Cyclic Shift on Multi-component Grammars

Alexander Okhotin[1(✉)] and Alexey Sorokin[2,3]

[1] St. Petersburg State University, Saint Petersburg, Russia
alexander.okhotin@spbu.ru
[2] Moscow State University, Moscow, Russia
alexey.sorokin@list.ru
[3] Moscow Institute of Physics and Technology, Dolgoprudny, Russia

Abstract. Multi-component grammars, known in the literature as "multiple context-free grammars" and "linear context-free rewriting systems", describe the structure of a string by defining the properties of k-tuples of its substrings, in the same way as ordinary formal grammars (Chomsky's "context-free") define properties of substrings. It is shown that, for every fixed k, the family of languages described by k-component grammars is closed under the cyclic shift operation. On the other hand, the subfamily defined by well-nested k-component grammars is not closed under the cyclic shift, yet their cyclic shifts are always defined by well-nested $(k+1)$-component grammars.

1 Introduction

The cyclic shift operation on formal languages, defined as $\text{SHIFT}(L) = \{vu \mid uv \in L\}$ for a language L, is notable for several interesting properties. The closure of the class of regular languages under this operation is likely folklore, and proving it is a standard exercise in automata theory [2, Exercise 3.4(c)]. An interesting detail is that the cyclic shift incurs a huge blow-up in the number of states in a DFA, which is of the order $2^{n^2+n\log n-O(n)}$. [3,9] An analogous (quite an unobvious one) result for context-free grammars was first discovered by Maslov [10] and by Oshiba [12], and a direct construction of a grammar was later presented in the textbook by Hopcroft and Ullman [2, Exercise 6.4(c)]. In their proof, a grammar describing a language L is transformed to a grammar for the cyclic shift of L, and the transformation *turns the grammar inside out*, so that each parse tree in the new grammar simulates a parse tree in the original grammar, while reversing the order of nodes on one of its paths.

In contrast to this remarkable closure result, all noteworthy subfamilies of the ordinary grammars—that is, unambiguous, LR, LL, linear, input-driven, etc.—are not closed under the cyclic shift. A non-closure result for the linear conjunctive languages [11] was established by Terrier [17]. For conjunctive grammars [11], whether they are closed under the cyclic shift, remains an open

Research supported by Russian Science Foundation, project 18-11-00100.

A. Leporati et al. (Eds.): LATA 2020, LNCS 12038, pp. 287–299, 2020.
https://doi.org/10.1007/978-3-030-40608-0_20

problem. A summary of these results can be found in a fairly recent survey [11, Sect. 8.2].

This paper investigates the cyclic shift operation on one of the most well-known families of formal grammars, the *multi-component grammars*. These grammars describe the syntax of a string by defining the properties of k-tuples of its substrings, in the same way as ordinary formal grammars and their basic variants, such as conjunctive grammars, define properties of individual substrings. In their modern form, multi-component grammars were independently introduced by Seki, Matsumura, Fujii and Kasami [14] (as "multiple context-free grammars", MCFG), and by Vijay-Shankar, Weir and Joshi [18] (as "linear context-free rewriting systems", LCFRS). These grammars are subject to much ongoing research [1,7,8,19]. Also much attention is given to their special case: the *well-nested multi-component grammars*, in which all components of any intermediate k-tuple are listed in the order, in which they occur in the final string, and the grammar rules combine these k-tuples. This family is believed to correspond to the natural language syntax better than other grammar formalisms.

The first result of this paper is the closure of the language family defined by k-component grammars under the cyclic shift operation. The proof, presented in Sect. 3, proceeds by transforming an arbitrary k-component grammar to another k-component grammar describing the cyclic shift of the original language.

However, this construction does not preserve well-nestedness. A new construction adapted for well-nested grammars is presented in Sect. 4, and it incurs the increase of the number of components by one. In the final Sect. 5, it is shown that, whereas the language $\{a_1^m b_1^n c_1^n d_1^m \dots a_k^m b_k^n c_k^n d_k^m \mid m, n \geqslant 0\}$ is defined by a well-nested k-component grammar, its cyclic shift is defined by no grammar from this class, and accordingly requires $k+1$ components. This points out a peculiar difference between the general and the well-nested cases of multi-component grammars.

2 Multi-component Grammars

Definition 1. (Vijay-Shankar et al. [18]; Seki et al. [14]). *A multi-component grammar is a quintuple $G = (\Sigma, N, \dim, R, S)$, where*

- *Σ is the alphabet of the language being described;*
- *N is the set of syntactic categories defined in the grammar, usually called "nonterminal symbols";*
- *$\dim\colon N \to \mathbb{N}$ is a function that defines the number of components in each nonterminal symbol, so that if $\dim A = k$, then A describes k-tuples of substrings;*
- *R is a set of grammar rules, each of the form*

$$A(\alpha_1, \dots, \alpha_{\dim A}) \leftarrow B_1(x_{1,1}, \dots, x_{1,\dim B_1}), \dots, B_\ell(x_{\ell,1}, \dots, x_{\ell,\dim B_\ell}), \quad (*)$$

where $\ell \geqslant 0$, the variables $x_{i,j}$ are pairwise distinct, $\alpha_1, \dots, \alpha_{\dim A}$ are strings over symbols from Σ and variables $x_{i,j}$, and each variable $x_{i,j}$ occurs in $\alpha_1 \dots \alpha_{\dim A}$ exactly once;

– *a nonterminal symbol $S \in N$ of dimension 1 is the "initial symbol", that is, the category of all well-formed sentences defined by the grammar.*

A grammar is a logical system for proving elementary propositions of the form $A(w_1, \ldots, w_k)$, with $k = \dim A$ and $w_1, \ldots, w_k \in \Sigma^*$, meaning that the given k-tuple of strings has the property A. A proof proceeds using the rules in R, with each rule (*) treated as a schema for derivation rules, for any strings substituted for all variables $x_{i,j}$.

$$B_1(x_{1,1}, \ldots, x_{1,\dim B_1}), \ldots, B_\ell(x_{\ell,1}, \ldots, x_{\ell,\dim B_\ell}) \vdash A(\alpha_1, \ldots, \alpha_{\dim A})$$

The language generated by the grammar, denoted by $L(G)$, is the set of all such strings w that the proposition $S(w)$ can be derived in one or more such steps.

Whenever a string w is generated by G, the derivation of a proposition $S(w)$ forms a *parse tree*. Each node in the tree is labelled with a proposition $A(w_1, \ldots, w_k)$, where $k = \dim A$ and $w_1, \ldots, w_k$ are substrings of w. Every node has a corresponding rule (*), by which the proposition is derived, and the direct successors of this node are labelled with $B_1(x_{1,1}, \ldots, x_{1,\dim B_1})$, $\ldots$, $B_\ell(x_{\ell,1}, \ldots, x_{\ell,\dim B_\ell})$, as in the definition of a derivation step.

The dimension of a grammar, $\dim G$, is the largest dimension of a nonterminal symbol. A multi-component grammar of dimension k shall be called a k-component grammar.

A special case of these grammars are *well-nested multi-component grammars*, in which, whenever multiple constituents are joined in a single rule, their components cannot be intertwined, unless one's components are completely embedded within another's components. Thus, patterns such as $A(x_1y_1, x_2y_2) \leftarrow B(x_1, x_2)C(y_1, y_2)$ are prohibited.

Definition 2. *A multi-component grammar is called well-nested, if every rule (*), satisfies the following conditions.*

1. *(non-permuting condition) For every i, the variables $x_{i,1}, \ldots, x_{i,\dim B_i}$ occur inside $\alpha_1 \ldots \alpha_{\dim A}$ in this particular order.*
2. *For all i, i' the concatenation $\alpha_1 \ldots \alpha_{\dim A}$ satisfies one of the following patterns:*
   - $\ldots x_{i,d_i} \ldots x_{i',1} \ldots$.
   - $\ldots x_{i',d_{i'}} \ldots x_{i,1} \ldots$.
   - $\ldots x_{i,r} \ldots x_{i',1} \ldots x_{i',d_{i'}} \ldots x_{i,r+1} \ldots$.

Example 1. A language $L = \{a^m b^n c^m d^n \mid m, n \in \mathbb{N}\}$ is defined by a 2-component grammar with the rules

$$\begin{aligned} S(x_1y_1x_2y_2) &\leftarrow A(x_1, x_2), B(y_1, y_2), \\ A(ax_1, cx_2) &\leftarrow A(x_1, x_2), \\ B(by_1, dy_2) &\leftarrow B(y_1, y_2). \end{aligned}$$

A well-nested 2-component grammar for the same language is

$$\begin{aligned} S(x_1x_2) &\leftarrow A(x_1, x_2), \\ A(x_1, bx_2d) &\leftarrow A(x_1, x_2), \\ A(x_1, x_2) &\leftarrow B(x_1, x_2), \\ B(ax_1, cx_2) &\leftarrow B(x_1, x_2). \end{aligned}$$

A well-nested multi-component grammar can be transformed to the following form resembling the Chomsky normal form.

Proposition 1. (**[15]**, **Thm. 1**). *Each well-nested k-component grammar is equivalent to a well-nested k-component grammar, in which all rules are of the following form.*

$$\begin{aligned} A(x_1, \ldots, x_{m-1}, x_m y_1, y_2, \ldots, y_n) &\leftarrow B(x_1, \ldots, x_m), C(y_1, \ldots, y_n) \\ A(x_1, \ldots, x_i, x_i y_1, y_2, \ldots y_n x_{i+1}, x_{i+2}, \ldots, x_m) &\leftarrow B(x_1, \ldots, x_m), C(y_1, \ldots, y_n) \\ A(a) &\leftarrow \\ S(\varepsilon) &\leftarrow \end{aligned}$$

Rules of the first kind generalize the concatenation. The operation implemented in the rules of the second kind, defined for $i \in \{1, \ldots, m-1\}$, is known as ***displacement*** *or* ***discontinuous product****.*

A multi-component grammar of dimension 1 is an *ordinary grammar*, or "context-free" in Chomsky's terminology. A well-nested multi-component grammar of dimension 2 is known in the literature as a "head grammar" [13]; these grammars are equivalent in power to tree-adjoining grammars [4].

3 Cyclic Shift on k-component Grammars

Let G be a non-permuting k-component grammar, the goal is to construct a new k-component grammar G' that describes the language $\textsc{shift}(L(G))$.

Whenever G generates a string w, G' should generate vu for every partition $w = uv$. Consider a parse tree of uv according to G, that is, a proof tree of the proposition $S(uv)$. Each node in the tree is labelled with a proposition $A(w_1, \ldots, w_k)$, where $k = \dim A$ and $w_1, \ldots, w_k$ are substrings of w. We call a node *split*, if one of its components w_s spans over the boundary between u and v, that is, contains both the last symbol of u and the first symbol of v.

In the proposed construction of a grammar for the cyclic shift, each split node $A(w_1, \ldots, w_k)$ is represented by another node of dimension k, which, however, specifies an entirely different k-tuple of strings. Consider that, whenever the original split node $A(w_1, \ldots, w_k)$ is used in a parse tree of a string uv, this string contains $w_1, \ldots, w_k$ as substrings, in any order. The corresponding node in the parse tree of vu according to the grammar for the cyclic shift shall contain all symbols of uv *except* the symbols in $w_1, \ldots, w_k$. For the moment, assume that $w_1, \ldots, w_k$ occur in uv in the order listed, and that some w_s spans over

the boundary between u and v. Then, $uv = y_0 w_1 y_1 w_2 y_2 \dots y_{k-1} w_k y_k$, and the symbols not in $w_1, \dots, w_k$ are arranged into $k+1$ substrings $y_0, \dots, y_k$. However, note that in the string vu generated by the new grammar, y_k and y_0 come concatenated as a single substring $y_k y_0$, and there is no need to represent them as separate components. Therefore, the new grammar can represent this split node $A(w_1, \dots, w_k)$ by another node $\widetilde{A}(y_k y_0, y_1, \dots, y_{k-1})$ of the same dimension k, where $\widetilde{A}$ is a new nonterminal symbol representing *the whole string with a gap for a k-tuple generated by A*.

To see how this transformation can be done, the structure of split nodes in the original parse tree ought to be examined, As long as $u \neq \varepsilon$ and $v \neq \varepsilon$, the root $S(uv)$ is split. Each split node has at most one split node among its immediate successors, because the last symbol of u and the first symbol of v cannot be in two successors at once. If a node is not split, then none of its successors are split. Thus, split nodes form a path in a parse tree, beginning in the root and ending somewhere inside the tree. This path shall be called the *main path*, and the new grammar G' retraces this path using the nonterminal symbols of the form $\widetilde{A}$.

In the original grammar, whenever a rule $A(\dots) \leftarrow B(\dots), C(\dots)$ is used in one of the nodes on the main path, where B is the next node along the path, shorter substrings described by B are concatenated to something taken from C to form longer substrings described by A. In the new grammar, a nonterminal symbol $\widetilde{A}$ generates all symbols of the string *except* those generated by A, whereas $\widetilde{B}$ generates all symbols except the symbols generated by B. Therefore, $\widetilde{B}$ can be defined by a rule that partially fills the gap for A in $\widetilde{A}$, replacing it with a smaller gap for B in $\widetilde{B}$. This is achieved by a rule $\widetilde{B}(\dots) \leftarrow \widetilde{A}(\dots), C(\dots)$. The node $\widetilde{B}$ is accordingly higher up than $\widetilde{A}$ in the parse tree of vu, and the main path of the original parse tree is retraced in the reverse direction. Each rule along the path is inverted, and the parse tree is effectively *turned inside out.*

Theorem 1. *For every k-component grammar G with n nonterminal symbols, there exists another k-component grammar with at most $(k!+1)n$ nonterminal symbols that describes the language* $\textsc{shift}(L(G))$.

Proof. Let $G = (\Sigma, N, \dim, R, S)$, The new grammar is defined as $G' = (\Sigma, N \cup \widetilde{N} \cup \{S'\}, \dim, R \cup R', S')$, where every new nonterminal symbol in $\widetilde{N}$ is of the form $\widetilde{A}_{p_1,\dots,p_k}$, where $A \in N$ is a symbol of dimension k, and $(p_1, \dots, p_k)$ is a permutation of $(1, \dots, k)$; the dimension of this new symbol is also k.

Each symbol from N is defined in G' by the same rules as in G, and hence $L_{G'}(A) = L_G(A)$ for all $A \in N$. For each new symbol $\widetilde{A}_{p_1,\dots,p_k}$ in $\widetilde{N}$, with $k = \dim A$, the intention is that it generates all such k-tuples $(w_0, \dots, w_{k-1})$ that, for some partition $w_0 = v_0 u_0$, a proposition $S(u_0 x_{p_1} w_1 x_{p_2} w_2 \dots w_{k-1} x_{p_k} v_0)$ can be derived using an assumption $A(x_1, \dots, x_k)$. In other words, a k-tuple generated by $\widetilde{A}_{p_1,\dots,p_k}$ is a string from $L(G)$ with k gaps, which should be filled by a k-tuple generated by A, Note that the components of $\widetilde{A}_{p_1,\dots,p_k}(w_0, w_1, \dots, w_{k-1})$ occur in the final string generated by the grammar G exactly in the given order, though w_0 is split into a suffix and a prefix. On the other hand, the components

of $A(x_1, \ldots, x_k)$ may occur in the final string in $L(G)$ in any order, and this order is specified in the permutation $p_1, \ldots, p_k$.

The grammar G' has three kinds of rules for the new symbols. The **first** rule creates an empty string with one gap for a string generated by S.

$$\widetilde{S}_1(\varepsilon) \leftarrow \tag{1}$$

Indeed, using an assumption $S(x)$, one can derive $S(x)$ in zero steps.

For the **second** type of rules in G', consider any rule in G, which defines a symbol A of dimension k, and fix any nonterminal symbol B on its right-hand side. Let $y_1, \ldots, y_\ell$ be the variables of B. Denote the remaining nonterminal symbols referenced in this rule by $C_1, \ldots, C_q$.

$$A(\alpha_1, \ldots, \alpha_k) \leftarrow B(y_1, \ldots, y_\ell), C_1(\ldots), \ldots, C_q(\ldots)$$

For every i-th argument of A, consider all occurrences of variables $y_1, \ldots, y_\ell$ in α_i, and accordingly let $\alpha_i = \beta_{i,0} y_{r_{i,1}} \beta_{i,1} \ldots \beta_{i,m_i-1} y_{r_{i,m_i}} \beta_{i,m_i}$, where $m_i \geqslant 0$ is the number of these occurrences, $\beta_{i,j}$ are strings over the alphabet Σ and over the variables of $C_1, \ldots, C_q$, and $r_{i,j} \in \{1, \ldots, \ell\}$, for each i. Since each variable is referenced exactly once, $m_1 + \ldots + m_k = \ell$ and $(r_{1,1}, \ldots, r_{1,m_1}, \ldots, r_{1,j}, \ldots, r_{k,m_k})$ is a permutation of $(1, \ldots, \ell)$.

To see how to transform this rule, consider any proposition $\widetilde{A}_{p_1,\ldots,p_k}(w_0, w_1, \ldots, w_{w-1})$, where $(p_1, \ldots, p_k)$ is a permutation of $(1, \ldots, k)$. This symbol represents a full string generated by G, with a gap for A. If A is derived from B and $C_1, \ldots C_q$ using the above rule for A, then the substrings obtained from $C_1, \ldots, C_q$ partially fill the gaps for A, leaving smaller gaps for B. The resulting symbol $\widetilde{B}_{p'_1,\ldots,p'_\ell}$ has ℓ gaps for B, and the permutation $(p'_1, \ldots, p'_\ell)$ of $(1, \ldots, \ell)$ is defined by listing the numbers of the variables of B in the order they occur as gaps: the sequence $y_{r_{p_1,1}}, \ldots, y_{r_{p_1,m_{p_1}}}, \ldots, y_{r_{p_k,1}}, \ldots, y_{r_{p_k,m_{p_k}}}$ is the same as $p'_1, \ldots, p'_\ell$.

The corresponding transformed rule in the new grammar has to fill the gaps in the right order. Let $z_0, z_1, \ldots, z_{w-1}$ be the variables of $\widetilde{A}_{p_1,\ldots,p_k}$. Then the circular sequence $z_0 \alpha_{p_1} z_1 \ldots z_{k-1} \alpha_{p_k}$ containing variables of $\widetilde{A}_{p_1,\ldots,p_k}$, B and $C_1, \ldots, C_j$ represents the entire string, and every occurrence of a variable of B becomes a gap in the new rule. Accordingly, the sequence between any two subsequent variables of B forms an argument of $\widetilde{B}_{p'_1,\ldots,p'_\ell}$. The first argument is the one containing z_0. The variables of B become gaps between the variables of $\widetilde{B}_{p'_1,\ldots,p'_\ell}$, and the resulting rule is defined as follows.

$$\begin{aligned}\widetilde{B}_{p'_1,\ldots,p'_\ell}(&\beta_{p_k,m_{p_k}} z_0 \beta_{p_1,0},\ \beta_{p_1,1},\ \ldots,\ \beta_{p_1,m_{p_1}-1},\ \beta_{p_1,m_{p_1}} z_1 \beta_{p_2,0},\ \beta_{p_2,1},\ \ldots,\\ &\beta_{p_{k-1},m_{p_{k-1}}-1},\ \beta_{p_{k-1},m_{p_{k-1}}} z_{k-1} \beta_{p_k,0},\ \beta_{p_k,1},\ \ldots, \beta_{p_k,m_{p_k}-1}) \leftarrow\\ &\leftarrow \widetilde{A}_{p_1,\ldots,p_k}(z_0, \ldots, z_{k-1}), C_1(\ldots), \ldots, C_q(\ldots)\end{aligned} \tag{2}$$

Rules of the **third** and the last type are defined for the initial symbol of the new grammar. They correspond to the bottom split node on the main path of the parse tree in G, where the last symbol of u and the first symbol of v

are finally assigned to different substrings. Denote the bottom split node by $A(x_1, \ldots, x_k)$, and let $u_0 x_{p_1} w_1 x_{p_2} w_2 \ldots w_{k-1} x_{p_k} v_0$ be the entire string generated by the original grammar. In the new grammar, the node $A(x_1, \ldots, x_k)$ is represented by a proposition $\widetilde{A}_{p_1,\ldots,p_k}(v_0 u_0, w_1, \ldots, w_{k-1})$. Let x_{p_s}, with $s \in \{1, \ldots, k\}$, be the split component of $A(x_1, \ldots, x_k)$. The plan is to fill the gaps in $\widetilde{A}_{p_1,\ldots,p_k}(v_0 u_0, w_1, \ldots, w_{k-1})$ with the symbols in the subtree of $A(x_1, \ldots, x_k)$. However, it is not possible to do this directly in a rule of the form $S'(\ldots) \leftarrow \widetilde{A}_{p_1,\ldots,p_k}(\ldots), A(x_1, \ldots, x_k)$, because the component x_{p_s} is split.

Consider the rule used to derive $A(x_1, \ldots, x_k)$ in the new grammar, and let $C_1, \ldots, C_\ell$ be all nonterminal symbols on its right-hand side.

$$A(\alpha_1, \ldots, \alpha_k) \leftarrow C_1(\ldots), \ldots, C_q(\ldots)$$

The split component α_{p_s} generates a substring $x_{p_s} = \widehat{x}_1 \widehat{x}_2$, where the first part $\widehat{x}_1$ is a suffix of u and the second part $\widehat{x}_2$ is a prefix of v. Let $\alpha_{p_s} = \eta\theta$ be a partition of α_{p_s} into the symbols generating $\widehat{x}_1$ and the symbols generating $\widehat{x}_2$. Then the new grammar has the following rule, where the components of A are inserted into the gaps in $\widetilde{A}_{p_1,\ldots,p_k}$, and the resulting string is cyclically shifted to begin in the middle of the component α_{p_s}.

$$\begin{aligned} S'(\theta z_s \alpha_{p_{s+1}} z_{s+1} \ldots z_{k-1} \alpha_{p_k} z_0 \alpha_{p_1} z_1 \ldots z_{s-2} \alpha_{p_{s-1}} z_{s-1} \eta) \leftarrow \\ \leftarrow \widetilde{A}_{p_1,\ldots,p_k}(z_0, \ldots, z_{k-1}), C_1(\ldots), \ldots, C_q(\ldots) \end{aligned} \tag{3}$$

Overall, for every two strings u and v, the string uv is in $L(G)$ if and only if vu belongs to $L(G')$.

It can be easily observed that our construction does not preserve well-nestedness. Consider the well-nested rule $A(x_1, ax_2b) \leftarrow A(x_1, x_2)$, by our construction it produces the rule $S'(ax_2by_2x_1y_1) \leftarrow A(x_1, x_2), \widetilde{A}_{(12)}(y_1, y_2)$, which is not well-nested.

4 Cyclic Shift on Well-Nested k-component Grammars

The construction for the cyclic shift in the case of well-nested grammars is generally easier, since it does not involve turning parse trees inside out. All paths in the transformed trees continue in the same direction, at the expense of using one extra component. On the other hand, special care has to be taken to preserve the order of components and their well-nestedness.

Theorem 2. *If a language is defined by a well-nested k-component grammar, then its cyclic shift can be defined by a well-nested $(k+1)$-component grammar.*

Proof. Assume that all rules in the original grammar G are as in Proposition 1. If G defines a string $w = uv$, the new grammar G' should generate vu. In the parse tree of uv according to G, a node $A(w_1, \ldots, w_k)$ is *split*, if one of its components w_s spans over the boundary between u and v. Let $w_s = w'_s w''_s$, where u ends with

w'_s, and v begins with w''_s. Then, the new grammar shall have a new nonterminal symbol $\widehat{A}_s$, which defines a $(k+1)$-tuple $\widehat{A}_s(w''_s, w_{s+1}, \ldots, w_k, w_1, \ldots, w_{s-1}, w'_s)$.

For a non-split node, let $w_1, \ldots, w_s$ be in u and let $w_{s+1}, \ldots w_k$ be in v. Then the new grammar has a new nonterminal symbol A_s with defines a shifted k-tuple $A_s(w_{s+1}, \ldots, w_k, w_1, \ldots, w_s)$. In particular, the nonterminal $\widehat{S}_1$, where S is the initial symbol of G, generates the language $L_{G'}(\widehat{S}_1) = \{(v, u) \,|\, uv \in L,\, u, v \neq \varepsilon\}$. Adding a new initial nonterminal S' and the rules $S'(xy) \leftarrow \widehat{S}_1(x, y)$ and $S'(w) \leftarrow S_1(w)$ then yields the grammar for the language $\textsc{shift}(L(G))$. What remains is to equip the newly introduced nonterminals with the rules that match their definitions.

For each concatenation rule $A(x_1, \ldots, x_{m-1}, x_m y_1, y_2, \ldots, y_n) \leftarrow B(x_1, \ldots, x_m), C(y_1, \ldots, y_n)$ in the original grammar, first, there are $m + n - 1$ non-split shifts, which simply rotate the order of the components. They are using the rules below corresponding to different shifts; note that in each case one of B, C remains unshifted, and the other is shifted and wrapped around it.

$$\begin{aligned} A_i(x_{i+1}, \ldots, x_{m-1}, x_m y_1, y_2, \ldots, y_n, x_1, \ldots, x_i) &\leftarrow \\ B_i(x_{i+1}, \ldots, x_m, x_1, \ldots, x_i), C(y_1, \ldots, y_n) &\qquad (i < m) \\ A_{m+i}(y_{i+1}, \ldots, y_n, x_1, \ldots, x_{m-1}, x_m y_1, y_2, \ldots, y_i) &\leftarrow \\ B(x_1, \ldots, x_m), C_i(y_{i+1}, \ldots, y_n, y_1, \ldots, y_i) &\qquad (i \geqslant 1) \end{aligned}$$

Secondly, the cyclic shift may split one of the components of this $(m + n - 1)$-tuple. This is implemented in $\widehat{A}_i$: then, one of B, C is unshifted, and the other is split. There are the following cases.

$$\begin{aligned} \widehat{A}_i(x''_i, x_{i+1}, \ldots, x_{m-1}, x_m y_1, y_2, \ldots, y_n, x_1, \ldots, x_{i-1}, x'_i) &\leftarrow \\ \widehat{B}_i(x''_i, x_{i+1}, \ldots, x_m, x_1, \ldots, x_{i-1}, x'_i), C(y_1, \ldots, y_n) &\qquad (i < m) \\ \widehat{A}_{m+i}(y''_i, y_{i+1}, \ldots, y_n, x_1, \ldots, x_{m-1}, x_m y_1, y_2, \ldots, y_{i-1}, y'_i) &\leftarrow \\ B(x_1, \ldots, x_m), \widehat{C}_i(y''_i, y_{i+1}, \ldots, y_n, y_1, \ldots, y_{i-1}, y'_i) &\qquad (1 \leqslant i \leqslant n) \end{aligned}$$

Consider a displacement rule $A(x_1, \ldots, x_{j-1}, x_j y_1, \ldots y_n x_{j+1}, \ldots, x_m) \leftarrow B(x_1, \ldots, x_m), C(y_1, \ldots, y_n)$ in G, with $j \in \{1, \ldots, m-1\}$. Again, there are non-split and split shifts. Non-split shifts fall into the following three cases.

$$\begin{aligned} A_i(x_{i+1}, \ldots, x_{j-1}, x_j y_1, y_2, \ldots y_n x_{j+1}, x_{j+2}, \ldots, x_m, x_1, \ldots, x_i) &\leftarrow \\ B_i(x_{i+1}, \ldots, x_m, x_1, \ldots, x_i), C(y_1, \ldots, y_n) &\qquad (i < j) \\ A_{j+i}(y_{i+1}, \ldots, y_n, x_{j+1}, \ldots, x_m, x_1, \ldots, x_i, y_1, y_2, \ldots, y_i) &\leftarrow \\ B(x_1, \ldots, x_m), C_i(y_{i+1}, \ldots, y_n, y_1, \ldots, y_i) &\qquad (1 \leqslant i \leqslant n) \\ A_{m-1+i}(x_{i+1}, \ldots, x_m, x_1, \ldots, x_j y_1, y_2, \ldots y_n x_{j+1}, \ldots, x_{i-1}) &\leftarrow \\ B_i(x_{i+1}, \ldots, x_m, x_1, \ldots, x_i), C(y_1, \ldots, y_n) &\qquad (i > j) \end{aligned}$$

If one of the components is split, the corresponding rule for $\widehat{A}_i$ is one of the following.

$$\widehat{A}_i(x_i'', x_{i+1}, \ldots, x_j y_1, y_2, \ldots y_n x_{j+1}, \ldots, x_m, x_1, \ldots, x_{i-1}, x_i') \leftarrow$$
$$\widehat{B}_i(x_i'', x_{i+1}, \ldots, x_m, x_1, \ldots, x_{i-1}, x_i'), C(y_1, \ldots, y_n) \qquad (i < j)$$
$$\widehat{A}_{m+i}(y_i'', y_{i+1}, \ldots, y_n x_{j+1}, \ldots, x_m, x_1, \ldots, x_j y_1, \ldots, y_{n-1}, y_i') \leftarrow$$
$$B(x_1, \ldots, x_m), \widehat{C}_i(y_i'', y_{i+1}, \ldots, y_n, y_1, \ldots, y_{i-1}, y_i') \quad (1 \leqslant i \leqslant n)$$
$$\widehat{A}_i(x_i'', x_{i+1}, \ldots, x_m, x_1, \ldots, x_j y_1, y_2, \ldots y_n x_{j+1}, \ldots, x_{i-1}, x_i') \leftarrow$$
$$\widehat{B}_i(x_i'', x_{i+1}, \ldots, x_m, x_1, \ldots, x_{i-1}, x_i'), C(y_1, \ldots, y_n) \qquad (i > j)$$

A correctness proof for the construction proceeds by induction on the size of derivations in the respective grammars, formalizing the above explanations. □

5 Number of Components in Well-Nested Grammars[1]

Theorem 2 shows how to represent the cyclic shift of a well-nested k-component grammar by a well-nested $(k+1)$-component grammar. On the other hand, without the well-nestedness restriction, a k-component grammar can be constructed by Theorem 1. The growth in the number of components is caused by keeping a split substring as two components. The question is, whether this weakness is an artefact of the construction, or is determined by the fundamental properties of well-nested grammars. In this section we prove, that for any $k \geqslant 2$, there exists a well-nested k-component grammar, whose cyclic shift lies outside this class; thus the result of the previous section cannot be strengthened.

As such a counterexample, we take a very simple language $\mathrm{EmbBal}(2, k)$, containing all the strings of the form $a_1^m b_1^n c_1^n d_1^m \ldots a_k^m b_k^n c_k^n d_k^m$, with $m, n \geqslant 0$, which is defined by a well-nested k-component grammar (see Example 2). It is claimed that the cyclic shift of this language cannot be represented by a well-nested k-component grammar. Since this language family is closed under rational transductions, it suffices to demonstrate that the language $\mathrm{NonEmbBal}(2, k) = \{a_1^m b_1^m c_1^n d_1^n \ldots a_k^m b_k^m c_k^n d_k^n \mid m, n > 0\}$ cannot be generated by a well-nested k-component grammar, because this language is obtained from $\mathrm{CyclicShift}(\mathrm{EmbBal}(2, k))$ by intersection with a regular language $b_1^+ \Sigma^* a_1^+$, and with a circular letter renaming $b_i \to a_i, c_i \to b_i, d_i \to c_i, a_i \to d_{i-1}, a_1 \to d_k$.

Example 2. The language $\mathrm{EmbBal}(2, k)$, containing all the strings of the form $a_1^m b_1^n c_1^n d_1^m \ldots a_k^m b_k^n c_k^n d_k^m$, with $m, n \geqslant 0$, is defined by the following well-nested k-component grammar.

$$S(x_1 \ldots x_k) \leftarrow A(x_1, \ldots, x_k)$$
$$A(a_1 x_1 d_1, \ldots, a_k x_k d_k) \leftarrow A(x_1, \ldots, x_k)$$
$$A(x_1, \ldots, x_k) \leftarrow B(x_1, \ldots, x_k)$$
$$B(b_1 x_1 c_1, \ldots, b_k x_k c_k) \leftarrow B(x_1, \ldots, x_k)$$
$$B(\varepsilon, \ldots, \varepsilon) \leftarrow$$

[1] Most of the proofs are omitted due to space restrictions.

The definitions below are taken from Kanazawa [5].

Definition 3. *An r-pump D is a nonempty derivation of the form $D\colon A(x_1,\ldots,x_r)\vdash A(y_1,\ldots,y_r)$.*

Note that in case of a well-nested grammar in Chomsky normal form, $x_1\ldots x_r$ is a proper subsequence of $y_1\ldots y_r$. For each pump D, we define the sequence of its pumping strings: $\mathrm{strings}(D)=\bigcup_{i=1}^{r}[w_{i,j}|y_i=w_{i,0}x_sw_{i,1}\ldots x_{s+t}w_{i,t}]$. For example, the derivation $A(x_1,x_2,x_3)\vdash A(ax_1bcx_2,a,bx_2)$ produces the pumping sequence $[a,bc,\varepsilon,a,b,\varepsilon]$. Informally, the pumping strings are maximal contiguous strings that the pump subtree injects into the derived string. It is easy to prove that the pumping sequence of an r-pump consists of exactly $2r$ strings.

Definition 4. *An even r-pump is a nonempty derivation of the form $D\colon A(x_1,\ldots,x_r)\vdash A(u_1x_1v_1,\ldots,u_rx_rv_r)$.*

Obviously, for an even pump D the pumping strings are $\mathrm{strings}(D)=[u_1,v_1,\ldots,u_n,v_n]$.

We use the term "pump" not only for derivations, but also for derivation trees. Given a derivation tree, we call a letter occurrence *covered* if it occurs in the yield of some pump, and *evenly covered* if this pump is even.

In what follows we consider only grammars in the Chomsky normal form, as in Proposition 1. The following lemma is a mathematical folklore for context-free grammars, the proof for well-nested multicomponent grammars is the same.

Lemma 1. *For every language L defined by a well-nested grammar, there exists a number p, such that for every $w\in L$ at most $p-1$ letters are not covered.*

In the case of ordinary grammars (well-nested 1-component grammars), this lemma implies a weak version of the Ogden property [6,16] However, as shown by Kanazawa and Salvati [8], that is not the case for well-nested grammars of higher dimensions. Namely, the existence of an uneven pump does not imply the k-pumping lemma. However, in our case we may get rid of uneven pumps.

Definition 5. *A language is called bounded if it is a subset of the language $a_1^+\ldots a_m^+$, for some symbols $a_1,\ldots,a_m\in\Sigma$. A language is strictly bounded if all the symbols $a_1,\ldots,a_m$ are distinct.*

For a bounded language $L\subseteq a_1^+\ldots a_m^+$, its *decoration* is the language $\mathrm{Dec}(L)=\{a_1^{r_1}\$_1a_2^{r_2}\$_2\ldots a_m^{r_m}|a_1^{r_1}a_2^{r_2}\ldots a_m^{r_m}\in L\}$. We call decorations of bounded languages *decorated bounded* and decorations of strictly bounded languages *decorated strictly bounded*. Obviously, $\mathrm{Dec}(L)$ is rationally equivalent to L. Therefore, in what follows we consider the decorated strictly bounded language $\mathrm{NonEmbBal}_D(2,k)=\mathrm{Dec}(\mathrm{NonEmbBal}(2,k))$.

Lemma 2. *Let G be a grammar in Chomsky normal form without useless nonterminals for a decorated strictly bounded language. Let $\tau_L(w)=i$ if $w[0]\in\{a_i,\$_i\}$, and $\tau_R(w)=i$ if $w[-1]\in\{a_i,\$_{i-1}\}$ (both functions are undefined for the empty string). Let $A\vdash_G(u_1,\ldots,u_r)$ and $A\vdash_G(v_1,\ldots,v_r)$. Then, for every j, it holds that*

1. *if* $v_j \neq \varepsilon$ *and* $u_j \neq \varepsilon$, *then* $\tau_L(u_j) = \tau_L(v_j)$ *and* $\tau_R(u_j) = \tau_R(v_j)$;
2. *if* $u_j = \varepsilon$, *then* $v_j = a_i^k$ *for some* i *and* k.

Lemma 3. *If there exists a well-nested* k*-component grammar for* $\mathrm{NonEmbBal}_D(2,k)$ *in Chomsky normal form without useless nonterminals, then its derivations contain only even pumps.*

The next result follows from the definition of well-nestedness by simple geometrical considerations.

Lemma 4. *Let* $\vdash_G A(u_1,\ldots,u_r) \vdash_G S(w_0u_0w_1\ldots u_rw_r)$ *and* $\vdash_G B(u'_1,\ldots,u'_s) \vdash_G S(w'_0u'_0w'_1\ldots u'_sw'_s)$ *be two derivations corresponding to the same derivation tree of the string* $w = w_0u_0w_1\ldots u_rw_r = w'_0u'_0w'_1\ldots u'_sw'_s$. *Then one of the following is the case:*

1. $u_0w_1\ldots w_{r-1}u_r$ *is a substring of* $u'_0w'_1\ldots w'_{s-1}u'_s$.
2. $u'_0w'_1\ldots w'_{s-1}u'_s$ *is a substring of* $u_0w_1\ldots w_{r-1}u_r$.
3. $u_0w_1\ldots w_{r-1}u_r$ *and* $u'_0w'_1\ldots w'_{s-1}u'_s$ *are two disjunct substrings of* w.

Informally speaking, the "continuous spans" of two constituents either are embedded or do not intersect. Now we are ready to prove our main theorem.

Theorem 3. *The language* $L = \mathrm{NonEmbBal}_D(2,k)$ *is not defined by any well-nested* k*-component grammar.*

Proof. Assuming the contrary, let such a grammar exist. Then, by Lemma 1, there exists a number p such that at most $p-1$ letters in every string $w \in L$ are uncovered. For the string $w = a_1^pb_1^pc_1^pd_1^p\ldots a_1^pb_1^pc_1^pd_1^p \in L$, at least one c_1 in this string is covered by some pump D_1. By Lemma 3, this pump must be of the form

$$A(c_1^{m_1}d_1^{n_1},\ldots,c_k^{m_k}d_k^{n_k}) \vdash A(c_1^{m_1+r}d_1^{n_1+r},\ldots,c_k^{m_k+r}d_k^{n_k+r}) \vdash S(w)$$

for some nonterminal A, and natural numbers $m_j, n_j \geqslant 0$ and $r > 0$. By analogous arguments applied to the occurrences of a_1, we obtain another derivation

$$A(a_1^{m'_1}b_1^{n'_1},\ldots,a_k^{m'_k}b_k^{n'_k}) \vdash A(a_1^{m'_1+r}b_1^{n'_1+r},\ldots,a_k^{m'_k+r}b_k^{n'_k+r}) \vdash S(w).$$

However, the continuous spans of these two derivations contradict Lemma 4.

Theorem 4. *The family defined by well-nested* k*-component grammars is not closed under the cyclic shift.*

6 Conclusion

This paper has settled the closure under the cyclic shift for both general and well-nested multi-component grammars, as well as pointed out an interesting

difference between these two grammar families. This contributes to the general knowledge on multi-component grammars.

This result has an interesting consequence: since the identity language of any group is closed under cyclic shift, and rational transformations preserve this closure property, no group identity language can be a rational generator of well-nested k-component grammars, for any $k \geqslant 2$. This is not the case for $k = 1$, where the Chomsky-Schützenberger theorem states that any such language can be obtained from the language D_2, that includes the words equal to 1 in a free group with two generators, by a composition of intersection with regular language and a homomorphism.

References

1. Clark, A., Yoshinaka, R.: An algebraic approach to multiple context-free grammars. In: Asher, N., Soloviev, S. (eds.) LACL 2014. LNCS, vol. 8535, pp. 57–69. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43742-1_5
2. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Adison-Wesley, Reading (1979)
3. Jirásková, G., Okhotin, A.: State complexity of cyclic shift. RAIRO-Theoret. Inform. Appl. **42**(2), 335–360 (2008)
4. Joshi, A.K., Levy, L.S., Takahashi, M.: Tree adjunct grammars. J. Comput. Syst. Sci. **10**(1), 136–163 (1975)
5. Kanazawa, M.: The pumping lemma for well-nested multiple context-free languages. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 312–325. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02737-6_25
6. Kanazawa, M.: Ogden's lemma, multiple context-free grammars, and the control language hierarchy. Inf. Comput. (2019)
7. Kanazawa, M., Kobele, G.M., Michaelis, J., Salvati, S., Yoshinaka, R.: The failure of the strong pumping lemma for multiple context-free languages. Theory Comput. Syst. **55**(1), 250–278 (2014)
8. Kanazawa, M., Salvati, S.: Mix is not a tree-adjoining language. In: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 666–674 (2012)
9. Maslov, A.N.: Estimates of the number of states of finite automata. Dokl. Akad. Nauk **194**(6), 1266–1268 (1970)
10. Maslov, A.N.: Cyclic shift operation for languages. Problemy Peredachi Informatsii **9**(4), 81–87 (1973)
11. Okhotin, A.: Conjunctive and Boolean grammars: the true general case of the context-free grammars. Comput. Sci. Rev. **9**, 27–59 (2013)
12. Oshiba, T.: Closure property of family of context-free languages under cyclic shift operation. Electron. Commun. Jpn **55**(4), 119–122 (1972)
13. Pollard, C.J.: Generalized phrase structure grammars, head grammars, and natural language. Ph.D. dissertation, Stanford University (1984)
14. Seki, H., Matsumura, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theoret. Comput. Sci. **88**(2), 191–229 (1991)
15. Sorokin, A.: Normal forms for multiple context-free languages and displacement Lambek grammars. In: Artemov, S., Nerode, A. (eds.) LFCS 2013. LNCS, vol. 7734, pp. 319–334. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35722-0_23

16. Sorokin, A.: Ogden property for linear displacement context-free grammars. In: Artemov, S., Nerode, A. (eds.) LFCS 2016. LNCS, vol. 9537, pp. 376–391. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-27683-0_26
17. Terrier, V.: Closure properties of cellular automata. Theoret. Comput. Sci. **352**(1–3), 97–107 (2006)
18. Vijay-Shanker, K., Weir, D.J., Joshi, A.K.: Characterizing structural descriptions produced by various grammatical formalisms. In: Proceedings of the 25th Annual Meeting on Association for Computational Linguistics, pp. 104–111. Association for Computational Linguistics (1987)
19. Yoshinaka, R., Kaji, Y., Seki, H.: Chomsky-Schützenberger-type characterization of multiple context-free languages. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 596–607. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13089-2_50