Partially-commutative context-free languages

Wojciech Czerwiński
Institute of Informatics, University of Warsaw *
wczerwin@mimuw.edu.pl

Sławomir Lasota
Institute of Informatics, University of Warsaw †
sl@mimuw.edu.pl

The paper is about a class of languages that extends context-free languages (CFL) and is stable under shuffle. Specifically, we investigate the class of *partially-commutative context-free languages* (pc CFL), where non-terminal symbols are commutative according to a binary independence relation, very much like in trace theory. The class has been recently proposed as a robust class subsuming CFL and commutative CFL. This paper surveys properties of pc CFL. We identify a natural corresponding automaton model: stateless multi-pushdown automata. We show stability of the class under natural operations, including homomorphic images and shuffle. Finally, we relate expressiveness of pc CFL to two other relevant classes: CFL extended with shuffle and trace-closures of CFL. Among technical contributions of the paper are pumping lemmas, as an elegant completion of known pumping properties of regular languages, CFL and commutative CFL.

1 Introduction

Closure of languages classes under shuffle is intensively investigated, see for instance [3] and further references therein. This paper is about a subtle way of introducing shuffle into context-free grammars.

Process algebraic motivation. In the context of infinite-state verification there are two basic well known classes of systems. *Context-free processes*, called traditionally BPA[1] [2], stand for the most fundamental abstract model of sequential recursive programs. BPA contains configuration graphs induced by context-free grammars in Greibach normal form. The commutative variant, *commutative context-free processes*, traditionally called BPP[2], was proposed in [5] as the abstract model of concurrent programs. BPP differs from BPA in that it has parallel composition instead of sequential composition. Thus a configuration is a finite multiset of non-terminals rather than a sequence.

A natural generalization of both BPA and BPP is *Process Algebra* (PA) [2] where one allows for both kinds of composition.[3] A standard reference for a process-rewrite formulation of PA is [12]. However, PA does not seem to have good algorithmic properties. For instance, bisimulation equivalence is not known to be decidable, a long standing open problem [15], while the algorithm for normed PA is very complex and as costly as double exponential time [11]. This has recently motivated investigation of an alternative but equally natural generalization of both BPA and BPP, namely *partially-commutative context-free processes*, called BPC[4] in [6]. BPC processes are also defined by a Greibach grammar, but one additionally assumes a binary *independence relation* among non-terminals, like in trace theory [13], and only independent pairs of non-terminals commute. We stress that the independence is imposed not on alphabet letters, which is usually the case in trace languages, but on non-terminals. Thus a configuration

*The first author acknowledges a partial support by the Polish MNiSW grant N N206 568640.
†The second author acknowledges a partial support by the Polish MNiSW grant N N206 356036.
[1] A shorthand for Basic Process Algebra.
[2] A shorthand for Basic Parallel Processes Algebra.
[3] The algebra of [2] includes also left merge operation, not considered in this paper.
[4] A shorthand for Basic Partially Commutative Algebra.

B. Luttik and M.A. Reniers (Eds.): Combined Workshop on Expressiveness in Concurrency and Structural Operational Semantics (EXPRESS/SOS 2012)
EPTCS 89, 2012, pp. 35–48, doi:10.4204/EPTCS.89.4

may be modeled as a trace over non-terminals. BPA is a special case where no non-terminals commute while BPP, on the other hand, is another special case where all non-terminals commute.

In [6, 7] an efficient polynomial-time procedure has been developed for bisimulation equivalence, that works correctly in the subclass of normed BPC that strictly contains both normed BPA and BPP. We also very recently analyzed the reachability problem for BPC [8]. In this paper we continue the program that aims at finding a robust class subsuming BPA and BPP, however this time from the language-theoretic perspective.

Language theoretic motivation. BPA clearly defines context-free languages (CFL) and BPP defines so called commutative context-free languages ($_{c}$CFL) [5][5], equivalently characterized as languages of communication-free Petri nets. In this paper we focus on *partially-commutative context-free languages* ($_{pc}$CFL) [7] that are defined by BPC. Our aim is to investigate properties of this class and to relate its expressiveness with other classes.

The class $_{pc}$CFL extends CFL and is closed under the shuffle operation. By a shuffle of two words we mean here an arbitrary interleaving of these words and by shuffle of two languages we mean all shuffles of all pairs of words from the two languages. Other similar extensions of CFL may be found in the literature. One such extension is PA languages. We use a shorthand $_{shuffle}$CFL for this class – as far as languages are concerned, PA is equivalent to context-free grammars where one allows to use both concatenation and shuffle in productions [10, 14]. Another related class is trace-closures of CFL (name this class $_{trace}$CFL), where one assumes, contrary to $_{pc}$CFL, an independence relation on alphabet letters. We have found it appealing to relate the expressive power of $_{pc}$CFL with $_{shuffle}$CFL and $_{trace}$CFL.

Our contribution. First, we show that a relevant subclass of $_{pc}$CFL, subject to the restriction that the complement of independence relation is transitive, has a natural corresponding automaton model: stateless multi-pushdown automata (Section 2). We also prove that the membership problem for $_{pc}$CFL is NP-complete thus the complexity remains the same as for $_{c}$CFL (Section 3).

Second, in Section 4 we investigate stability of $_{pc}$CFL under natural operations. In particular, $_{pc}$CFL turns out to be stable under homomorphic images, substitutions and shuffle. On the other hand, the class is not stable under inverse homomorphic images and under intersections with regular languages. The latter is not very surprising as we consider a natural extension of $_{c}$CFL, the class that lacks not only the two closure properties, but even lacks closure under concatenation and homomorphic images! With $_{pc}$CFL one regains closure under concatenation and homomorphic images.

Third, in Sections 6 and 7 we perform mutual comparison of expressiveness of $_{pc}$CFL, transitive $_{pc}$CFL, $_{shuffle}$CFL and $_{trace}$CFL, proving them all pairwise incomparable (except for the trivial inclusion of transitive $_{pc}$CFL in $_{pc}$CFL, that we prove to be strict). Note that incomparability with respect to languages implies incomparability with respect to bisimulation or other equivalences. As one of the tools we formulate and prove pumping lemmas for classes $_{pc}$CFL, transitive $_{pc}$CFL and $_{shuffle}$CFL. This provides an elegant completion of known pumping properties of regular, context-free and commutative context-free languages.

Technically, the most difficult part is Sections 6 and 7. On the other hand, the results of Sections 2 and 4 confirm clearly that $_{pc}$CFL is a natural class of languages extending CFL, with good algorithmic and closure properties.

Yet another relevant language class is that defined by so called Dynamic Pushdown Networks [4]. The class extends CFL and is closed under shuffle. We do not investigate this class here, but we conjecture that it is incomparable with $_{pc}$CFL.

[5] In fact, BPA and BPP define CFL and $_{c}$CFL not containing the empty word, respectively.

Some of the proofs are omitted due to space limitation.

2 Preliminaries

By an *interleaving* of two words w and v, of length m and n, respectively, we mean any word u of length $m+n$ such that its positions $I = \{1,\ldots,m+n\}$ may be split into two disjoint sets I_w and I_v such that u restricted to I_w equals w and u restricted to I_v equals v. Let $w \,||\, v$ denote the set of all the interleavings of w and v, which is clearly a finite set. By a shuffle of two languages L and K we mean

$$L||K = \bigcup_{w\in L, v\in K} w||v.$$

Partially-commutative context-free languages. The class of languages to be defined below has been introduced in [7], however our presentation and terminology here is different.

A Greibach context-free grammar consists of a finite alphabet, a finite set of non-terminal symbols V with a distinguished initial symbol $S \in V$ and a finite set of productions of the form

$$X \xrightarrow{a} \alpha, \tag{1}$$

where $X \in V$, $\alpha \in V^*$ and a is an alphabet letter. Additionally we assume that a grammar is always equipped with a symmetric and irreflexive relation $I \subseteq V \times V$ called the *independence relation*. For convenience we also use the complement $D = (V \times V) \setminus I$, called the *dependence relation*. Two non-terminals $X, Y \in V$ are called *independent* if $(X,Y) \in I$, and otherwise *dependent*.

Any $\alpha \in V^*$ we call a *configuration*. A *derivation* is a sequence of configurations such that every configuration is obtained from the preceding one via a *step* and the last one is the empty configuration. There are two kinds of steps:

- *production step*: $X\beta \xrightarrow{a} \alpha\beta$, for a production $X \xrightarrow{a} \alpha$;
- *swap step*: $\alpha XY\beta \longrightarrow \alpha YX\beta$, where X and Y are independent.

Every derivation *defines* a word w obtained by concatenation of alphabet letters occurring in the production steps. We write $\alpha \xrightarrow{w} \beta$ if there is a derivation that defines w, starts in α and ends in β. We usually assume that a derivation starts with a configuration consisting of a single non-terminal, say X. If $X \xrightarrow{w} \varepsilon$ then we say that X *generates* w. Note that the length of w is the same as the number of production steps performed in any derivation that defines w. We assume wlog. that every non-terminal X generates some word.

The language generated by a grammar is the set of all words generated by the initial non-terminal. The class of all so generated languages we call *partially-commutative context-free languages* (pc CFL) [7]. It clearly contains all context-free languages (CFL) and commutative context-free languages[6] (c CFL) [5]. These two subclasses are special cases, where independence is either the identity, or the full relation, respectively.

Example 1. For illustration, consider the grammar:

$$\begin{array}{llllllll} P \xrightarrow{a} WBC\bar{B} & & W \xrightarrow{a} WBC & & \bar{B} \xrightarrow{\bar{b}} \varepsilon & & B \xrightarrow{b} \varepsilon \\ & & W \xrightarrow{\bar{a}} \bar{C} & & \bar{C} \xrightarrow{\bar{c}} \varepsilon & & C \xrightarrow{c} \varepsilon \end{array}$$

[6] The commutative context-free languages are also called *BPP languages*.

The initial non-terminal is P and the independence relation is the symmetric closure of $\{B,\bar{B}\} \times \{C,\bar{C}\}$. Here is an example derivation of the word $a\bar{a}b\bar{b}\bar{c}c$.

$$\begin{array}{lllllllll} P & \xrightarrow{a} & WBC\bar{B} & \xrightarrow{\bar{a}} & \bar{C}BC\bar{B} & \longrightarrow & \bar{C}B\bar{B}C & \longrightarrow & B\bar{C}\bar{B}C & \longrightarrow \\ & & B\bar{B}\bar{C}C & \xrightarrow{b} & \bar{B}\bar{C}C & \xrightarrow{\bar{b}} & \bar{C}C & \xrightarrow{\bar{c}} & C & \xrightarrow{c} & \varepsilon \end{array}$$

In a similar way a word $a^n\bar{a}b^n\bar{b}\bar{c}c^n$ is generated, for any $n \geq 1$, but also $a^n\bar{a}\bar{c}c^nb^n\bar{b}$ or $a^n\bar{a}\bar{c}b^nc^n\bar{b}$. The language generated is

$$\bigcup_{n\geq 1} a^n\bar{a}\,(b^n\bar{b} \,||\, \bar{c}c^n).$$

We might have defined configurations as Mazurkiewicz traces [13] rather than words over non-terminals (like in [7]). This would mean that *trace equivalent* configurations are not distinguished. In our terminology, two configurations are trace equivalent when one may be transformed into another using solely swap steps. It is our deliberate choice to keep the swap steps explicit.

Transitive dependence. We distinguish a subclass of $_{\text{pc}}\text{CFL}$ where dependence is assumed to be transitive, being thus an equivalence. This subclass we name $^{\text{tr}}_{\text{pc}}\text{CFL}$. Equivalence classes of dependence will be called *threads*.

In Example 1 the dependence is not transitive, as it contains (P,B) and (P,C) but not (B,C). In fact we show later that this language does not belong to $^{\text{tr}}_{\text{pc}}\text{CFL}$. Both CFL and $_{\text{c}}\text{CFL}$ are strict subclasses of $^{\text{tr}}_{\text{pc}}\text{CFL}$.

Example 2. As an illustration, consider the language generated by:

$$\begin{array}{llllllllll} S & \xrightarrow{s} & \varepsilon & \qquad S & \xrightarrow{a} & SA & \qquad A & \xrightarrow{c} & A' & \qquad A' & \xrightarrow{a} & \varepsilon \\ & & & \qquad S & \xrightarrow{b} & SB & \qquad B & \xrightarrow{c} & B' & \qquad B' & \xrightarrow{b} & \varepsilon \end{array}$$

with initial non-terminal S and the threads $\{S,A,B\}$, $\{A'\}$ and $\{B'\}$. Here is an example derivation of the word *absccab*.

$$S \xrightarrow{a} SA \xrightarrow{b} SBA \xrightarrow{s} BA \xrightarrow{c} B'A \longrightarrow AB' \xrightarrow{c} A'B' \xrightarrow{a} B' \xrightarrow{b} \varepsilon.$$

The language contains words of the form wsv, where w contains only a and b and v contains only a, b and c. Writing $\#_a(w)$ for the number of occurrences of a in w and $|w|$ for the length of w, we may characterize the language by the following conditions:

- $\#_a(w) = \#_a(v)$, $\#_b(w) = \#_b(v)$ and $\#_c(v) = \#_a(v) + \#_b(v)$,
- any prefix v' of v and any suffix w' of w such that $\#_c(v') = |w'|$ fulfills

$$\#_a(w') \geq \#_a(v') \qquad \text{and} \qquad \#_b(w') \geq \#_b(v').$$

Automaton model. A multi-pushdown automaton is like a single-pushdown one. In a single step one symbol is popped from one of the stacks,[7] and a number of symbols are pushed on the stacks. The

[7] If we allowed for popping from more than one stack at a time, the model would clearly become Turing-complete, even with only one state.

number of stacks is fixed for an automaton. Assume there is only one state, or equivalently no state, and k stacks. Then a transition of an automaton is of the form:

$$X \xrightarrow{a} \alpha_1 \dots \alpha_k, \tag{2}$$

to mean that when an automaton reads a, it pops X and pushes the sequence of symbols α_i on the ith stack, for $i = 1 \dots k$. Observe that wlog. one may assume that stack alphabets are disjoint. The following result is an easy observation:

Theorem 1 ([8]) *The* $\overset{\text{tr}}{\text{pc}}\text{CFL}$ *class is expressively equivalent to stateless multi-pushdown automata.*

Indeed, an equivalence class of configurations with respect to trace equivalence is represented by a tuple of strings, one per thread. Similarly, a production $X \xrightarrow{a} \alpha$ is represented, up to swap steps, exactly as in (2), with α_i being the projection of α on the ith thread.

Similarly, one could also define an operational model for general $\text{pc}\,\text{CFL}$, with a stack replaced by a partially ordered structure.

3 Derivation trees

It is very convenient to use derivation trees instead of derivations themselves. However it is not completely obvious how to define this notion in presence of commutativity of non-terminals. Below we adopt an intuitive approach using colors.

Fix a derivation $X \xrightarrow{w} \varepsilon$. Clearly a configuration is a sequence of *non-terminal occurrences*. We assume that every non-terminal occurrence in a derivation will be colored, including the occurrence of X in the initial configuration. We impose the following simple discipline of coloring:

- if a swap step $\alpha XY\beta \longrightarrow \alpha YX\beta$ is performed, every non-terminal occurrence in the right-hand side configuration inherits its color from the corresponding occurrence of the same non-terminal on the left-hand side.
- if a production step $X\beta \xrightarrow{a} \alpha\beta$ is performed, the non-terminal occurrences in β preserve their colors, while all the non-terminals occurrences in α get fresh colors. Note that the color of the occurrence of X in the beginning of $X\beta$ disappears as a result of the step. We say that this disappearing color *drops* the fresh colors.

Intuitively, a color is intended to represent the 'life cycle' of one occurrence of a non-terminal during a derivation. Observe that non-terminal occurrences in a given configuration are always labeled with different colors, and that the total number of colors used in a derivation equals the number of production steps.

Example 3. A disciplined coloring of the derivation from Example 2 is shown below. Colors are $1, 2, \dots$ and the coloring is denoted by subscripts.

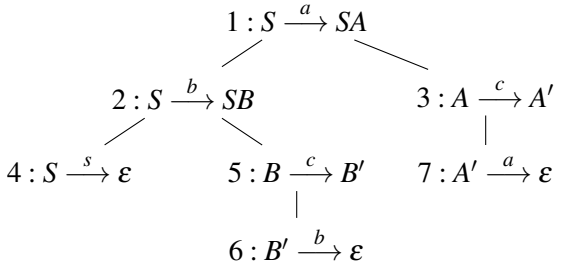
$$S_1 \xrightarrow{a} S_2A_3 \xrightarrow{b} S_4B_5A_3 \xrightarrow{s} B_5A_3 \xrightarrow{c} B'_6A_3 \longrightarrow A_3B'_6 \xrightarrow{c} A'_7B'_6 \xrightarrow{a} B'_6 \xrightarrow{b} \varepsilon. \tag{3}$$

Color 1 drops colors 2 and 3, color 3 drops color 7, etc.

With the use of our coloring discipline, every derivation induces naturally a tree. The tree nodes are all colors appearing in the derivation. The color c_1 is a parent of c_2 precisely if c_1 drops c_2. Every tree node c is labeled by a non-terminal. If convenient, one may think that every node is labeled by a production that made color c disappear.

There may be many different derivations inducing the same tree. Even worse, two derivations *of different words* may induce the same tree, as shown in the example below.

Example 4. Continuing the last example, the derivation (3) induces the following tree:



However, exactly the same tree is induced by the derivation:

$$S_1 \xrightarrow{a} S_2A_3 \xrightarrow{b} S_4B_5A_3 \xrightarrow{s} B_5A_3 \xrightarrow{c} B'_6A_3 \xrightarrow{b} A_3 \xrightarrow{c} A'_7 \xrightarrow{a} \varepsilon$$

of a different word $abscbca \neq absccab$. Intuitively, the words defined by subtrees rooted in 3 and 6, namely ca and b respectively, this time come in a different order. In fact all the interleavings of these two words are allowed.

Useful properties. The examples confirm that our notion of derivation tree is more complex than the classical one. However, trees may be still very useful for reasoning about partially-commutative context-free languages, as they immediately bring to light the following useful properties:

INDUCED SUBWORD. Given a derivation tree of a word w, every node c induces a subword (i.e. a subsequence but not an infix in general) of w. Indeed, the subword is obtained by concatenating only those letters from w whose color, as a tree node, belongs to the subtree rooted in c. We implicitly assign here to the letter of every production step a color that disappears in this step. For instance, for both words considered in the last example, the subword induced by the node 2 is $bscb$. Analogously one defines the subword induced by a subset of nodes of a derivation tree, assuming this subset to be an antichain with respect to the tree ancestor relation.

INFIX REARRANGEMENT. The induced subword may be rearranged into an infix. Let $L \in {}^{\text{pc}}\text{CFL}$ and let v be the subword of $w \in L$ induced by a tree node c. Clearly, $w \in v \,||\, u$, i.e., v is interleaved with the remaining subword u of w. Then u may be split into $u = u_1u_2$ so that $u_1vu_2 \in L$. Indeed, let u_1 be the prefix of w preceding the first letter of v. In any derivation, after u_1, the non-terminal that labels c is clearly active. Performing the whole derivation $X \xrightarrow{v} \varepsilon$ immediately after u_1 does the job.

SUBSTITUTIVITY. In any derivation tree, one may replace a subtree rooted in a node c by an arbitrary derivation tree t, assumed that both c and the root of t are labeled with the same non-terminal. The resulting tree is clearly induced by some derivation too.

Membership problem. A derivation tree is of linear size in terms of the length of the word, which is useful for easily obtaining the upper bound for the membership problem, where given a word w and a presentation of a language L, one asks if $w \in L$?

Theorem 2 *The membership problem is NP-complete both for* ${}^{\text{pc}}\text{CFL}$ *and* ${}^{\text{tr}}_{\text{pc}}\text{CFL}$.

NP-hardness follows easily from NP-hardness of the membership problem for ${}^{\text{c}}\text{CFL}$, shown in [9]. The NP upper bound one obtains easily: guess a tree and the order of its nodes, and then check in polynomial time whether the tree is induced by some derivation of the given word that respects the order of nodes.

4 Closure properties

In this section we argue that $\mathrm{pc}\,\mathrm{CFL}$ and $\overset{\mathrm{tr}}{\mathrm{pc}}\,\mathrm{CFL}$ classes are closed under union and shuffle, and $\mathrm{pc}\,\mathrm{CFL}$ is closed under concatenation while $\overset{\mathrm{tr}}{\mathrm{pc}}\,\mathrm{CFL}$ is not. Then we show that $\mathrm{pc}\,\mathrm{CFL}$ is closed under homomorphic images and substitutions. In case of $\overset{\mathrm{tr}}{\mathrm{pc}}\,\mathrm{CFL}$ we do not know the answer, however we suppose it is negative. Finally, we show that both classes lack closure under inverse homomorphic images and intersections with regular languages.

Comparing $\mathrm{pc}\,\mathrm{CFL}$ with CFL, roughly speaking, one sacrifices intersection with regular languages and inverse homomorphic images but one gains shuffle. Even if at first sight the properties listed above do not seem exciting, one should remember that both the classes considered here subsume also commutative context-free languages $\mathrm{c}\,\mathrm{CFL}$. Knowing that $\mathrm{c}\,\mathrm{CFL}$ lacks closure under concatenation and homomorphic images, as shown in [5], it seems that with $\mathrm{pc}\,\mathrm{CFL}$ one *retrieves* these relevant closure properties. This seems to confirm that $\mathrm{pc}\,\mathrm{CFL}$ is a natural class of languages.

Union and complement. Both classes are closed under union and the construction is entirely standard. On the other hand none of the classes is closed under complement.

Shuffle and concatenation. Both classes are closed under shuffle and the construction of a grammar for the shuffle $L_1 \,||\, L_2$ is easy. Wlog assume that the grammars that generate the two languages use distinct non-terminals. Let S_1 and S_2 be the initial non-terminals. Consider the union of grammars extended with one additional initial non-terminal S. Add additional productions

$$S \xrightarrow{a_1} \alpha_1 S_2 \qquad S \xrightarrow{a_2} \alpha_2 S_1 \tag{4}$$

for any production $S_1 \xrightarrow{a_1} \alpha_1$ or $S_2 \xrightarrow{a_2} \alpha_2$. Finally, extend independence by imposing that whenever two non-terminals come from different grammars they are independent. This clearly preserves transitivity of dependence.

In $\mathrm{pc}\,\mathrm{CFL}$, concatenation $L_1 L_2$ is obtained similarly as shuffle. The only difference is that two non-terminals coming from different grammars are always declared dependent, and that only the left-hand productions in (4) are added. Note that concatenation is in our setting no more natural than shuffle.

$\overset{\mathrm{tr}}{\mathrm{pc}}\,\mathrm{CFL}$ is not closed under concatenation, which one shows similarly as for $\mathrm{c}\,\mathrm{CFL}$ [5]. Consider $L_1 = \{w : \#_a(w) = \#_b(w) = \#_c(w) \geq 1, \#_d(w) = 0\}$ and $L_2 = \{d\}$. In the derivation of some $w \in L_1 L_2$ a configuration is necessarily reached with at least two different threads nonempty, as otherwise the language would be context-free. Thus the remaining suffix of w is some shuffle of at least two words generated by these non-empty threads, and only one of these words ends with d. If that subword is generated first, the whole word is not in $L_1 L_2$, which proves that $L_1 L_2$ may not belong to $\mathrm{c}\,\mathrm{CFL}$.

Homomorphic images and substitutions. As we consider only Greibach grammars, the empty word never belongs to a partially-commutative context-free language. Thus it is natural to consider only homomorphisms h that do not contain the empty word in the image: $h(a) \neq \varepsilon$ for all letters a. Below we show that $\mathrm{pc}\,\mathrm{CFL}$ is closed under images of such homomorphisms. For $\overset{\mathrm{tr}}{\mathrm{pc}}\,\mathrm{CFL}$ the question is still open; we conjecture however a negative answer.

We prefer to show a slightly stronger result: $\mathrm{pc}\,\mathrm{CFL}$ is closed under *substitutions*. A substitution s assigns to each alphabet letter a a language $s(a) \in \mathrm{pc}\,\mathrm{CFL}$. Similarly as above, we assume that the languages $s(a)$ do not contain the empty word. For a language L, the substitution $L[s]$ contains all words that may be obtained from a word in L, by replacing each letter a with any word from $s(a)$.

Assume a language $L \in \mathrm{pc}\,\mathrm{CFL}$, generated by a grammar G, and a substitution s. Thus each language $s(a)$ has its generating grammar G_a. We describe the construction of the grammar G' for $L[s]$. The non-

terminals of G' will be the union of non-terminals of G and all grammars G_a. Wlog we assume that the non-terminal sets are disjoint.

Consider an arbitrary production $X \xrightarrow{a} \alpha$ in G. Let S_a be the initial non-terminal in G_a. For any production $S_a \xrightarrow{b} \beta$ in G_a, we add to G' the production: $X \xrightarrow{b} \beta\alpha$. The independence in G' is defined as the set-theoretic union of independence relations of grammars G and G_a. Thus any pair of non-terminals coming from different grammars is declared dependent (note that this is not achievable if the dependence has to be transitive).

The construction guarantees that G' generates exactly $L[s]$. Indeed, once a production $X \xrightarrow{b} \beta\alpha$ is fired, the non-terminals of G_a block activity of other non-terminals, due to the dependence, until a word of $s(a)$ is generated.

We do not know whether the $^{\mathrm{tr}}_{\mathrm{pc}}\mathrm{CFL}$ class is closed under homomorphic images; however we suppose it is not. We conjecture that a counterexample is given by the language

$$L = \{w : \#_a(w) = \#_b(w) = \#_c(w), \#_d(w) = 1\}$$

together with the homomorphism $h(a) = a$, $h(b) = b$, $h(c) = c$, $h(d) = dd$.

Intersection with regular languages. Both classes $_{\mathrm{pc}}\mathrm{CFL}$ and $^{\mathrm{tr}}_{\mathrm{pc}}\mathrm{CFL}$ lack closure under intersection with regular languages. Let $L = \{w : \#_a(w) = \#_b(w) = \#_c(w)\}$. Clearly $L \in {}_{\mathrm{c}}\mathrm{CFL}$ but $L \cap a^*b^*c^*$ is not in $_{\mathrm{pc}}\mathrm{CFL}$ (and also not in $_{\mathrm{shuffle}}\mathrm{CFL}$ defined in a moment) according to:

Lemma 1 *The language $L = \{a^n b^n c^n : n \geq 1\}$ is not in $_{\mathrm{pc}}\mathrm{CFL} \cup {}_{\mathrm{shuffle}}\mathrm{CFL}$.*

It is worth noting that the lack of closure is not surprising as the emptiness problem for intersection of a partially-commutative context-free language with a regular language is undecidable, even if the dependence is assumed to be transitive. Roughly speaking $^{\mathrm{tr}}_{\mathrm{pc}}\mathrm{CFL}$ correspond to stateless multi-pushdown automata and intersection with regular language corresponds do adding the state which makes the model Turing powerful.

Inverse homomorphic images. Both $_{\mathrm{pc}}\mathrm{CFL}$ and $^{\mathrm{tr}}_{\mathrm{pc}}\mathrm{CFL}$ are not closed under inverse homomorphic images. Consider the shuffle $L = L_1 \,||\, L_2$ of two context-free languages

$$L_1 = \{A^{n+1}SB^nT : n \geq 1\} \qquad L_2 = \{SB^nTC^n : n \geq 1\},$$

and the homomorphism h given by $h(a) = A$, $h(s) = SS$, $h(b) = BB$, $h(t) = TT$ and $h(c) = C$. If $h^{-1}(L) = \{a^{n+1}sb^ntc^n : n \geq 1\}$ were in $_{\mathrm{pc}}\mathrm{CFL}$ then its image under a homomorphism $g(s) = b, g(t) = c$, that is the language L in Lemma 1, would be in $_{\mathrm{pc}}\mathrm{CFL}$ as well – a contradiction.

5 Other extensions of context-free languages

There are two other language classes know from the literature that, similarly as $_{\mathrm{pc}}\mathrm{CFL}$, extend CFL with some amount of commutation.

PA languages. The formalism to be described below is traditionally called *Process Algebra* (PA) [2, 12]. It is however nothing else than an extension of Greibach context-free grammars with an explicit shuffle operation: a production has the form

$$X \xrightarrow{a} t,$$

where t is an arbitrary term built from non-terminals using binary operations of sequential composition ';' and parallel composition '$||$'. The first operation one may interpret as concatenation of languages,

and the second one as shuffle (thus the overloading of the symbol $||$ is absolutely deliberate). The *empty term* ε is also allowed.

For convenience, terms are only considered up to a structural equivalence, that imposes associativity of both operations, commutativity of $||$, and neutrality of ε with respect to both operations.

A *configuration* is an arbitrary term of the above form. Steps between configurations are defined by the following rules (the last rule is in fact redundant due to commutativity of $||$, but we prefer to keep it for readability):

$$\frac{X \xrightarrow{a} t \text{ is a production}}{X \xrightarrow{a} t} \qquad \frac{t \xrightarrow{a} t'}{t;u \xrightarrow{a} t';u} \qquad \frac{t \xrightarrow{a} t'}{t\,||\,u \xrightarrow{a} t'\,||\,u} \qquad \frac{u \xrightarrow{a} u'}{t\,||\,u \xrightarrow{a} t\,||\,u'}$$

As usual, a derivation is a sequence of configurations starting from a distinguished initial configuration S, ending in the empty configuration, such that every subsequent configuration is obtained from a preceding one by a single step. Other notions, including the language generated by a grammar, or derivation trees, may be defined similarly as for ${}_{\text{pc}}\text{CFL}$. The class of languages we denote by ${}_{\text{shuffle}}\text{CFL}$.

In particular, ${}_{\text{shuffle}}\text{CFL}$ satisfy the three properties mentioned above: INDUCED SUBWORD, INFIX REARRANGEMENT and SUBSTITUTIVITY.

The difference between ${}_{\text{pc}}\text{CFL}$ and ${}_{\text{shuffle}}\text{CFL}$ is, roughly, a difference between specifying commutation explicitly in productions, or implicitly by an independence relation.

Trace-closures of CFL. To define ${}_{\text{trace}}\text{CFL}$ we need to assume that an independence relation ranges not over non-terminals but over alphabet letters instead. As usual, one defines *trace equivalence* over words: two words are equivalent if one may be transformed into another by swaps of neighboring independent letters. A context-free language L is not closed under this equivalence in general and its *trace closure*

$$\{w : w \text{ is trace equivalent to some } v \in L\}$$

is in general not context-free. By ${}_{\text{trace}}\text{CFL}$ we denote the class containing trace closures of context-free languages. Clearly ${}_{\text{trace}}\text{CFL}$ is a superclass of CFL.

6 Pumping lemmas

Now we analyze how much the classical idea of pumping extends from CFL to larger classes. Roughly speaking, the intuitive cutting and pasting in a derivation tree does not translate to the property of a language as easily as in the case of CFL.

We formulate two different pumping lemmas. Remarkably, with one of them we complete nicely the picture of pumping lemmas known for regular, context-free and commutative context-free languages.

As expected, the pumping lemmas appear to be useful tool for relating the expressive power of language classes, as we demonstrate in Section 7.

The pumping lemmas. The length of a word w is written $|w|$. To motivate our conditions we start by recalling the pumping scheme proposed for ${}_{\text{c}}\text{CFL}$ by [5].

(${}_{\text{c}}$CFL-PUMPING [5]) *There is a constant N such that if $w \in L$ with $|w| > N$ then there exist words x, y, s such that*

1. $w \in x(s\,||\,y)$,
2. $1 \leq |s| \leq N$, *and*

3. $\forall m \geq 0$, $xs^m y \in L$.[8]

Point 1 reads as: w is a concatenation of some prefix x and an interleaving of s and y. We define now two new conditions on a language L.

(SHUFFLE PUMPING) *There is a constant N such that if $w \in L$ with $|w| > N$ then there exist words x, y, z, s, t such that*

1. $w \in x((s(y||t))||z)$,
2. $1 \leq |s|, |syt| \leq N$, *and*
3. $\forall m \geq 0$, $xs^m yt^m z \in L$.

Point 1 reads as: there is some subword y' of w with $w \in x(y'||z)$ and $y' \in s(y||t)$.

(CONCAT. PUMPING) *There is a constant N such that if $w \in L$ with $|w| > N$ then there exist words x, y, z, s, t such that*

1. $w = xyz$,
2. $1 \leq |st| \leq N$, *and*
3. $\forall m \geq 0$, $xs^m yt^m z \in L$.

Call the words s, t *repeatable words*. The difference between the two conditions concentrates on the word y that separates the repeatable words in $xs^m yt^m z$. On one hand SHUFFLE PUMPING seems weaker as y is no more an infix of w, but an arbitrary subword (subsequence). On the other hand SHUFFLE PUMPING seems stronger as the length of y is bounded.

Lemma 2 *Every language $L \in {}_{\text{pc}}\text{CFL} \cup {}_{\text{shuffle}}\text{CFL}$ satisfies* SHUFFLE PUMPING.

As an example of application we provide now a proof missing in Section 4.

Proof of Lemma 1. Assume towards contradiction that $L = \{a^n b^n c^n : n \geq 1\}$ is in ${}_{\text{pc}}\text{CFL}$ or in ${}_{\text{shuffle}}\text{CFL}$ and apply Lemma 2. Observe that the two repeatable words s and t have necessarily jointly the same number of letters a, b and c. Thus one of them has to contains two different letters. Repeating this word twice leads to a contradiction. □

Lemma 3 *Every language $L \in {}^{\text{tr}}_{\text{pc}}\text{CFL} \cup {}_{\text{shuffle}}\text{CFL}$ satisfies* CONCAT. PUMPING.

Class ${}_{\text{pc}}\text{CFL}$ does not satisfy CONCAT. PUMPING, as witnessed by the language from Example 1. Moreover in CONCAT. PUMPING one can not bound the length of the word y.

Relating conditions. The condition SHUFFLE PUMPING is similar to the classical context-free pumping – the only difference is the words s, y, t and z are subwords, not necessarily infixes, of w. We claim it is an elegant completion of the pumping lemmas for regular languages (RL), context-free languages (CFL) and commutative context-free languages (${}_{\text{c}}\text{CFL}$) (see [5]). All of these lemmas may be characterized by the following two characteristics:

1. Are there one or two pumping positions?
2. Are repeatable words infixes or subwords a given word?

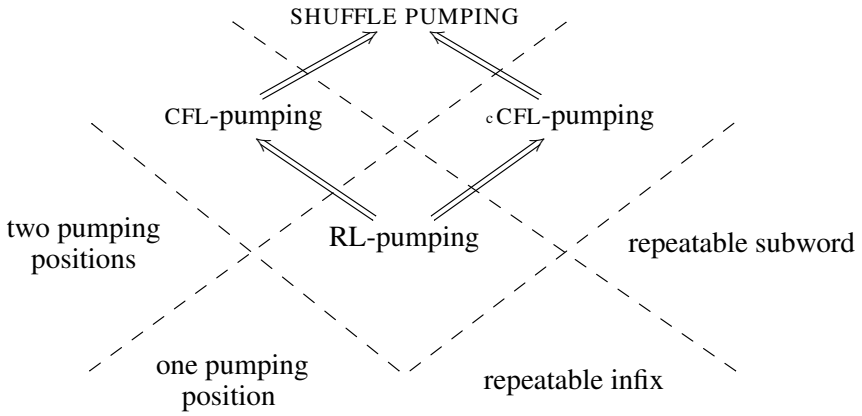
The known pumping lemmas have the following characteristics:

- RL: 1 pumping position, a repeatable word is an infix

[8] In fact in [5], the pumping scheme was $xs^m y'$, with a suffix y' of w (think of $y' \in s||y$), rather than $xs^m y$. The proofs of both are very similar. We discuss this issue further in Remark 1.

- CFL: 2 pumping positions, repeatable words are infixes
- $_{c}$CFL: 1 pumping position, a repeatable word is a subword [5].

In this light, our condition SHUFFLE PUMPING offers an elegant completion of the picture: 2 pumping positions, repeatable words are subwords. In other words, SHUFFLE PUMPING weakens $_{c}$CFL-pumping in the same way as CFL-pumping weakens RL-pumping (2 pumping positions instead of one). The other way around: SHUFFLE PUMPING weakens CFL-pumping in the same way as $_{c}$CFL-pumping weakens RL-pumping (repeatable word is no more an infix). The relationships between the four pumping conditions is depicted in the following diagram:



Remark 1 *It is worth mentioning that another pumping scheme could be used in place of* SHUFFLE PUMPING *in Lemma 2: instead of xs^myt^mz, one may consider*

$$xs^my't^mz,$$

with $w \in x(y' \,||\, z)$ and $y' \in (s(y \,||\, t))$. The proof would be very similar.

7 Expressiveness

Now we are ready to compare the expressive power of $^{\text{tr}}_{\text{pc}}$CFL and $_{\text{pc}}$CFL with other classes. We show that $^{\text{tr}}_{\text{pc}}$CFL is a strict subclass of $_{\text{pc}}$CFL and that both $_{\text{shuffle}}$CFL and $_{\text{trace}}$CFL are incomparable with either $_{\text{pc}}$CFL or $^{\text{tr}}_{\text{pc}}$CFL. More specifically, our results are as follows:

Theorem 3 $^{\text{tr}}_{\text{pc}}$CFL *is a strict subclass of* $_{\text{pc}}$CFL.

Theorem 4 *The following non-inclusions hold:*

(1) $^{\text{tr}}_{\text{pc}}$CFL $\cap$ $_{\text{shuffle}}$CFL *is not included in* $_{\text{trace}}$CFL.

(2) $^{\text{tr}}_{\text{pc}}$CFL $\cap$ $_{\text{trace}}$CFL *is not included in* $_{\text{shuffle}}$CFL.

Theorem 5 *The following non-inclusions hold:*

(1) $^{\text{tr}}_{\text{pc}}$CFL *is not included in* $_{\text{shuffle}}$CFL $\cup$ $_{\text{trace}}$CFL*;*

(2) $_{\text{shuffle}}$CFL *is not included in* $_{\text{pc}}$CFL $\cup$ $_{\text{trace}}$CFL*;*

(3) $_{\text{trace}}$CFL *is not included in* $_{\text{pc}}$CFL $\cup$ $_{\text{shuffle}}$CFL.

The proofs of the results are by identifying witnessing languages $L_1 \ldots L_6$, as illustrated in Figure 1. The pumping lemmas, namely Lemma 3 and Lemma 2, are sufficient to prove Theorem 3 and Theorem 5(3), respectively. On the other hand they are not sufficient for Theorem 4(2) and Theorem 5(1)–(2), as $_{\text{shuffle}}\text{CFL}$ satisfies both the lemmas, and thus we have to perform a more delicate analysis of a derivation tree. We illustrate the first kind of argument in the proof of Theorem 5(3) and the second kind in the proof of Theorem 5(2) below.

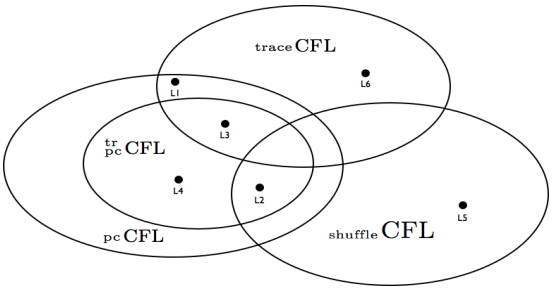


Figure 1: Relating the expressive power.

Proof of Theorem 5(3). Consider the language

$$L_6 = \{w \in \bigcup_{n\geq 0} \left(a^n \bar{a}\, d\, \bar{d}\, d^n \,||\, b^n c^n\right) : \text{every } b \text{ preceds every } d \text{ and } \bar{d} \text{ in } w\}. \tag{5}$$

Clearly, L_6 is the trace closure of the context-free language $\{(ab)^n \bar{a}\, \bar{d}\, (cd)^n : n \geq 0\}$, if for the independence on alphabet letters one chooses the symmetric closure of:

$$\{a, \bar{a}\} \times \{b, c\} \cup \{\bar{d}, d\} \times \{c\}.$$

Using Lemma 2 we will show that L_6 belongs to neither $_{\text{pc}}\text{CFL}$ nor $_{\text{shuffle}}\text{CFL}$. Consider a word

$$w_n = a^n \bar{a}\, b^n c^n \bar{d}\, d^n$$

and recall that for n larger that N of Lemma 2 we would obtain

$$w_n \in x\,(y' \,||\, z) \qquad y' \in s\,(y \,||\, t)$$

for a substring y' of w_n. Recall also the pumping scheme of SHUFFLE PUMPING from Lemma 2:

$$x s^m y t^m z \in L_6, \quad \text{for } m \geq 0. \tag{6}$$

We do a sequence of simple observations. First, to keep the same number of appearances of letters a, b, c and d, each of the four letters must appear either in s or t. Second, both s and t are necessarily non-empty as otherwise we would observe an illegal order of letters in (6), and moreover a and b occur in s and c and d occur in t, keeping in mind that in L_6 every a precedes every d and every b precedes every c and d. Third, the length of the prefix x is at most n, as otherwise both s and t would appear to the right of $\bar{a}$ and thus could not contain a. Thus, x contains only a. Now, $\overline{d}$ is not in x, cannot be in s or t, and cannot be in z since otherwise (s and) t could not contain d. Therefore $\overline{d}$ is in y, and z contains no b. As neither x nor z contains b, and $w_n \in x(y' \parallel z)$, y' must contain n occurrences of b, but $|y'| = |syt| \leq N$, hence this

is not possible. We have thus shown that L_6 does not satisfy Lemma 2 and therefore it does not belong to $_{\text{pc}}\text{CFL} \cup {}_{\text{shuffle}}\text{CFL}$. □

Proof of Theorem 4(2). Consider the language $L_3 \in {}^{\text{tr}}_{\text{pc}}\text{CFL}$:

$$L_3 = \bigcup_{n \geq 0} a^n s (b^n \,||\, c^n) \tag{7}$$

and a grammar that generates the language:

$$\begin{array}{llll} S \xrightarrow{a} SP & P \xrightarrow{b} C & C \xrightarrow{c} \varepsilon \\ S \xrightarrow{s} \varepsilon & P \xrightarrow{c} B & B \xrightarrow{b} \varepsilon \end{array}$$

The initial non-terminal is S and the threads are $\{S,P\},\{B\},\{C\}$. L_3 also belongs to $_{\text{trace}}\text{CFL}$ as it is the trace closure of the context-free language $\{a^n s(bc)^n : n \geq 0\}$ with independence $\{(b,c),(c,b)\}$.

It remains thus to show that $L_3 \notin {}_{\text{shuffle}}\text{CFL}$. Intuitively, the idea is to show that L_3 cannot benefit from parallel composition.

Assume that $L_3 \in {}_{\text{shuffle}}\text{CFL}$, aiming at deducing a contradiction. Fix a grammar that generates L_3. For simplicity think of the productions of the following form (the first two we will call *sequential*):

$$X \xrightarrow{a} \varepsilon \qquad X \xrightarrow{\varepsilon} Y;Z \qquad X \xrightarrow{\varepsilon} Y \,||\, Z.$$

We will exploit the property that s divides every word in L_3 into two separated regions. We partition the non-terminals into symbols that generate some word containing s, and symbols that do not; and call them s-symbols and non-s-symbols, respectively. By SUBSTITUTIVITY, each word generated by an s-symbol contains necessarily s.

Consider a derivation tree T of a word $wsv \in L_3$. The unique path leading from the root to the leaf labeled by s call *the spine*. Observe that an s-symbol may only appear on the spine and a non-s-symbol may only appear outside the spine. Knowing that the number of occurrences of a and b on both sides of the spine is the same, we deduce that

$$\text{each production labeling a node of the spine is necessarily sequential.} \tag{8}$$

Indeed, assume a parallel production $X \xrightarrow{\varepsilon} Y \,||\, Z$ labels a node of the spine. Wlog. let Y be a s-symbol. Let u, u' be the subwords induced by the Y-node and Z-node, respectively. Clearly there are two interleavings of u and u' such that the letter s, appearing in u, is placed in the interleaving in two different positions in the word u'. Thus at least one of these interleavings must lead to a violation of the condition (7) in a word belonging to L_3. Condition (8) is thus proved.

Now consider a non-s-symbol X appearing in T. The number of occurrences $\#_a(u)$ of a in all words u generated by X is necessarily the same, and the same applies to $\#_b(u)$ and $\#_c(u)$. Indeed, otherwise one gets a similar contradiction as above by considering two words induced by the X node, differing in the number of occurrences of a or b, and using SUBSTITUTIVITY. As a consequence X generates a finite language which may clearly be defined by a context-free grammar, say G_X.

If we apply the last observation to the very first non-s-symbol X on every path in T (except the spine), we obtain a tree without parallel nodes. As G_X does not depend on the particular derivation tree T chosen, and the word $wsv \in L_3$ was chosen arbitrary, we conclude that L_3 is generated by a context-free grammar. The grammar is obtained by replacing productions of every non-s-symbol X in G with G_X. As L is clearly not context-free we obtain a contradiction and thus complete the proof. □

References

[1] J. A. Bergstra & J. W. Klop (1985): *Algebra of Communicating Processes with Abstraction*. Theor. Comput. Sci. 37, pp. 77–121, doi:10.1016/0304-3975(85)90088-X.

[2] Jan A. Bergstra & Jan Willem Klop (1984): *Process Algebra for Synchronous Communication*. Information and Control 60(1-3), pp. 109–137, doi:10.1016/S0019-9958(84)80025-X.

[3] Jean Berstel, Luc Boasson, Olivier Carton, Jean-Eric Pin & Antonio Restivo (2010): *The expressive power of the shuffle product*. Inf. Comput. 208(11), pp. 1258–1272, doi:10.1016/j.ic.2010.06.002.

[4] Ahmed Bouajjani, Markus Müller-Olm & Tayssir Touili (2005): *Regular Symbolic Analysis of Dynamic Networks of Pushdown Systems*. In: CONCUR, pp. 473–487, doi:10.1007/11539452_36.

[5] S. Christensen (1993): *Decidability and Decomposition in Process Algebras*. Ph.D. thesis, Department of Computer Science, University of Edinburgh.

[6] W. Czerwiński, S. B. Fröschle & S. Lasota (2009): *Partially-Commutative Context-Free Processes*. In: CONCUR, pp. 259–273, doi:10.1007/978-3-642-04081-8_18.

[7] W. Czerwiński, S. B. Fröschle & S. Lasota (2011): *Partially-commutative context-free processes: expressibility and tractability*. Inf. Comput. 209(5), pp. 782–798, doi:10.1016/j.ic.2010.12.003.

[8] W. Czerwiński, P. Hofman & S. Lasota (2012): *Reachability problem for weak multi-pushdown automata*. To appear.

[9] J. Esparza (1997): *Petri Nets, Commutative Context-Free Grammars, and Basic Parallel Processes*. Fundam. Inform. 31(1), pp. 13–25, doi:10.1007/3-540-60249-6_54.

[10] Jay L. Gischer (1981): *Shuffle Languages, Petri Nets, and Context-Sensitive Grammars*. Commun. ACM 24(9), pp. 597–605, doi:10.1145/358746.358767.

[11] Y. Hirshfeld & M. Jerrum (1999): *Bisimulation Equivalence Is Decidable for Normed Process Algebra*. In: ICALP, pp. 412–421, doi:10.1007/3-540-48523-6_38.

[12] R. Mayr (2000): *Process Rewrite Systems*. Inf. Comput. 156(1-2), pp. 264–286, doi:10.1006/inco.1999.2826.

[13] A. W. Mazurkiewicz (1988): *Basic notions of trace theory*. In: REX Workshop, pp. 285–363.

[14] M.-J. Nederhof, G. Satta & S. Shieber (2003): *Partially Ordered Multiset Context-Free Grammars And Free-Word-Order Parsing*. In: Proc. 8th Intl Workshop on Parsing Technologies, pp. 171–182.

[15] J. Srba (2002): *Roadmap of Infinite Results*. Bulletin of the EATCS 78, pp. 163–175.