RAIRO-Theor. Inf. Appl. 52 (2018) 219–234
https://doi.org/10.1051/ita/2018016

RAIRO - Theoretical Informatics and Applications
www.rairo-ita.org

REGULAR AND LINEAR PERMUTATION LANGUAGES

Grzegorz Madejski*

Abstract. A permutation rule is a non-context-free rule whose both sides contain the same multiset of symbols with at least one non-terminal. This rule does not add or substitute any symbols in the sentential form, but can be used to change the order of neighbouring symbols. In this paper, we consider regular and linear grammars extended with permutation rules. It is established that the generative power of these grammars relies not only on the length of the permutation rules, but also whether we allow or forbid the usage of erasing rules. This is quite surprising, since there is only one non-terminal in sentential forms of derivations for regular or linear grammars. Some decidability problems and closure properties of the generated families of languages are investigated. We also show a link to a similar model which mixes the symbols: grammars with jumping derivation mode.

Mathematics Subject Classification. 68Q42, 68Q85.

Accepted November 21, 2018.

1. Introduction

Permutation languages ($\mathcal{PL}$) were introduced in [12, 13] to fill in the gap between context-free ($\mathcal{CFL}$) and context-sensitive ($\mathcal{CSL}$) language classes. Indeed, many phenomena of the world are described by strings of information that cannot be parsed by context-free grammars. On the other hand, in many cases context-sensitive languages are too broad and complex to be of practical use. Therefore, families of mildly context-sensitive languages are often investigated and $\mathcal{PL}$ was an interesting candidate for such an attempt. In [12, 13], permutation grammars were defined by extending context-free grammars with interchange rules of the form $XY \to YX$ where X, Y are non-terminal symbols. This class was shown to be strictly between $\mathcal{CFL}$ and $\mathcal{CSL}$.

In [14], the definition of permutation rules was extended to arbitrarily many neighbouring symbols. Generative power of such grammars was analysed, as well as closure properties of the generated language classes. An interchange lemma was also presented and it was used to prove that if we allow to permute three neighbouring symbols, we obtain a class of languages strictly greater than when only permutations of length 2 are allowed. In [7], an infinite hierarchy with respect to the length (span) of the rules was established.

It was noted, in [13], that regular grammars extended with permutation rules would be an interesting class to consider. Such languages have only one non-terminal symbol in a derivation and permutation rules must contain this symbol. Some basic properties of such grammars were considered in [15]. It was shown that some non-context-free languages can be generated in this way. Linear and regular permutation languages were also

Keywords and phrases: Permutation languages, interchange rules, context-free grammars, linear grammars, regular grammars, closure properties, decidability, generative power.

Institute of Informatics, Faculty of Mathematics, Physics, and Informatics, University of Gdańsk, 80-308 Gdańsk, Poland.

* Corresponding author: gmadejsk@inf.ug.edu.pl

studied in [8] with respect to their membership problem complexity. It was shown that membership problem for these classes is, in most cases, NP-hard.

As there was no thorough research done on these classes, we took on this task and studied them from different perspectives. The results were presented at NCMA 2016 in Debrecen [10]. This version of the article is slightly longer than the proceedings version. It includes some proof corrections and also provides a connection between this model of grammars and jumping grammars. Actually, some of the results on jumping grammars included here were also presented as a short paper at NCMA 2016 by the author [9].

In Section 2, we provide all the necessary definitions and illustrate some of them with examples. In Section 3, the results of this paper are presented. We study the generative power of linear and regular permutation grammars (Sect. 3.1) and we establish an infinite hierarchy of languages with respect to the length of permutation rules (Sect. 3.2). In Section 3.3, we investigate some closure properties and in Section 3.4, we present some decidability results. Next, in Section 3.5, we recall jumping grammars and show that this model bears some similarity to our permutation-based model. We conclude the paper with Section 4, where we point out some open problems and prospects for future research.

Although this paper presents a theoretical oriented study, we present some remarks and suggestions for possible future applications. Permutation languages can describe phenomena where pieces of information can permute or the order in which they are parsed is not important. Such a free-order behaviour can be found in the research area of many disciplines.

- Relation with concurrent programming: permutation languages are similar to the family of partially commutative context-free languages (PCCFLs) which were presented in [2]. PCCFLs are generated by context-free grammars in Greibach normal form which use the left-most derivation for context-free rules and are additionally equipped with a binary relation that allows swapping of two neighbouring non-terminal symbols in the sentential form. This relation, called independence relation, can be simulated by permutation rules of the form $XY \to YX$. PCCFLs induce a class of partially commutative context-free processes (BPC) which is an alternative for other abstract models of concurrent and recursive programs, such as Process Algebra. The study of permutation languages could provide more insight into the research of PCCFLs and other models using the reordering of symbols.
- Natural language processing: as it was noted in some previous papers on permutation languages (see [12, 13]), permutation grammars can be used to parse natural languages with a relatively free word order, like Hungarian, Finnish or Polish. Indeed, the words in a Polish sentence "Jan naprawia komputery" (Jan repairs computers) could be permuted. The sentences "Komputery naprawia Jan", "Jan komputery naprawia" and "Komputery Jan naprawia" are all correct.
- Multisets: multisets, which are used for some biologically inspired computation models, can be viewed as strings for which the order of letters is not important, *i.e.* can be arbitrarily reordered using permutations.

2. Preliminaries

We assume the reader is familiar with the basic concepts of the formal language theory (see [4]). Let $T = \{t_1, t_2, \ldots, t_n\}$ be the set of terminal symbols (or alphabet) and $w \in T^*$ a word. The length of w is denoted by $|w|$ and the number of occurrences of letter t in w by $|w|_t$. The Parikh vector is a function $\Psi : T^* \to \mathbb{N}^{|T|}$ given by the formula $\Psi(w) = (|w|_{t_1}, \ldots, |w|_{t_n})$. For example, for $T = \{a, b, c\}$, $\Psi(ccabac) = (2, 1, 3)$. In many cases, we will extend this definition to $\Psi : (T \cup N)^* \to \mathbb{N}^{|T \cup N|}$, where N is a set of non-terminal symbols (to construct the vector, first, we order lexicographically the letters, then the non-terminals). For two words $u, v \in T^*$, the shuffle operation $\odot$ is defined as follows $u \odot v = \{u_1 v_1 u_2 v_2 \cdots u_k, v_k : v_1, \ldots v_k, u_1, \ldots, u_k \in T^*, u_1 u_2 \cdots u_k = u, v_1 v_2 \cdots v_k = v\}$.

Grammars are quadruples $G = (N, T, P, S)$ where N is the set of non-terminal symbols, T is the set of terminal symbols (alphabet), $P \subseteq (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$ – a finite relation, S – the start symbol. The elements of P are called rules and are denoted by $x \to y$ instead of (x, y). For brevity, we write $x \leftrightarrow y$ if

$x \to y, y \to x \in P$. The derivation step relation $\Longrightarrow$ is defined in a standard way and the language generated by G is the set $L(G) = \{w \in T^* : S \Longrightarrow^* w\}$, where $\Longrightarrow^*$ is the transitive-reflexive closure of $\Longrightarrow$.

Definition 2.1. We say that a grammar $G = (N, T, P, S)$ is *a permutation grammar* (PG, for short) if all the non-context-free rules are of the form $\alpha \to \beta$ where $\alpha, \beta \in (N \cup T)^* N (N \cup T)^*$, $\Psi(\alpha) = \Psi(\beta)$ (we call them permutation rules). The languages generated by PGs are called *permutation languages* and denoted as $\mathcal{PL}$.

If a permutation rule was used in a derivation step, we may mark it as $\underset{\text{Perm}}{\Longrightarrow}$. Similarly, we can mark the usage of context-free, linear or regular rules by $\underset{\text{CF}}{\Longrightarrow}$, $\underset{\text{Lin}}{\Longrightarrow}$, $\underset{\text{Reg}}{\Longrightarrow}$, respectively.

It has been noted above that the context-free rules in a permutation grammar can be restricted to linear or regular ones to form some subclasses. We give a formal definition.

Definition 2.2. Let $G = (N, T, P, S)$ be a PG and let $X, Y \in N$, $a \in T$, $u, v, z \in T^*$, $w \in T^+$. A rule is:

- *non-erasing regular* if it is of the form $X \to aY \mid Y \mid a$,
- *regular* if it is of the form $X \to aY \mid Y \mid a \mid \lambda$,
- *non-erasing right-linear* if it is of the form $X \to uY \mid w$,
- *right-linear* if it is of the form $X \to uY \mid v$,
- *non-erasing linear* if it is of the form $X \to uYv \mid w$.
- *linear* if it is of the form $X \to uYv \mid z$.

Grammar	Allowed type of CF rules	Generated language family
Non-erasing regular permutation grammar (neRPG)	Non-erasing regular*	$\mathcal{RPL}$
Regular permutation grammar (RPG)	Regular	$\mathcal{RPL}_\lambda$
Non-erasing right-linear permutation grammar (neRLPG)	Non-erasing right-linear*	$\mathcal{RLPL}$
Right-linear permutation grammar (RLPG)	Right-linear	$\mathcal{RLPL}_\lambda$
Non-erasing linear permutation grammar (neLPG)	Non-erasing linear*	$\mathcal{LPL}$
Linear permutation grammar (LPG)	Linear	$\mathcal{LPL}_\lambda$

*Erasing rule $S \to \lambda$ is allowed if S is not in the right-hand side of any rule.

To complete the notation, we denote the family of context-free, linear and regular languages by $\mathcal{CFL}$, $\mathcal{LL}$, $\mathcal{RL}$, respectively.

It is quite interesting to see that the difference between RPGs, LPGs and their non-erasing counterparts seems small. The derivations in these families of grammars may only differ in the last step of the derivation, which is done by using an erasing or non-erasing rule.

Definition 2.3. Let $G = (N, T, P_1 \cup P_2, S)$ be a PG generating a language L, where P_1 contains only context-free rules and P_2 only permutation rules. The language generated by the grammar $G' = (N, T, P_1, S)$ is called the basis language of L with respect to G.

Let us illustrate the above definitions with the following example.

Example 2.4. Consider a neRPG $G = (\{S, X, Y, Z\}, \{a, b, c\}, P, S)$ where P contains the following rules: $S \to \lambda \mid aX$; $X \to bY$; $Y \to cZ \mid c$; $Z \to aX, Xa \leftrightarrow aX,\ Xb \leftrightarrow bX,\ Xc \leftrightarrow cX, Ya \leftrightarrow aY,\ Yb \leftrightarrow bY,\ Yc \leftrightarrow cY,\ Za \leftrightarrow aZ,\ Zb \leftrightarrow bZ,\ Zc \leftrightarrow cZ$.

We see that the basis language is $(abc)^*$. The permutation rules allow the non-terminal symbol to change its position in the sentential form, *e.g.*

$$S \Rightarrow aX \Rightarrow abY \Rightarrow aYb \Rightarrow Yab \Rightarrow cab.$$

Therefore, it is easy to see that the generated language is

$$L = \{w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c\}.$$

Notice that with a slight change in the rules, *e.g.* $Y \to Zc$ instead of $Y \to cZ$, we obtain neLPG with a different basis language: $\{(ab)^n c^n : n \geq 0\}$, but also generating L.

Next, we define the notion of span.

Definition 2.5. A rule $x \to y$ is of span m if $|x| \leq m$ and $|y| \leq m$. A grammar is of span m if all its rules are of span m. We may also say that a language generated by such grammar is of span m.

The classes of languages of span m are denoted by adding the span in the brackets to the abbreviation of the name of the class, *e.g.* $\mathcal{RPL}(m)$, $\mathcal{LPL}_\lambda(m)$. We see that the language L from Example 2.4 is in $\mathcal{RPL}(2)$.

The final definition establishes the notion of closure of a grammar with respect to permutation rules.

Definition 2.6. A permutation grammar $G = (N, T, P, S)$ is m-permutation-closed if for all possible $\alpha, \beta \in (N \cup T)^* N (N \cup T)^*$, such that $\Psi(\alpha) = \Psi(\beta)$ and $2 \leq |\alpha|, |\beta| \leq m$, we have $\alpha \leftrightarrow \beta \in P$.

A family of languages generated by m-permutation-closed grammars will be denoted using an additional superscript $m - pc$ attached the symbol of the family, *e.g.* $\mathcal{RPL}^{m-pc}$, $\mathcal{LPL}_\lambda^{m-pc}$.

3. Results

3.1. Generative power

We start this section by showing that right-linear permutation languages are the same as regular permutation languages. This is analogous to the standard classes of languages, without permutation rules.

Proposition 3.1. $\mathcal{RLPL} = \mathcal{RPL}$, $\mathcal{RLPL}_\lambda = \mathcal{RPL}_\lambda$.

Proof. The proof is similar to the one for standard classes. Every right-linear rule can be transformed into a number of regular rules using additional symbols. □

From now on, we do not consider the classes of right-linear permutation languages: $\mathcal{RLPL}$, $\mathcal{RLPL}_\lambda$, as they coincide with the regular ones.

Lemma 3.2. *Let $L \in \mathcal{LPL}_\lambda$ (or $L \in \mathcal{RPL}_\lambda$). If $w \in L$ and the statement $(\Psi(u) = \Psi(w) \Rightarrow u = w)$ is satisfied for any $u \in L$, then $L \in \mathcal{LL}$ (or $L \in \mathcal{RL}$, correspondingly).*

Proof. If $w \in L$, then it is derived using rules of some linear permutation grammar. If permutations were used, we remove them from the derivation. We then get a word $u \in L$. By the assumptions of the lemma, we have $u = w$. We conclude that permutations are not needed in any derivation, so $L \in \mathcal{LL}$. □

We see that the lemma characterises languages generated by grammars which always produce the same basis language.

The lemma is a useful tool for showing that certain languages are not in $\mathcal{LPL}_\lambda$ or $\mathcal{RPL}_\lambda$. Indeed, by showing that a language is linear or regular and applying a standard pumping lemma for regular or linear languages (see [4, 5]), we reach a contradiction.

Corollary 3.3. $L_1 = \{a^n b^n : n \geq 1\} \notin \mathcal{RPL}_\lambda$, $L_2 = \{a^n b^n c^k d^k : n, k \geq 1\} \notin \mathcal{LPL}_\lambda$.

Another language important for the study of generative power of the grammar classes is presented in the example below.

Example 3.4. Consider an RPG $G = (\{S, X, Y, Z\}, \{a, b, c, d\}, P, S)$ where P contains the rules:

$$S \to abX \mid abY, \quad X \to abX \mid cdY, \quad Y \to cdY \mid Z, \quad Z \to \lambda$$

$$tZ \leftrightarrow Zt, \text{ where } t \in \{a, b, c, d\}, \quad aZb \leftrightarrow bZa, \quad cZd \leftrightarrow dZc.$$

An example derivation:

$$S \Rightarrow abX \Rightarrow ababX \Rightarrow ababcdY \Rightarrow ababcdZ \Rightarrow \ldots \Rightarrow$$

$$\Rightarrow abZabcd \Rightarrow aaZbbcd \Rightarrow aabbcd.$$

We see that $L(G) = \{w \in \{a, b\}^+ : |w|_a = |w|_b\} \cdot \{w \in \{c, d\}^+ : |w|_c = |w|_d\}$.

The language from Example 3.4 will be denoted as L_3 throughout the paper.

Before we study L_3, we present a useful lemma. The lemma characterises RPGs with permutations that are of the form $uX \to vX$, where $X \in N$, $u, v \in T^+$, $\Psi(u) = \Psi(v)$. We shall call them right-symbol permutations.

Lemma 3.5. *Let $G = (N, T, P, S)$ be a RPG of span m with right-symbol permutation rules. Then:*

- *All sentential forms can be described by an expression uX or u where $u \in T^*$, $X \in N$.*
- *Let v_1v_2X be a sentential form with $|v_2| = m - 1$. If $v_1v_2X \Rightarrow^* w$, then $w = v_1z$ where $z \in T^*$.*

Proof. Regular rules and right-symbol permutation rules have the non-terminal symbol always at the rightmost end. Therefore, sentential forms can have the non-terminal symbol only at the last position.

If v_1v_2X is a sentential form such that $v_1, v_2 \in T^*$, $|v_2| = m - 1$, $X \in N$, then indeed all the words derived from it are of the form v_1z, $z \in T^*$. This follows from the fact that the permutations are of span m, so they can mix letters in v_2, but not in v_1. □

We use the above results in the proof of the next lemma.

Lemma 3.6. *Let $G = (N, \{a, b\}, P, S)$ be an RPG of span m such that $L = L(G) \subseteq \{w \in \{a, b\}^+ : |w|_a = |w|_b\}$ and $a^{m|N|}b^{m|N|} \in L$. Then, P contains at least one permutation rule which is not a right-symbol permutation.*

Proof. Let $T = \{a, b\}$. We assume that P has only regular rules and right-symbol permutations. We consider a word $w = a^{m|N|}b^{m|N|}$ which is definitely in L. By Lemma 3.5, all sentential forms in the derivation of w have the non-terminal symbol at the rightmost end. By Lemma 3.5, we also know that

$$S \Longrightarrow^* a^{m|N|}vX \Longrightarrow^* w$$

where $X \in N$, $v = b^{m-1}$, $vX \Rightarrow^* b^{m|N|}$. We remove the right-symbol permutations from the whole derivation. Then, it is of the form:

$$S \underset{Reg}{\Longrightarrow^*} u_1u_2 \cdots u_mv'X \underset{Reg}{\Longrightarrow^*} w'$$

where $u_1, u_2, \ldots, u_m$ are words of length $|N|$, v' is a word of length $m - 1$ and $u_1, \ldots, u_m, v'$ altogether consist of $m|N|$ letters a and $m - 1$ letters b. Since the number of b's is $m - 1$ and the number of u-words is m, it means

that one word, say u_i where $1 \leq i \leq m$, consists solely of letters a.

$$S \underset{\text{Reg}}{\Longrightarrow}^* u_1u_2\cdots u_{i-1}Y \underset{\text{Only } a\text{'s}}{\Longrightarrow}^* u_1u_2\cdots u_{i-1}u_iZ \underset{\text{Reg}}{\Longrightarrow}^* u_1u_2\cdots u_mv'X \underset{\text{Reg}}{\Longrightarrow}^* w'$$

for some $Y, Z \in N$. To generate u_i we need at least $|N|$ steps, each one adding a letter a. So the part of derivation $u_1u_2\cdots u_{i-1}Y \underset{\text{Only } a\text{'s}}{\Longrightarrow}^* u_1u_2\cdots u_{i-1}u_iZ$ consists of $|N|+1$ sentential forms. Since there are $|N|$ non-terminal symbols, at least two sentential forms contain the same non-terminal symbol. The subword derived between them can be pumped an arbitrary number of times, producing additional letters a. After pumping, we get on output a word $w'' \notin L$, but derived using the rules of G. A contradiction is due to the false assumption that G has only regular rules and right-symbol permutations. □

We now go back to studying L_3.

Proposition 3.7. *$L_3 \notin \mathcal{LPL}$.*

Proof. We assume that $L_3 \in \mathcal{LPL}$. Let $G = (N, \{a, b, c, d\}, P, S)$ be an neLPG of span m generating L_3 and $n = m|N|$. We consider a word $w = a^nb^nc^nd^n \in L_3$ and its derivation in G.

Firstly, notice that every sentential form must be of the form $(a+b)^*(c+d)^* \odot X$ where $X \in N$. Indeed, if the first part of the word, with letters a and b, was mixed into the second part with c's and d's, then by applying no permutation rules, we would obtain a word not in L_3.

There are two cases to consider: the last rule has at least one letter a or b in the right-hand side or it has at least one letter c or d. We investigate only the second case because the first one is analogous.

If the last used rule is $Y \to utv$ where $u, v \in \{a, b, c, d\}^*$, $t \in \{c, d\}$, then the derivation is of the form

$$S \Longrightarrow \underbrace{\alpha_1 \Longrightarrow \alpha_2 \Longrightarrow \ldots \Longrightarrow \alpha_k}_{\text{of the form } (a+b)^*(c+d)^*X(c+d)^*} \Longrightarrow w$$

where $X \in N$.

We use an erasing homomorphism $h(a) = a$, $h(b) = b$, $h(c) = \lambda$, $h(d) = \lambda$, $h(X) = X$ where $X \in N$, on the left-hand and right-hand sides of the rules of G. Then, we remove all rules that were not used to generate w. We obtain a grammar $G' = (N, \{a, b\}, P', S)$. Let us make some important observations.

(1) The derived word w becomes $h(w) = a^nb^n$ and its derivation under homomorphism looks as follows

$$S \Longrightarrow \underbrace{h(\alpha_1) \Longrightarrow h(\alpha_2) \Longrightarrow \ldots \Longrightarrow h(\alpha_k)}_{\text{of the form } (a+b)^*X} \Longrightarrow a^nb^n.$$

We see that the non-terminal symbol is always on the right-hand side. This leads us to the second observation below.

(2) G' has only regular rules and right-symbol permutation rules.

(3) $L(G') \subseteq \{w \in \{a, b\}^+ : |w|_a = |w|_b\}$, $a^nb^n \in L(G')$ where $n = m|N|$. Indeed, every word of $L(G)$ has to have the same number of a's and b's, and the words of $L(G')$ as well. By observation 1, we see that $h(w) = a^nb^n \in L(G')$.

By Lemma 3.6, we see that it is not possible to derive a^nb^n where $n = m|N|$ in G' when it has only right-symbol permutations. □

The above results lead us to the following theorem.

Theorem 3.8. *$\mathcal{RPL}_\lambda$ and $\mathcal{LPL}$ are incomparable.*

We now show that $\mathcal{RPL}_\lambda \cup \mathcal{LPL}$ are strictly contained within the class $\mathcal{LPL}_\lambda$, by establishing some results on the following language.

Example 3.9. Consider an LPG $G = (\{S, X, Y, Z\}, \{a, b, c, d\}, P, S)$ where P contains the rules $S \to abScd \mid abXcd$, $X \to \lambda$, $tX \leftrightarrow Xt$, where $t \in \{a, b, c, d\}$, $aXb \leftrightarrow bXa$, $cXd \leftrightarrow dXc$.

We see that the basis language is $\{(ab)^n(cd)^n : n > 0\}$. Similarly to the Example 3.4, the symbol X is used to mix letters.

$$S \Rightarrow abScd \Rightarrow ababXcdcd \Rightarrow ababcXdcd \Rightarrow ababdXccd \Rightarrow \ldots \Rightarrow$$

$$\Rightarrow abXabdccd \Rightarrow aaXbdccd \Rightarrow aabbdccd.$$

We see that $L(G) = \{uw : u \in \{a, b\}^+, w \in \{c, d\}^+, |u|_a = |u|_b = |w|_c = |w|_d\}$.

From now on, we denote this language as L_4.

Proposition 3.10. $L_4 \notin \mathcal{LPL}$.

Proof. The proof is analogous to the one of Proposition 3.7. □

Proposition 3.11. $L_4 \notin \mathcal{RPL}_\lambda$.

Proof. Suppose that $L_4 \in \mathcal{RPL}_\lambda$ and it is generated by an RPG G. Let $n > 0$. We consider a word $w = a^n b^n c^n d^n \in L_4$. We remove all permutation rules from its derivation and get a word $w' = uv$, where $u \in \{a, b\}^+$, $v \in \{c, d\}^+$, $|u|_a = |u|_b = |v|_c = |v|_d = n$. All the rules used in its derivation are regular. We use standard pumping lemma for regular languages. We pump the number of letters a and b so that their number is bigger than that of c's and d's: $w_i = u_1 u_2^i u_3 v$ where $i > 0$, $u_1 u_2 u_3 = u$, $u_2 \neq \lambda$. □

We conclude this section with the following theorem.

Theorem 3.12. $\mathcal{RPL}_\lambda \cup \mathcal{LPL} \subsetneq \mathcal{LPL}_\lambda$.

Proof. The inclusion follows from the definition of all classes. The strictness is a consequence of Propositions 3.10, 3.11. □

All the results are summarised in Figure 1.

3.2. Infinite hierarchy

It is interesting to analyse the generative power not only with respect to the context-free rules but also permutations. A simple restriction is to consider only permutation rules of span m. Then, an obvious question arises: are there linear permutation languages of span m that are not of span $m-1$? This problem was partially solved in [7] for the class $\mathcal{PL}$, where it was shown that $\mathcal{PL}(4m-2) \subsetneq \mathcal{PL}(4m-1)$ for $m \geq 1$. Although we now establish a result only for the subclass of $\mathcal{PL}$, it is stronger, as there are no gaps in the hierarchy.

We present now a special kind of lemma, which plays a crucial role in the proof of the hierarchy theorem. It is a variation on the interchange lemma presented in [14]. The idea of the lemma is this: if a word is derived using at least one permutation rule, then by removing the last permutation rule from the derivation, we obtain a different word which is also in the language.

Lemma 3.13 (Interchange lemma for $\mathcal{LPL}_\lambda(m)$)**.** *Let $G = (N, T, P, S)$ be a linear permutation grammar of span m. For an arbitrary word $w \in L(G)$ not in the basis language, there exist words $u, v, u', v' \in T^*$ such that:*

(1) $0 < |uv| < m$, $\Psi(uv) = \Psi(u'v')$, $uv \neq u'v'$,
(2) $w = xuyvz$, *for some* $x, y, z \in T^*$,

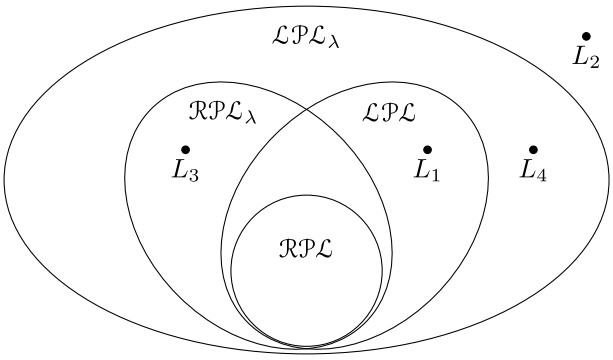

FIGURE 1. The studied classes with the marked languages: $L_1 = \{a^n b^n : n \geq 1\}$ (Cor. 3.3), $L_2 = \{a^n b^n c^k d^k : n, k \geq 1\}$ (Cor. 3.3), $L_3 = \{w \in \{a,b\}^+ : |w|_a = |w|_b\} \cdot \{w \in \{c,d\}^+ : |w|_c = |w|_d\}$ (Ex. 3.4, Prop. 3.7) and $L_4 = \{uw : u \in \{a,b\}^+, w \in \{c,d\}^+, |u|_a = |u|_b = |w|_c = |w|_d\}$ (Ex. 3.9, Prop. 3.10, 3.11).

(3) $w' = xu'yv'z \in L$.

Proof. We see that if a word $w \in L(G)$ is not in the basis language, at least one permutation rule is used in the derivation. We consider the last permutation rule, which is of the form $u'Xv' \to uXv$, for some words $u, v, u', v' \in T^*$, $0 < |uv| < m$, $\Psi(uv) = \Psi(u'v')$ and a non-terminal X. The derivation is $S \Longrightarrow^* xu'Xv'z \underset{\text{Perm}}{\Longrightarrow} xuXvz \underset{\text{Lin}}{\Longrightarrow}^* xuyvz = w$. If we omit the last permutation rule, we get: $S \Longrightarrow^* xu'Xv'z \underset{\text{Lin}}{\Longrightarrow}^* xu'yv'z$. □

The lemma is used to disprove that a language belongs to $\mathcal{LPL}_\lambda(m)$, the same way as classical pumping lemmas. We now consider an important language, which is used to establish the hierarchy result.

Example 3.14. Let $m > 0$. Consider an LPG $G = (\{S, X, Y\}, \{a, b, c\}, P, S)$ where P contains the rules $S \to a^m X$, $X \to b^m Y$, $Y \to c^m S | \ c^m$, $Sa^m \leftrightarrow a^m S$, $Sb^m \leftrightarrow b^m S$, $Sc^m \leftrightarrow c^m S$, $Xa^m \leftrightarrow a^m X$, $Xb^m \leftrightarrow b^m X$, $Xc^m \leftrightarrow c^m X$, $Ya^m \leftrightarrow a^m Y$, $Yb^m \leftrightarrow b^m Y$, $Yc^m \leftrightarrow c^m Y$.

The right-linear rules produce the same number of m-segments a^m, b^m, c^m while the permutation rules allow them to be mixed in any order. Note, that the construction of the rules forbids putting one m-segment in between letters of another m-segment. An example derivation:

$$S \underset{\text{Lin}}{\Longrightarrow} a^m X \underset{\text{Perm}}{\Longrightarrow} Xa^m \underset{\text{Lin}}{\Longrightarrow} b^m Y a^m \underset{\text{Lin}}{\Longrightarrow} b^m c^m a^m.$$

Clearly, $L_{m+1} = L(G) = \{w \in (a^m + b^m + c^m)^+ : |w|_a = |w|_b = |w|_c\} \in \mathcal{LPL}(m+1)$.

Proposition 3.15. $L_{m+1} \notin \mathcal{LPL}_\lambda(m)$.

Proof. Suppose $L_{m+1} \in \mathcal{LPL}_\lambda(m)$. Then, there exists an LPG $G = (N, \{a, b, c\}, P, S)$ of span m, such that $L_{m+1} = L(G)$. Let $n = |N|$ and $w = a^{mn} b^{mn} c^{mn} \in L_{m+1}$ (we can use pumping lemma for linear languages to prove that w does not lie in the basis language). We see that the word uv from the Interchange Lemma 3.13 is of length at most $m - 1$. Then, $|u| + |v| \leq m - 1$. Each of the words u, v can lie inside an m-segment, but not containing it or on the boarder of two m-segments.

Let us consider one of a few possible cases where u lies on the border of an m-segment of a's and b's and v lies the border of an m-segment of b's and c's. First part (including y):

$$\overbrace{a^{m(n-1)}(aaa\cdots}^{x}\overbrace{\underbrace{aa\cdots aaa}_{u_1})(\underbrace{bbb\cdots bb}_{u_2}}^{u}\overbrace{\cdots bbb)b^{m(n-2)}(bbb\cdots}^{y}.$$

Second part (also including y):

$$\overbrace{\cdots bbb)b^{m(n-2)}(bbb\cdots}^{y}\overbrace{\underbrace{bb\cdots bbb}_{v_1})(\underbrace{ccc\cdots cc}_{v_2}}^{v}\overbrace{\cdots ccc)c^{m(n-1)}}^{z}.$$

Since $u = u_1u_2$, $v = v_1v_2$, we see $|u_1| + |u_2| + |v_1| + |v_2| \leq m - 1$. We observe that u_1 must consist of letters a. If swapped with other letters, the last m-segment of a's would be destroyed. Similarly, v_2 must consist of c's. What is left, are letters b in u_2 and v_1. Mixing these letters with each other does not produce anything other than what was before permuting. We come to a conclusion that $uv = u'v'$ from Interchange Lemma 3.13. In other cases, we would obtain a similar result which contradicts our assumption that $L_{m+1} \in \mathcal{LPL}_\lambda(m)$. □

Theorem 3.16. *$\mathcal{LPL}_\lambda(m) \subsetneq \mathcal{LPL}_\lambda(m+1)$ for any $m \geq 2$.*

Proof. The inclusion follows from the definition of span and the strictness from Proposition 3.15. □

3.3. Closure properties

In this subsection, we study some closure properties of linear and regular permutation languages. All results are presented in Table 1. The results are not exhaustive. There are some open problems presented at the end of the section and in conclusion of this paper.

Theorem 3.17. *The classes $\mathcal{RPL}$, $\mathcal{LPL}$, $\mathcal{RPL}_\lambda$, $\mathcal{LPL}_\lambda$ are closed under union, but are not closed under intersection, intersection with a regular language and complement.*

Proof. Closure under union is proven the same way as for standard classes of the Chomsky hierarchy. For the two regular permutation languages $L = \{w \in \{a,b,c\}^+ : |w|_a = |w|_b = |w|_c\}$ and $K = a^*b^*c^*$, the language $L \cap K = \{a^nb^nc^n : n \geq 1\}$ is not in $\mathcal{LPL}_\lambda$. The lack of closure under complement follows from the fact that $L \cap K = (L^c \cup K^c)^c$. □

Theorem 3.18. *The classes $\mathcal{RPL}$, $\mathcal{LPL}$, $\mathcal{LPL}_\lambda$ are not closed under concatenation.*

Proof. Consider languages $L = \{w \in \{a,b\}^+ : |w|_a = |w|_b\}$ and $K = \{w \in \{c,d\}^+ : |w|_c = |w|_d\}$, both of which are in $\mathcal{RPL}$. We have already proven that $L_3 = LK \notin \mathcal{RPL}$ (Prop. 3.7). We proceed similarly for the linear permutation languages. If $L = \{a^nb^n : n \geq 1\}$ and $K = \{c^nd^n : n \geq 1\}$, then $L_2 = LK \notin \mathcal{LPL}_\lambda$. □

Theorem 3.19. *The classes $\mathcal{LPL}$, $\mathcal{LPL}_\lambda$ are not closed under Kleene star.*

Proof. We take the languages $L = \$\{a^nb^n : n \geq 1\} \in \mathcal{LPL}$ and $K = \$\{c^nd^n : n \geq 1\} \in \mathcal{LPL}$ and construct a language $L_* = (L + K)^*$. Notice that L and K have a \$-sign concatenated at the beginning of each word. These signs serve as counters *i.e.* by looking at the number of \$, we know how many times the concatenation by Kleene star occurred.

We assume that $L_* \in \mathcal{LPL}_\lambda$. For $T = \{a,b,c,d\}$, let $G = (N,T,P,S)$ be an LPG of span m such that $L_* = L(G)$. Let $n \geq m|N|$. The word $w = \$a^nb^n\$c^nd^n \in L_*$, but w is not in the basis language. Indeed, words of this form cannot be in any linear language which is easy to prove using a pumping lemma for linear languages (see [5]). By the Interchange Lemma 3.13, there exist words $u,v,u',v',x,y,z \in T^*$ such that $0 < |uv| < m$, $\Psi(uv) = \Psi(u'v')$, $uv \neq u'v'$, $w = xuyvz$ and $w' = xu'yv'z \in L_*$. The only possible w' such that $\Psi(w) = \Psi(w')$ is the word $\$c^nd^n\$a^nb^n$, but it would require $|uv| > m$, which leads to a contradiction. □

We believe that $\mathcal{RPL}$ is also not closed under Kleene star. The idea would be to use languages $L = \$\{w \in \{a,b\}^+ : |w|_a = |w|_b\} \in \mathcal{RPL}$ and $K = \$\{w \in \{c,d\}^+ : |w|_c = |w|_d\} \in \mathcal{RPL}$ and, similarly to the linear permutation languages, to consider a language $L_* = (L + K)^*$. The proof that $L_* \notin \mathcal{RPL}$ would be probably similar to the one of Proposition 3.7. We leave it, however, as an open problem.

It is also an open problem to establish closure results under concatenation and Kleene star for the class $\mathcal{RPL}_\lambda$. This probably requires some additional techniques not presented in this paper.

TABLE 1. Summary of closure properties.

	$\mathcal{RPL}$	$\mathcal{RPL}_\lambda$	$\mathcal{LPL}$	$\mathcal{LPL}_\lambda$
Union	Yes	Yes	Yes	Yes
Intersection	No	No	No	No
Intersection with Reg.	No	No	No	No
Complement	No	No	No	No
Concatenation	No	?	No	No
Kleene star	?	?	No	No
Commutative closure	Yes	Yes	Yes	Yes

TABLE 2. Decidable (D) and undecidable (UD) results for the investigated classes.

	$\mathcal{RPL}$	$\mathcal{RPL}_\lambda$	$\mathcal{LPL}$	$\mathcal{LPL}_\lambda$
Membership	D	D	D	D
Emptiness	D	D	D	D
Disjointness	UD	UD	UD	UD

The closure under letter-mixing operations such as shuffle and shuffle closure would be interesting to study, but we also leave it as an open problem. Instead, we establish a result for the commutative closure operation.

Theorem 3.20. *The classes $\mathcal{RPL}$, $\mathcal{LPL}$, $\mathcal{RPL}_\lambda$, $\mathcal{LPL}_\lambda$ are closed under commutative closure.*

Proof. We can assume that the regular or linear rules of the given grammar generate only one letter. We modify the grammar by adding all possible permutation rules of span 2. They allow the non-terminal symbol to freely move to any position in the sentential form. If a word w is in the language generated by the modified grammar, then every word u satisfying the condition $\Psi(u) = \Psi(w)$ is also in this language. □

3.4. Decidability results

We now present some decidability results (see Tab. 2). We analyse only three decision problems and leave other for future study.

First, we study the membership problem which has a grammar G and a word w on input, and answers the question "$w \in L(G)$?".

Theorem 3.21. *The membership problem for $\mathcal{RPL}$, $\mathcal{RPL}_\lambda$, $\mathcal{LPL}$, $\mathcal{LPL}_\lambda$ is decidable.*

Proof. It is easy to see that all types of grammars considered in this paper can be simulated by a linear bounded automaton and the membership problem for context-sensitive languages is known to be decidable. □

We move to the next problem. The disjointness problem has two grammars G_1, G_2 on input and answers the question "$L(G_1) \cap L(G_2) = \emptyset$?". To prove that this problem is undecidable, we use the Post correspondence problem (PCP).

Let $\alpha_1, \ldots, \alpha_k \in \{a, b\}^*$, $\beta_1, \ldots, \beta_k \in \{a, b\}^*$, for some $k > 0$, be an instance of PCP. We construct grammars over alphabet $T = \{a, b, 0, 1, \$\}$.

Let $G_\alpha = (\{S, X\}, T, P, S)$ be an neRPG with P consisting of the following rules:

$$S \to \alpha_1 101 S \mid \alpha_2 1001 S \mid \ \ldots \ \mid \alpha_{k-1} 10^{k-1} 1 S \mid \alpha_k 10^k 1 S \mid X,$$

$$X 10^i 1 \leftrightarrow 10^i 1 X, X\alpha_i \leftrightarrow \alpha_i X \text{ where } 1 \leq i \leq k,$$

$$10^i 1\alpha_j X 10^l 1 \to \alpha_j X 10^i 110^l 1 \text{ where } 1 \le i, j, l \le k,$$

$$X \to \$.$$

Similarly, we construct an neRPG $G_\beta = (\{S, X\}, T, P', S)$ with P' consisting of the following rules:

$$S \to \beta_1 101S \mid \beta_2 1001S \mid \ldots \mid \beta_{k-1}10^{k-1}1S \mid \beta_k 10^k 1S \mid X,$$

$$X10^i 1 \leftrightarrow 10^i 1X, X\beta_i \leftrightarrow \beta_i X \text{ where } 1 \le i \le k,$$

$$10^i 1\beta_j X 10^l 1 \to \beta_j X 10^i 110^l 1 \text{ where } 1 \le i, j, l \le k,$$

$$X \to \$.$$

These grammars generate languages L_α, L_β, correspondingly.

Lemma 3.22. *If PCP has a solution $(i_1, i_2, \ldots, i_n)$ where $i_1, i_2, \ldots, i_n \in \{1, 2, \ldots, k\}$, then a word $w = \alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_n}\$10^{i_1}110^{i_2}1\cdots10^{i_n}1$ is in L_α and L_β.*

Proof. The derivation of w in G_α is as follows. We start off with generating all the alphas and their numbers using right-linear rules.

$$S \Rightarrow \alpha_{i_1}10^{i_1}1S \Rightarrow \ldots \Rightarrow \alpha_{i_1}10^{i_1}1\cdots\alpha_{i_n}10^{in}1X.$$

We then use the permutations to move X in the sentential form and use it to move all numbers (the parts of the form 10^*1) to the right of the word.

$$\alpha_{i_1}10^{i_1}1\alpha_{i_2}X10^{i_2}1\cdots\alpha_{i_n}10^{in}1 \Rightarrow \alpha_{i_1}\alpha_{i_2}X10^{i_1}110^{i_2}1\cdots\alpha_{i_n}10^{in}1 \Rightarrow \ldots \Rightarrow$$

$$\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_n}X10^{i_1}110^{i_2}1\cdots10^{i_n}1.$$

Finally, we use the ending rule $X \to \$$ to get w.

If PCP has a solution, then $\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_n} = \beta_{i_1}\beta_{i_2}\cdots\beta_{i_n}$ and the derivation of w in G_β is analogous. □

Lemma 3.23. *If PCP has no solution, then $L_\alpha \cap L_\beta = \emptyset$.*

Proof. We assume that $L_\alpha \cap L_\beta \neq \emptyset$. Let $w \in L_\alpha \cap L_\beta$. It is easy to see that if a sequence with numbers $10^{i_1}1$, $10^{i_2}1$,..., $10^{i_n}1$ was used in the derivation of w in G_α, then it must also be used in the derivation of w in G_β. Indeed, the form of 10^*1-subwords and a strict fashion in which they are handled (*e.g.* they cannot change their order) does not leave any other option.

We also see that the letters a and b cannot be mixed. So under the homomorphism $h(a) = a, h_1(b) = b, h_1(0) = \lambda, h_1(1) = \lambda, h_1(\$) = \lambda$, we obtain $h(w) = \alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_n}$, $h(w) = \beta_{i_1}\beta_{i_2}\cdots\beta_{i_n}$. Since, the 10^*1-subwords can be moved right in derivation in both grammars, we get $\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_n} = \beta_{i_1}\beta_{i_2}\cdots\beta_{i_n}$, which is a solution to PCP. □

Theorem 3.24. *Disjointness problem for $\mathcal{RPL}$ is undecidable.*

Proof. The proof follows from Lemmas 3.22 and 3.23 and undecidability of PCP. □

3.5. A link to jumping grammars

A different approach to mixing symbols in the sentential form is jumping. Lately, it acquired much attention and was studied in several papers [3, 6, 11].

A jumping derivation mode works in a discontinuous way. By applying a rule $\alpha \to \beta$, we remove α from the sentential form and place β in an arbitrary position in the string.

Definition 3.25. We call $G = (N, T, P, S)$ a jumping grammar if in derivation of words, it uses a jumping mode defined as follows. Let $u, v \in (N \cup T)^*$, $u \notin T^*$. $u \Rightarrow v$ if and only if there exists $x \to y \in P$, $w, z, w', z' \in (N \cup T)^*$ such that $u = wxz, v = w'yz'$ and $wz = w'z'$.

We denote by $\mathcal{JRL}$, $\mathcal{JRLL}$, $\mathcal{JLL}$ and $\mathcal{JCFL}$ languages generated by jumping regular, right-linear, linear and context-free grammars, respectively.

Intuitively, we notice that jumping in an arbitrary position in sentential form is like moving the non-terminal symbol to the specified place by some permutation rules. Indeed, if we allow any permutation rules of length 2, that allow the non-terminal to swap places, then we could simulate the jumping mode. Therefore, we use the formerly defined m-permutation-closed grammars with $m = 2$. This leads us to the following theorem.

Theorem 3.26. $\mathcal{JRL} = \mathcal{RPL}_{\lambda}^{2-pc}$, $\mathcal{JRLL} = \mathcal{RLPL}_{\lambda}^{2-pc}$, $\mathcal{JLL} = \mathcal{LPL}_{\lambda}^{2-pc}$, $\mathcal{JCFL} = \mathcal{PL}^{2-pc}$.

Proof. For all the families of languages presented above, the proof is analogous. We prove that $\mathcal{JLL} = \mathcal{LPL}_{\lambda}^{2-pc}$. If we consider a jumping linear grammar $G = (N, T, P, S)$, then we can construct a 2-permutation-closed linear permutation grammar: $G' = (N, T, P', S)$ where $P' = P \cup \{Xt \leftrightarrow tX : t \in T, X \in N\}$. We can also do it the other way around. Having a 2-permutation-closed grammar G', we remove all permutation rules and change the derivation mode to jumping. We then obtain a linear jumping grammar G. It is always important to remember, that G uses the jumping derivation mode and G' uses the classical derivation mode.

We prove that $w \in L(G) \Leftrightarrow w \in L(G')$.

If $w \in L(G)$, then each step of the derivation in jumping mode can be simulated by a number of permutations. We use the rules $Xt \leftrightarrow tX$ where $t \in T, X \in N$, to move the non-terminal symbol to the position where it jumps, and then use the linear rule to add the terminal symbols. Therefore $w \in L(G')$.

If $w \in L(G')$, then we remove all permutation steps from its derivation. The pruned output is a proper jumping derivation of w in G, so $w \in L(G)$. □

Using a similar technique, we could prove that $\mathcal{JCFL} = \mathcal{PL}^{2-pc}$, but due to space constraints of this paper, we leave it as an open problem.

Having established a connection between permutation languages and jumping languages, we can move to solving an interesting problem. In the previous chapter, we have already studied generative power of the permutation grammars. It would be interesting to prove similar results for jumping grammars.

In an attempt to establish a hierarchy of jumping classes similar to the Chomsky hierarchy, it was already shown in [6] that $\mathcal{JRL} \subsetneq \mathcal{JLL} = \mathcal{JRLL}$. By definition, they are subsets of $\mathcal{JCFL}$, but it was an open problem, whether the inclusion is proper. In this subsection, we formulate a jumping lemma and use it to show that $\mathcal{JRLL} \subsetneq \mathcal{JCFL}$.

For convenience, we show that we can eliminate rules of the form $X \to Y$ (X, Y are non-terminals) from any jumping right-linear grammar. We shall call them unit rules.

Lemma 3.27. *Let $G = (N, T, P, S)$ be a jumping right-linear grammar. There exists a jumping right-linear grammar $G' = (N, T, P', S)$ such that $L(G) = L(G')$ and P' have no unit rules.*

Proof. The construction of grammar G' based on G is done in the same as in the algorithm for eliminating unit rules, used to acquire grammars in Chomsky normal form (see [4]). Despite having a different derivation mode, proving that $L(G) = L(G')$ is also analogous. □

Example 3.28. Let $G = (\{S, X, Y, Z\}, \{a, b, c, d\}, P, S)$ be a jumping right-linear grammar of span $m = 3$, where P contains the rules $S \to aaX,\ X \to aaY,\ Y \to bbZ,\ Z \to ccY,\ Z \to ddd$. We derive a word $w = bbaabbaaccddd$ in G: $S \Rightarrow aaX \Rightarrow aaaaY \Rightarrow \Rightarrow aabbZaa \Rightarrow aabbaaccY \Rightarrow bbZaabbaacc \Rightarrow bbaabbaaccddd = w$.

It is easy to see that if the number of derivation steps is greater than the number of non-terminals, then there are two sentential forms containing the same non-terminal symbol. In our example, we have $n = |\{S, X, Y, Z\}| = 4$. Our derivation of w consists of 7 steps and contains two sentential forms with Y: $aaaaY$, $aabbaaccY$. There are two rules used between them $Y \to bbZ, Z \to ccY$, that added subwords $w_1 = bb$, $w_2 = cc$ to the derived word. Similarly to the pumping lemma for regular languages, we can copy the words w_1, w_2 any number of times. The jumping derivation mode lets us put them in any position in the derived string, for example at the end:

$$S \Rightarrow aaX \Rightarrow aaaaY \Rightarrow aabbZaa \Rightarrow aabbaaccY \Rightarrow aabbaacc\underline{bb}Z \Rightarrow aabbaacc\underline{bbcc}Y$$

$$\Rightarrow bbZaabbaacc\underline{bbcc} \Rightarrow bbaabbaaccddd\underline{bbcc} = ww_1w_2,$$

Pumping for words derived in 4 steps may be impossible, for example:

$$S \Rightarrow aaX \Rightarrow aaaaY \Rightarrow aabbZaa \Rightarrow aabbaaddd.$$

We see that the maximal length of the words generated in 4 steps is $3 \cdot 2 + 3 = 9$. For an arbitrary grammar of span m and n non-terminal symbols, this number is $(n-1)(m-1) + m = n(m-1) + 1$. Words of greater length must contain a loop in derivation.

We now extend the above investigations to any jumping right-linear languages.

Lemma 3.29 (Jumping and pumping lemma for $\mathcal{JRLL}(m)$)**.** *Let $G = (N, T, P, S)$ be a jumping right-linear grammar of span m without unit rules and $|N| = n$. Let $w \in L(G)$, $|w| > r$ where $r = n(m-1) + 1$. Then:*

- *The derivation of w in G consists of at least $n + 1$ steps.*
- *Let $X_0 \to w_1X_1, X_1 \to w_2X_2, \ldots, X_{n-1} \to w_nX_n, X_n \to w_{n+1}$ be the rules used in the last $n + 1$ steps of the derivation of w (in the given order), where $0 < |w_i| < m$ for all $1 \leq i \leq n$ and $0 < |w_{n+1}| \leq m$. Then, there exist indices $1 \leq k \leq l \leq n$ such that for any permutation*

$$\sigma = \begin{pmatrix} k & k+1 & \cdots & l \\ \sigma(k) & \sigma(k+1) & \cdots & \sigma(l) \end{pmatrix}$$

words $w_1 = w_{\sigma(k)}w_{\sigma(k+1)} \cdots w_{\sigma(l)}w$ and $w_2 = ww_{\sigma(k)}w_{\sigma(k+1)} \cdots w_{\sigma(l)}$ are also in $L(G)$.

Proof. The first point of the lemma is easy to prove, see Example 3.28. If there are at least $n + 1$ steps, then let the rules used in the last $n + 1$ steps be the following (in the given order):

$$X_0 \to w_1X_1, X_1 \to w_2X_2, \ldots, X_{n-1} \to w_nX_n, X_n \to w_{n+1}.$$

Among the $n+1$ non-terminal symbols $X_0, \ldots, X_n$ there exist two which are the same. Let $X_{k-1} = X_l$ for some $1 \leq k \leq l \leq n$. Then, the rules $X_{k-1} \to w_kX_k, \ldots, X_{l-1} \to w_lX_{k-1}$ allow us to pump the words $w_k, \ldots, w_l$. Due to the jumping derivation mode, we can put them at the beginning or the end of the word, in any order represented by a permutation σ (see Fig. 2).

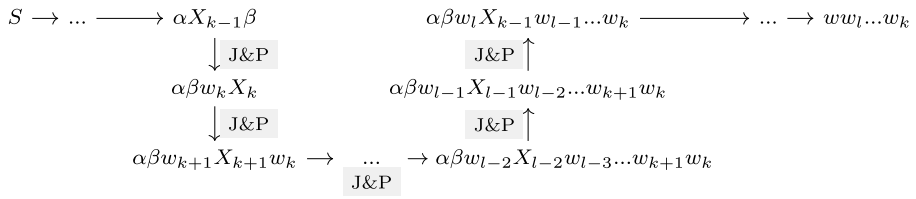

FIGURE 2. An example derivation of a pumped word. The steps where jumping and pumping was used are marked with J&P. A permutation $\sigma(i) = l + k - i$ was used to order the pumped words $w_k, \ldots, w_l$ at the end of the derived word (after w).

□

The lemma can be generalised. We see that we use pumping only once, but the loop could be repeated any number of times. We also give a big restriction on the jumping, as we only consider jumps to the beginning or the end. However, such a restricted version of the lemma suffices for the purpose of this paper.

We now consider an important language and show that it contains words of a specific form.

Example 3.30. Let L be a language generated by the following jumping context-free grammar

$$G = (\{S, X\}, \{a, b, c, d\}, \{S \to adSX, S \to adX, X \to bc\}, S).$$

By definition $L \in \mathcal{JCFL}$.

We state and briefly prove some important properties of language L from the above example.

Lemma 3.31. *L has the following properties:*

(1) Words from L have the same number of letters a, b, c and d.
(2) Words of the form $a^r b^r c^r d^r$ where $r > 0$ are in L.
(3) Words from L cannot contain any letter a after the last letter d.
(4) Words from L cannot contain any letter d before the first letter a.

Proof. (1) We see that in an arbitrary derivation a pair ad always comes with a symbol X, which is later substituted by bc.

(2) We use the jumping rule $S \to adSX$ $r-1$-times, always jumping in between a and d. Then we use $S \to adX$, acquiring $S \Rightarrow adSX \Rightarrow aadSXdX \Rightarrow \ldots \Rightarrow a^r(dX)^r$. After that, we use the rule $X \to bc$ r-times firstly jumping between a and d, and afterwards between b and c.

$$S \Rightarrow \ldots \Rightarrow a^r(dX)^r \Rightarrow a^r bcd(dX)^{r-1} \Rightarrow a^r bbccdd(dX)^{r-2} \Rightarrow \ldots \Rightarrow a^r b^r c^r d^r.$$

(3, 4) This is due to the form of the rules, in which the letter a is accompanied by letter d to the right.

□

We are now ready to prove the following.

Proposition 3.32. $L \notin \mathcal{JRLL}$.

Proof. We assume $L \in \mathcal{JRLL}$. Then, there exists a jumping right-linear grammar $G = (N, T, P, S)$ of span m, with no unit rules, such that $L = L(G)$. Let $n = |N|$ and $r = n(m-1) + 1$. From property 2 from Lemma 3.31, we know that $w = a^r b^r c^r d^r \in L$.

Let $w_k, w_{k+1}, \ldots, w_l$ be the words from Lemma 3.29, from the derivation of w. We consider the following four cases:

- $\forall_{v\in\{w_k,w_{k+1},\ldots,w_l\}}\ |v|_a = 0 \wedge |v|_d = 0$.
  In this case, by pumping and jumping we violate the property 1 from Lemma 3.31. The number of b and c is greater than the number of a and d.
- $\exists_{v\in\{w_k,w_{k+1},\ldots,w_l\}}\ |v|_a \neq 0 \wedge |v|_d = 0$.
  We put the pumped v at the end of the word, which leads to a contradiction with the property 3 from Lemma 3.31.
- $\exists_{v\in\{w_k,w_{k+1},\ldots,w_l\}}\ |v|_a = 0 \wedge |v|_d \neq 0$.
  We violate the property 4 from Lemma 3.31 by putting the pumped v at the beginning of the derived word.
- $\exists_{v\in\{w_k,w_{k+1},\ldots,w_l\}}\ |v|_a \neq 0 \wedge |v|_d \neq 0$.
  Note that a should be before d in v, otherwise it is possible to destroy property 3. Before the application of the rule containing v, no letter b or c is derived. Otherwise, w would contain a subword with b or c appearing either before both a and d, or after. Thus, all $2r$ letters b and c must be derived during and after the application of the rule with v. This is not possible, as in the last $n+1$ steps, only $n(m-1)+m = r+m-1$ letters can be derived and that is strictly smaller than $2r$.

In all cases, we reached a contradiction which is due to a false assumption that $L \in \mathcal{JRLL}$. □

Theorem 3.33. $\mathcal{JRLL} \subsetneq \mathcal{JCFL}$.

Proof. The inclusion of the classes follows from their definition. The strictness of the inclusion follows from Example 3.30 and Proposition 3.32. □

4. Conclusions

In this article, we defined regular and linear grammars extended with permutation rules. These rules allow the non-terminal symbol and the neighbouring terminal symbols to permute. We investigated the generative power of these grammars. It was shown that by adding an erasing rule we obtain classes of languages strictly greater than those not using it: $\mathcal{RPL} \subsetneq \mathcal{RPL}_\lambda$, $\mathcal{LPL} \subsetneq \mathcal{LPL}_\lambda$. This result can be quite surprising, as the erasing rule is always used once, at the end of the derivation. We also studied the generative power with respect to the span of the rules of grammars and established an infinite hierarchy $\mathcal{LPL}_\lambda(m) \subsetneq \mathcal{LPL}_\lambda(m+1)$ for any $m \geq 2$.

We investigated some basic closure properties of these classes (union, intersection and concatenation). Many other operations (shuffle, shuffle closure and homomorphism) were not considered and proving or disproving closure under these operations remains an open problem.

We studied the decidability of some problems. The membership problem and emptiness problem were proven to be decidable and the disjointness problem is undecidable. There are several other classical decision problems not considered in this paper: equality, universality, inclusion, etc. Some of them are undecidable for linear languages [1].

Finally, a connection between permutation grammars and jumping grammars was established and some results on jumping grammars were presented.

References

[1] B.S. Baker and R.V. Book, Reversal-bounded multipushdown machines. *J. Comput. Syst. Sci.* **88** (1974) 315–332.
[2] W. Czerwinski and S. Lasota, Partially-commutative context-free languages, in *Proceedings Combined 19th International Workshop on Expressiveness in Concurrency* and *9th Workshop on Structured Operational Semantics, EXPRESS/SOS 2012, Newcastle upon Tyne, UK, September 3, 2012*, edited by B. Luttik and M.A. Reniers. Vol. 89 of *EPTCS*, Open Publishing Association, Waterloo (2012) 35–48.
[3] H. Fernau, M. Paramasivan, M.L. Schmid and V. Vorel, Characterization and complexity results on jumping finite automata. *Theor. Comput. Sci.* **679** (2017) 31–52.

[4] J.E. Hopcroft, R. Motwani and J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, 2nd edn. *Addison-Wesley Series in Computer Science.* Addison-Wesley-Longman, MA (2001).

[5] G. Horváth and B. Nagy, Pumping lemmas for linear and nonlinear context-free languages. *Acta Univ. Sapientiae, Informatica* **2** (2010) 194–209.

[6] Z. Krivka and A. Meduna, Jumping grammars. *Int. J. Found. Comput. Sci.* **26** (2015) 709–732.

[7] G. Madejski, Infinite hierarchy of permutation languages. *Fundam. Inform.* **130** (2014) 263–274.

[8] G. Madejski, The membership problem for linear and regular permutation languages, in *Implementation and Application of Automata – 20th International Conference, CIAA 2015, Umeå, Sweden, August 18–21, 2015*, Proceedings, edited by F. Drewes. Vol. 9223 of *Lecture Notes in Computer Science*. Springer, Cham (2015) 211–223.

[9] G. Madejski, Jumping and pumping lemmas and their applications, in *Eighth Workshop on Non-Classical Models of Automata and Applications, NCMA 2016, Debrecen, Hungary, August 29–30, 2016*. Short Papers, edited by H. Bordihn, R. Freund, B. Nagy and G. Vaszil. TU Wien, Vienna (2016) 25–33.

[10] G. Madejski, Regular and linear permutation languages, in *Eighth Workshop on Non-Classical Models of Automata and Applications, NCMA 2016, Debrecen, Hungary, August 29–30, 2016*. Proceedings, edited by H. Bordihn, R. Freund, B. Nagy and G. Vaszil, Vol. 321 of books@ocg.at. Österreichische Computer Gesellschaft, Wien (2016) 243–258.

[11] A. Meduna and P. Zemek, Jumping finite automata. *Int. J. Found. Comput. Sci.* **23** (2012) 1555–1578.

[12] B. Nagy, Languages generated by context-free grammars extended by type AB -> BA rules. *J. Autom. Lang. Comb.* **14** (2009) 175–186.

[13] B. Nagy, Permutation languages in formal linguistics, in Bio-Inspired Systems: Computational and Ambient Intelligence, *10th International Work-Conference on Artificial Neural Networks, IWANN 2009, Salamanca, Spain, June 10–12, 2009*. Proceedings, Part I, edited by J. Cabestany, F.S. Hernández, A. Prieto and J.M. Corchado. Vol. 5517 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg (2009) 504–511.

[14] B. Nagy, On a hierarchy of permutation languages, in Automata, Formal Languages and Algebraic Systems. *Proceedings of AFLAS 2008, Kyoto, Japan, September 20–22, 2008*, edited by M. Ito, Y. Kobayashi and K. Shoji. World Scientific, Singapore (2010) 163–178.

[15] B. Nagy, Linguistic power of permutation languages by regular help, in Bio-Inspired Models for Natural and Formal Languages, edited by G. Bel-Enguix and M.D. Jiménez-López. Cambridge Scholars, Cambridge (2011) 135–152.