{CF}}$ be the class of context-free languages.

5.1 The Hierarchy of Breveglieri et al.

In the terminology developed above the classes constituting the hierarchy of [1] can be reconstructed[10] as $\mathcal{C}_i^{\mathcal{R}} = L(\mathcal{S}_i)$ for each $i \geq 0$ where $\mathcal{S}_0 = \mathcal{S}_{\mathrm{triv}}$ and $\mathcal{S}_i = \mathcal{S}_{i-1} \circ_r \mathcal{S}_{\mathrm{pd}}$. ($\mathcal{C}$ is intended as a mnemonic for concatenation, the superscript $\mathcal{R}$ indicating that concatenation w.r.t. reading is meant.) By Theorem 1 and by the fact that $L(\mathcal{S}_{\mathrm{triv}}) = \mathcal{L}_{\mathrm{REG}}$, these classes of languages can be defined as well by setting $\mathcal{C}_0^{\mathcal{R}} = \mathcal{L}_{\mathrm{REG}}$ and $\mathcal{C}_i^{\mathcal{R}} = L(\mathcal{G}_{\mathrm{ELLD}}/\mathcal{C}_{i-1}^{\mathcal{R}})$.

Similarly we can define a "reverse" hierarchy based on concatenation w.r.t. writing or control of ERLDGs: $\mathcal{C}_i^{\mathcal{W}} = L(\mathcal{S}_i)$ for each $i \geq 0$ where $\mathcal{S}_0 = \mathcal{S}_{\mathrm{triv}}$ and $\mathcal{S}_i = \mathcal{S}_{i-1} \circ_w \mathcal{S}_{\mathrm{pd}}$. Theorem 2 gives a grammatical characterization of the same classes: $\mathcal{C}_0^{\mathcal{W}} = \mathcal{L}_{\mathrm{REG}}$ and $\mathcal{C}_i^{\mathcal{W}} = L(\mathcal{G}_{\mathrm{ERLD}}/(\mathcal{C}_{i-1}^{\mathcal{W}})^R)$. As an immediate consequence of Proposition 12 below (and the fact that the authors in [1] show that their hierarchy is a proper and infinite one), we find that the classes of this hierarchy form a proper infinite hierarchy.

Proposition 11 *$\mathcal{C}_i^{\mathcal{W}}$ and $\mathcal{C}_i^{\mathcal{R}}$ with $i \geq 0$ are closed under homomorphism, union and left and right concatenation with a symbol.*

Proof If the hierarchies are defined by control, closure under homomorphism follows directly from Proposition 9. Closure under union and concatenation with a symbol follow from the same proposition by induction, starting with the fact that $\mathcal{L}_{\mathrm{REG}}$ has the required closure properties. For the classes $\mathcal{C}_i^{\mathcal{R}}$ the proposition corresponds to statements 6, 9 and 11 of [1]. For the other classes the properties also follow by these statements and Proposition 12 below. □

Proposition 12 $(\mathcal{C}_i^{\mathcal{W}})^R = \mathcal{C}_i^{\mathcal{R}}$.

Proof In case $i = 0$ the assertion is true by the fact that $\mathcal{L}_{\mathrm{REG}}$ is closed under reversal. If the assertion is true for $i \in \mathbb{N}$, then it holds for $i + 1$ as well, since

$$\begin{aligned}
(\mathcal{C}_{i+1}^{\mathcal{W}})^R &= (L(\mathcal{G}_{\mathrm{ERLD}}/(\mathcal{C}_i^{\mathcal{W}})^R))^R && \text{(by Theorem 2)}\\
&= L(\mathcal{G}_{\mathrm{ELLD}}/(\mathcal{C}_i^{\mathcal{W}})^R) && \text{(by Proposition 6)}\\
&= L(\mathcal{G}_{\mathrm{ELLD}}/\mathcal{C}_i^{\mathcal{R}}) && \text{(by induction)}\\
&= \mathcal{C}_{i+1}^{\mathcal{R}} && \text{(by Theorem 1).}
\end{aligned}$$

□

[10] We assume that the equivalence of both definitions is obvious to the attentive reader and we will not give a proof of equivalence.

Proposition 13 *For $i > 1$, $\mathcal{C}_i^{\mathcal{W}} \neq \mathcal{C}_i^{\mathcal{R}}$.*

Proof In the proof of Theorem 2.2 of [6] it is shown that the language $L_i = \{a_1^n b_1^m b_2^m \cdots b_{2^i}^m a_2^n \cdots a_{2^i}^n \mid m, n \geq 0\}$ is not in $\mathcal{C}_i^{\mathcal{R}}$. This language is, however, generated by the controlled ERLDG $K_i = (G_i, H_i)$ where $G_i = (\{A_1, A_2, \ldots, A_{2^{i-1}}, B_1, B_2, \ldots, B_{2^{i-1}}\}, \{a_1, a_2, \ldots, a_{2^i}, b_1, b_2, \ldots, b_{2^i}\}, R, A_1)$ and

$$\begin{aligned} R = \{ & r_1 = A_1 \to a_1 A_1 a_{2^i}, \\ & r_1' = A_1 \to B_1 A_2, \\ & r_2 = A_2 \to a_2 A_2 a_{2^i - 1}, \\ & r_2' = A_2 \to A_3, \\ & \qquad \vdots \\ & r_{2^{i-1}} = A_{2^{i-1}} \to a_{2^{i-1}} A_{2^{i-1}} a_{2^{i-1}+1}, \\ & r_{2^{i-1}}' = A_{2^{i-1}} \to \epsilon, \\ & s_1 = B_1 \to b_1 B_1 b_{2^i}, \\ & s_1' = B_1 \to B_2, \\ & \qquad \vdots \\ & s_{2^{i-1}} = B_{2^{i-1}} \to b_{2^{i-1}} B_{2^{i-1}} b_{2^{i-1}+1}, \\ & s_{2^{i-1}}' = B_{2^{i-1}} \to \epsilon \} \end{aligned}$$

and

$$\begin{aligned} H = & \{r_1^n r_1' r_2^n r_2' \ldots r_{2^{i-1}}^n r_{2^{i-1}}' \mid n \in \mathbb{N}\} \\ & \cup \{s_1^n s_1' s_2^n s_2' \ldots s_{2^{i-1}}^n s_{2^{i-1}}' \mid n \in \mathbb{N}\}. \end{aligned}$$

It is left to the reader to verify that $L(K_i) = L_i$. Obviously, $H_i \in \mathcal{C}_{i-1}^{\mathcal{R}}$. Thus by the definition of the reverse hierarchy and by Proposition 12 we find $L_i \in \mathcal{C}_i^{\mathcal{W}}$. □

In other words, the classes of languages under consideration are not closed under reversal for $i > 1$. Note that $\mathcal{C}_1^{\mathcal{R}} = \mathcal{C}_1^{\mathcal{W}} = \mathcal{L}_{\mathrm{CF}}$. However, the super-families $\bigcup_{i=0}^{\infty} \mathcal{C}_i^{\mathcal{R}}$ and $\bigcup_{i=0}^{\infty} \mathcal{C}_i^{\mathcal{W}}$ are identical.

Proposition 14

(a) $\mathcal{C}_i^{\mathcal{R}} \subset \mathcal{C}_{i+1}^{\mathcal{W}}$,

(b) $\mathcal{C}_i^{\mathcal{W}} \subset \mathcal{C}_{i+1}^{\mathcal{R}}$.

Proof For case (a) we know by Theorem 2 and Proposition 12 that $\mathcal{C}_{i+1}^{\mathcal{W}} = L(\mathcal{G}_{\mathrm{ERLD}}/\mathcal{C}_i^{\mathcal{R}})$. By Propositions 7 and 11 we conclude that $\mathcal{C}_i^{\mathcal{R}} \subseteq \mathcal{C}_{i+1}^{\mathcal{W}}$, which inclusion follows also immediately from Lemma 4.6 of [6]. It is easy to see that $\mathcal{C}_i^{\mathcal{R}} = \mathcal{C}_{i+1}^{\mathcal{W}}$ leads to a contradiction: by Proposition 12 we would get $\mathcal{C}_i^{\mathcal{R}} = (\mathcal{C}_{i+1}^{\mathcal{R}})^R = (\mathcal{C}_{i+2}^{\mathcal{W}})^R = \mathcal{C}_{i+2}^{\mathcal{R}}$. Thus the inclusion is proper.

The assertion of (b) follows immediately from (a) and Proposition 12. □

Corollary 1 $\bigcup_{i=0}^{\infty} \mathcal{C}_i^{\mathcal{R}} = \bigcup_{i=0}^{\infty} \mathcal{C}_i^{\mathcal{W}}$.

5.2 The Hierarchies of Weir and Khabbaz

Given the representation of the classes of multi-pushdown languages as controlled grammars, the inclusion in the classes of Weir's hierarchy of LCGs follows immediately. The hierarchy of Weir can in turn be embedded in the multi-pushdown hierarchies as well. The multi-pushdown hierarchy however 'grows' slower than the other one. The results presented in this section were also found by [4–6]. However, Lemma 3 below states an explicit and more general relation between linear control of grammars and automata using concatenated pushdown storages. Thus this lemma closely relates to Theorems 1 and 2.

The classes of languages constituting Weir's hierarchy can be defined by setting $\mathcal{W}_0 = \mathcal{L}_{\mathrm{REG}}$ and $\mathcal{W}_i = L(\mathcal{G}_{\mathrm{LD}}/\mathcal{W}_{i-1})$ for each $i > 0$. By the definitions of the language classes and the fact that $\mathcal{W}_i$ is closed under reversal (cf. Proposition 6) but $\mathcal{C}_i^{\mathcal{W}}$ and $\mathcal{C}_i^{\mathcal{R}}$ not (cf. Proposition 13), we find immediately:

Corollary 2 *For* $i > 1$:

(a) $\mathcal{C}_i^{\mathcal{W}} \subset \mathcal{W}_i$

(b) $\mathcal{C}_i^{\mathcal{R}} \subset \mathcal{W}_i$.

This corollary is a stricter version of Theorem 4.2 of [5].

The inclusion of the multiple-pushdown languages in the classes from Weir's hierarchy can intuitively be understood in another way considering the storages defined by [28]. Take the automata accepting languages of the second class in Weir's hierarchy as an example. These automata use (linear) pushdowns of pushdowns. Among the allowed operations there is not only replacing of the topmost symbol of the topmost pushdown by a sequence of symbols, but as well replacing the entire topmost pushdown by a sequence of pushdowns including the original one. This means in fact that we can either write on the top or below the topmost pushdown. Thus in some sense these automata, too, incorporate the idea of having several possibilities for writing but only one for reading.

The languages accepted by these nested pushdowns are accepted by concatenated pushdowns as well. An LDG controlled by some $\mathcal{S}$-automaton can be simulated by an $(\mathcal{S} \circ_r (\mathcal{S}_{\mathrm{pd}} \circ_r \mathcal{S}_{\mathrm{pd}}))$-automaton. The idea is that the automaton follows one spine using the first component to control the expansion of the spine. Everything that is generated to the right of the spine is written on the lower pushdown, terminal and nonterminal symbols generated to the left of the spine are written on the upper pushdown. If the foot of the spine is reached the upper pushdown contains the left part of the derived sentential form in reversed order. The automaton continues by expanding the nonterminals on the upper pushdown, due to the reversed order on that store starting with the nonterminal directly to the left of the foot of the spine that is just reached. If the storage is in a semi-empty configuration the nonterminals on the lower pushdown are expanded. Thus the automaton simulates an inside-out derivation (see Sect. 3).

Lemma 3 *For each class of storages* $\mathcal{S}$,

$$L(\mathcal{G}_{\mathrm{LD}}/L(\mathcal{S})) \subseteq L(\mathcal{S} \circ_r (\mathcal{S}_{\mathrm{pd}} \circ_r \mathcal{S}_{\mathrm{pd}})).$$

Proof Let $\mathcal{S}$ be a class of storages, let $S = (C, c_\epsilon, F, id, m) \in \mathcal{S}$ and let $K = (G, L(M))$ be an LCG with $G = (N, \Sigma, R, A_{\mathrm{in}})$ an LDG and $M = (Q, R, S, \delta, Q_I, Q_F)$ an automaton. Construct an automaton

$$M' = (Q \times N_\epsilon, \Sigma, (S \circ_r (S_{\mathrm{pd}}(\Gamma) \circ_r S_{\mathrm{pd}}(\Gamma)), \delta', Q'_I, Q'_F)$$

with $Q'_I = \{(q, A_{\mathrm{in}}) \mid q \in Q_I\}$, $Q'_F = \{(q, \epsilon) \mid q \in Q_F\}$ and $\Gamma = N \cup \Sigma$, by setting

$$\begin{aligned}
\delta' = {} & \{((q_1, A), \epsilon, (q_2, B)), \lhd f \,\&\, \rhd \lhd \mathsf{push}(\beta_1^R) \,\&\, \rhd \rhd \mathsf{push}(\beta_2) \mid \\
& \quad r = A \to \beta_1 B ! \beta_2 \in R \text{ and } (q_1, r, q_2, f) \in \delta\} && (1)\\
& \cup \{((q_1, A), \epsilon, (q_2, \epsilon), \lhd f \,\&\, \rhd \lhd \mathsf{push}(v^R) \,\&\, \rhd \rhd \mathsf{push}(w)) \mid \\
& \quad r = A \to v ! w \in R \text{ and } (q_1, r, q_2, f) \in \delta\} && (2)\\
& \cup \{((q_1, A), \epsilon, (q_2, \epsilon), \lhd f) \mid \\
& \quad r = A \to \epsilon \in R \text{ and } (q_1, r, q_2, f) \in \delta\} && (3)\\
& \cup \{((q_1, A), \epsilon, (q_2, A), \lhd f) \mid A \in N_\epsilon \text{ and } (q_1, \epsilon, q_2, f) \in \delta\} && (4)\\
& \cup \{((q, \epsilon), \epsilon, (q_0, A), \rhd \lhd \mathsf{pop}(A)) \mid A \in N, q_0 \in Q_I, q \in Q_F\} && (5a)\\
& \cup \{((q, \epsilon), \epsilon, (q_0, A), \rhd \rhd \mathsf{pop}(A)) \mid A \in N, q_0 \in Q_I, q \in Q_F\} && (5b)\\
& \cup \{((q, \epsilon), a, (q, \epsilon), \rhd \lhd \mathsf{pop}(a)) \mid a \in \Sigma, q \in Q_F\} && (6a)\\
& \cup \{((q, \epsilon), a, (q, \epsilon), \rhd \rhd \mathsf{pop}(a)) \mid a \in \Sigma, q \in Q_F\}. && (6b)
\end{aligned}$$

The construction is very similar to the one used in the proof of Lemma 1. Again the proof starts establishing the relation between the derivation of a spine and its control word on the one hand side and the computation of M' on the other. For G we use an inside-out derivation in order to be able to follow the expansion of the distinguished children. This in fact is similar to the situation in the proof of Lemma 1, in which we simulated a leftmost derivation, which in the case of ELLDGs is the same as an inside-out derivation. For the rest of the proof all derivations are assumed to be inside-out if nothing else is indicated. By induction on the number of steps in a computation and in a derivation, respectively, it can be shown that

$$\begin{aligned}
& ((q_1, A), a, (c_1, (\epsilon, \epsilon))) \underset{M'}{\vdash}^* ((q_2, \epsilon), \epsilon, (c_\epsilon, (Y_1 \cdots Y_m, X_1 \cdots X_k))) \quad \text{and}\\
& M' \text{ does not use transitions of type (5) and (6) in this computation}\\
& \qquad\qquad \text{iff}\\
& \text{there exists } \omega \in R^+ \text{ such that}\\
& (A, \epsilon) \underset{G}{\Rightarrow}^* (Y_m, \epsilon) \cdots (Y_1, \epsilon)(a, \omega)(X_1, \epsilon) \cdots (X_k, \epsilon) \quad \text{and}\\
& (q_1, \omega, c_1) \underset{M}{\vdash}^* (q_2, \epsilon, c_\epsilon)
\end{aligned}$$

with $q_1 \in Q$, $q_2 \in Q_F$, $A \in N$, $a \in \Sigma_\epsilon$, $c_1 \in C$, $Y_1, \ldots, Y_m \in N \cup \Sigma$ and $X_1, \ldots, X_k \in N \cup \Sigma$. Once this implication is established, again by induction, we

can show that for all $A \in N$, $x \in \Sigma^*$, $Y_1, \ldots, Y_m \in N \cup \Sigma$ and $X_1, \ldots, X_k \in N \cup \Sigma$

there exist $q_1 \in Q_I$ and $q_2 \in Q_F$ such that
$((q_1, A), x, (c_\epsilon, (Y_1 \cdots Y_m, X_1 \cdots X_k))) \vdash_{M'}^* ((q_2, \epsilon), \epsilon, (c_\epsilon, (\epsilon, \epsilon)))$
iff
there exist $\omega_1, \ldots, \omega_n \in L(M)_\epsilon$ and $a_1, \ldots, a_n \in \Sigma_\epsilon$ such that
$(Y_m, \epsilon) \cdots (Y_1, \epsilon)(A, \epsilon)(X_1, \epsilon) \cdots (X_k, \epsilon) \Rightarrow_G^*$
$(a_1, \omega_1) \cdots (a_n, \omega_n)$ and $x = a_1 \cdots a_n$. □

From this general relation between linear control and concatenation we get an alternative proof for Theorem 4.3 of [5]:

Proposition 15 (Theorem 4.3 of [5]) *For each* $i > 1$: $\mathcal{W}_i \subset \mathcal{C}^{\mathcal{R}}_{2i-1}$.

Proof First we show by induction that $\mathcal{W}_i \subseteq \mathcal{C}^{\mathcal{R}}_{2i-1}$ for $i \geq 1$. For $i = 1$ the proposition is trivially true, since $\mathcal{W}_1 = \mathcal{C}^{\mathcal{R}}_1 = \mathcal{L}_{\mathrm{CF}}$. Suppose that the assertion is true for $i \in \mathbb{N}$ and let $\mathcal{S}_i$ be the class of storages such that $L(\mathcal{S}_i) = \mathcal{C}^{\mathcal{R}}_i$. For $i + 1$ we find:

$$\begin{aligned}
\mathcal{W}_{i+1} &= L(\mathcal{G}_{\mathrm{LD}}/\mathcal{W}_i) && \text{(by definition)}\\
&\subseteq L(\mathcal{G}_{\mathrm{LD}}/\mathcal{C}^{\mathcal{R}}_{2i-1}) && \text{(by induction)}\\
&= L(\mathcal{G}_{\mathrm{LD}}/L(\mathcal{S}_{2i-1}))\\
&\subseteq L(\mathcal{S}_{2i-1} \circ_r (\mathcal{S}_{\mathrm{pd}} \circ_r \mathcal{S}_{\mathrm{pd}})) && \text{(by Lemma 3)}\\
&= L((\mathcal{S}_{2i-1} \circ_r \mathcal{S}_{\mathrm{pd}}) \circ_r \mathcal{S}_{\mathrm{pd}}) && \text{(by the associativity of } \circ_r)\\
&= L(\mathcal{S}_{2i+1}) = \mathcal{C}^{\mathcal{R}}_{2i+1} = \mathcal{C}^{\mathcal{R}}_{2(i+1)-1} && \text{(by definition).}
\end{aligned}$$

It is known that both $\mathcal{C}^{\mathcal{R}}_i$ and $\mathcal{W}_i$ contain the language $\{a_1^n \cdots a_{2i}^n \mid n \in \mathbb{N}\}$ but not the language $\{a_1^n \cdots a_{2i+1}^n \mid n \in \mathbb{N}\}$ (cf. [6, Statement 2.1] and [27, p. 244]). Thus $\mathcal{C}^{\mathcal{R}}_{2i-1}$ contains the language $\{a_1^n \cdots a_{4i-2}^n \mid n \in \mathbb{N}\}$ whereas this language is not contained in $\mathcal{W}_i$ if $4i - 2 > 2i$. Thus the inclusion is proper for $i > 1$. □

This result combined with the fact that $\mathcal{C}^{\mathcal{R}}_i \subseteq \mathcal{W}_i$ (Corollary 2) implies that the languages from the multi-pushdown hierarchy are the same as those in Weir's hierarchy.

Corollary 3 (Corollary 4.3 of [5]) $\bigcup_{i=0}^{\infty} \mathcal{W}_i = \bigcup_{i=0}^{\infty} \mathcal{C}^{\mathcal{R}}_i$.

Let us finally consider the well-known hierarchy of Khabbaz [15, 16] that can be defined by control too, letting $\mathcal{K}_0 = \mathcal{L}_{\mathrm{CF}}$ and $\mathcal{K}_i = L(\mathcal{G}_{\mathrm{lin}}/\mathcal{K}_{i-1})$. Though we have nothing new to offer concerning this hierarchy, we repeat some results from [5] to round off the picture of hierarchies of controlled languages.

Proposition 16 (Theorem 6.11 of [13]) *For each* $i \geq 1$, $\mathcal{K}_i$ *is not closed under concatenation*; $\bigcup_{i=1}^{\infty} \mathcal{K}_i$ *is not closed under Kleene+*. [11]

[11] The theorem of Greibach states more generally that the proposition holds for each hierarchy defined as $\mathcal{L}_i = L(\mathcal{G}_{\mathrm{lin}}/\mathcal{L}_{i-1})$ and $\mathcal{L}_0$ a full semi-AFL.

Proposition 17 *For each $i \geq 0$, $\mathcal{W}_i$ is closed under union, concatenation and Kleene+.*

Proof The proof can be done straightforwardly by induction using Propositions 8 and 9 starting with the fact that $\mathcal{L}_{\mathrm{REG}}$ is closed under union, concatenation and Kleene+. Note that concatenation with a symbol is a special case of concatenation and Proposition 8 thus can be applied in each step. □

Corollary 4 (Corollaries 3.2 and 4.2 of [5]) *For each $i \geq 1$: $\mathcal{K}_i \subset \mathcal{W}_{i+1}$, $\mathcal{K}_i \subset \mathcal{C}^{\mathcal{R}}_{i+1}$ and $\mathcal{K}_i \subset \mathcal{C}^{\mathcal{W}}_{i+1}$.*

Proof By the definitions of the language classes we find immediately $\mathcal{K}_i \subseteq \mathcal{W}_{i+1}$. From Propositions 16 and 17 it follows that $\mathcal{K}_i \neq \mathcal{W}_{i+1}$. The other cases are shown similarly. □

Corollary 5 (Corollary 3.3 of [5]) $\bigcup_{i=0}^{\infty} \mathcal{K}_i \subset \bigcup_{i=0}^{\infty} \mathcal{W}_i$.

Proof The inclusion is clear by the definitions of the language classes and the previous corollary. From Proposition 17 it follows that $\bigcup_{i=0}^{\infty} \mathcal{W}_i$ is closed under Kleene+. Since $\bigcup_{i=0}^{\infty} \mathcal{K}_i$ is not closed under Kleene+ we find $\bigcup_{i=0}^{\infty} \mathcal{K}_i \neq \bigcup_{i=0}^{\infty} \mathcal{W}_i$. □

6 Conclusion

The main result of this paper is captured in Theorems 1 and 2 establishing a relation between control of ELLDGs and ERLDGs on the one hand and the composition of new classes of storages from pushdowns with $\circ_r$ and $\circ_w$ on the other hand. On the basis of this result some relations between hierarchies defined by iterative control and iterative concatenation of pushdowns were shown. Thus it was found that $\bigcup_{i=0}^{\infty} \mathcal{C}^{\mathcal{R}}_i = \bigcup_{i=0}^{\infty} \mathcal{C}^{\mathcal{W}}_i = \bigcup_{i=0}^{\infty} \mathcal{W}_i$ where for $i \in \mathbb{N}$, $\mathcal{C}^{\mathcal{R}}_i$ denote the classes of the multi-pushdown hierarchy established by Breveglieri et al., $\mathcal{C}^{\mathcal{W}}_i$ denote the classes of the reverse hierarchy and finally $\mathcal{W}_i$ are the classes of a hierarchy defined by Weir. For the inclusion of Weir's language classes in the multi-pushdown hierarchy we found $\mathcal{W}_i \subseteq \mathcal{C}^{\mathcal{R}}_j$ for $j = 2i - 1$. It remains an open question whether the inclusion holds for some smaller j.

Acknowledgements This paper is largely based on research carried out at the University Potsdam within the *Innovationskolleg* '*Formale Modelle kognitiver Komplexität*' (*INK 12*) funded by the DFG. We acknowledge the constructive comments of an anonymous referee, in particular the suggestion of Proposition 13. We would like to thank Peter Staudacher, Jens Michaelis and Lothar Budach for many helpful discussions and comments. Finally we thank Joost Engelfriet for his numerous useful suggestions on earlier versions of this paper.

References

1. Breveglieri, L., Cherubini, A., Citrini, C., Crespi Reghizzi, S.: Multi-push-down languages and grammars. Int. J. Found. Comput. Sci. **7**(3), 253–291 (1996)

2. Budach, L.: Environments, labyrinths and automata. In: Karpiński, M. (ed.) Fundamentals of Computation Theory. Lecture Notes in Computer Science, vol. 56, pp. 54–64. Springer, Berlin (1977)
3. Budach, L.: Personal communication (1999)
4. Cherubini, A., San Pietro, P.: On the relation between multi-depth grammars and tree-adjoining grammars. Publ. Math. Debrecen **54**(Supplement), 625–640 (1999)
5. Cherubini, A., San Pietro, P.: On the relations between multi-depth grammars and label-distinguished control grammars. In: Nehaniv, C.L., Ito, M. (eds.) Algebraic Engineering. World Scientific, Singapore (1999)
6. Cherubini, A., San Pietro, P.: Tree adjoining languages and multipushdown languages. Theory Comput. Syst. **33**(4), 257–293 (2000)
7. Dassow, J., Mitrana, V.: Stack cooperation in multistack pushdown automata. J. Comput. Syst. Sci. **58**, 611–621 (1999)
8. Duske, J., Parchmann, R.: Linear indexed languages. Theor. Comput. Sci. **32**(1–2), 47–60 (1984)
9. Engelfriet, J.: Context-free grammars with storage. Report 86–11, Vakgroep Informatica, Rijks-Universiteit Leiden (1986)
10. Gazdar, G.: Applicability of indexed grammars to natural languages. In: Reyle, U., Rohrer, C. (eds.) Natural Language Parsing and Linguistic Theories, pp. 69–94. Reidel, Dordrecht (1988)
11. Ginsburg, S.: Automata-theoretic Properties of Formal Languages. North-Holland, Amsterdam (1975)
12. Ginsburg, S., Spanier, E.H.: Control sets on grammars. Math. Syst. Theory **2**(2), 159–177 (1968)
13. Greibach, S.A.: Control sets on context-free grammar forms. J. Comput. Syst. Sci. **15**(1), 35–98 (1977)
14. Joshi, A.K.: Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In: Dowty, D., Karttunen, L., Zwicky, A. (eds.) Natural Language Parsing. Theoretical, Computational and Psychological Perspective, pp. 206–250. Cambridge University Press, Cambridge (1985)
15. Khabbaz, N.A.: Generalized context free languages. Ph.D. thesis, University of Iowa (1972)
16. Khabbaz, N.A.: A geometric hierarchy of languages. J. Comput. Syst. Sci. **8**(2), 142–157 (1974)
17. Michaelis, J., Wartena, C.: How linguistic constraints on movement conspire to yield languages analyzable with a restricted form of LIGs. In: Proceedings of the Conference on Formal Grammar (FG'97), pp. 158–168, Aix en Provence, 1997
18. Michaelis, J., Wartena, C.: LIGs with reduced derivation sets. In: Bouma, G., Kruijff, G.-J.M., Hinrichs, E., Oehrle, R.T. (eds.) Constraints and Resources in Natural Language Syntax and Semantics. Studies in Constrained Based Lexicalism, vol. II, pp. 263–279. CSLI, Stanford (1999)
19. Rado, J.: Topic-focus vs. background: the role of structural information in discourse interpretation. Ph.D. thesis, University of Massachusetts, Amherst (1997)
20. San Pietro, P.: Two-stack automata. Internal report 92–073, Dipartimento di Elettronica e Informazione, Politecnico di Milano (1992)
21. Scott, D.: Some definitional suggestions for automata theory. J. Comput. Syst. Sci. **1**, 187–212 (1967)
22. Vogler, H.: Iterated linear control and iterated one-turn pushdowns. Math. Syst. Theory **19**(2), 117–133 (1986)
23. Wartena, C.: On the concatenation of one-turn pushdowns. Grammars **2**(3), 259–269 (1999)
24. Wartena, C.: Storage structures and conditions on movement in natural languages. Ph.D. thesis, Universität Potsdam (1999)
25. Wartena, C.: Extending linear indexed grammars. In: Proceedings of the TAG+5 Workshop, pp. 207–214, Paris, 2000
26. Wartena, C.: Grammars with composite storages. In: Moortgat, M. (ed.) Logical Aspects of Computational Linguistics, pp. 266–285. Springer, Berlin/Heidelberg (2001)
27. Weir, D.J.: A geometric hierarchy beyond context-free languages. Theor. Comput. Sci. **104**(4), 235–261 (1992)
28. Weir, D.J.: Linear iterated pushdowns. Comput. Intell. **10**(4), 422–430 (1994)