Star-Freeness, First-Order Definability and Aperiodicity of Structured Context-Free Languages

Dino Mandrioli[1], Matteo Pradella[1,2(✉)], and Stefano Crespi Reghizzi[1,2]

[1] Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Piazza Leonardo Da Vinci 32, 20133 Milano, Italy
{dino.mandrioli,matteo.pradella,stefano.crespireghizzi}@polimi.it
[2] IEIIT, Consiglio Nazionale delle Ricerche, via Ponzio 34/5, 20133 Milano, Italy

Abstract. A classic result in formal language theory is the equivalence among aperiodic finite automata, star-free regular expressions, and first-order logic on words. Extending these results to structured subclasses of context-free languages, such as tree languages, did not work as smoothly: there are star-free tree languages that are counting. We argue that investigating the same properties within the family of operator precedence languages (OPLs) by going back to string languages rather than tree languages may lead to equivalences that perfectly match those on regular languages. We define operator precedence expressions; we show that they define exactly the class of OPLs and that, when restricted to the star-free subclass, coincide with first-order definable OPLs and are aperiodic.

Since operator precedence languages strictly include other classes of structured languages such as visibly pushdown languages, the same results given in this paper hold as trivial corollary for that family too.

Keywords: Operator precedence languages · First-order logic · Monadic second-order logic · Star-free languages · Aperiodic languages · Input-driven languages · Visibly-pushdown languages

1 Introduction

It is well known that regular languages are closed w.r.t. all basic operations and are characterized also in terms of classic monadic second-order (MSO) logic [8,17,36], but the same properties do not hold for general context-free (CF) languages with the exception of *structured CF languages*. With this term we mean those various families of languages whose typical tree structure is immediately visible in their sentences: two first equivalent examples are *parenthesis languages* [27] and *tree languages* [34]. More recently, *input-driven languages (IDLs)* [7], later renamed *visibly pushdown languages (VPLs)* [2], *height-deterministic languages* [29] have also been shown to share many important properties of regular languages. Tree-languages and VPLs, in particular, are closed w.r.t. Boolean

© Springer Nature Switzerland AG 2020
V. K. I. Pun et al. (Eds.): ICTAC 2020, LNCS 12545, pp. 161–180, 2020.
https://doi.org/10.1007/978-3-030-64276-1_9

operations, concatenation, Kleene *, and are characterized in terms of some MSO logic, although such operations and the adopted logic language are rather different in the two cases. For a more complete analysis of structured languages and how they extend various properties of regular languages, see [26].

In this paper we are interested in an important subfamily of regular languages and its extension to various types of (structured) CF languages, namely *noncounting (NC)* or *aperiodic* languages. Intuitively, aperiodicity is a property of a recognizing device which prevents from separating strings that differ from each other by the number of repetitions of some substring, e.g. odd versus even. It is well-known [28] that NC regular languages coincide with those expressible by means of *star-free* regular expressions or by the first-order (FO) fragment of MSO. FO definability has a strong impact on the success of model-checking, thanks to the first-order completeness of linear temporal logic [32].

Various attempts have been done to extend the notion of aperiodicity beyond regular languages, specifically to some kind of structured CF languages. In facts, linguists as well as language designers have observed over and over that modulo-counting features are not present or needed either in natural or in technical languages.

NC parenthesis languages have been first introduced in [10]; then an equivalent definition thereof has been given in [35]. That paper, however, showed that the same equivalences holding for regular languages do not extend to tree languages: e.g., there are counting star-free tree languages. Further investigations (e.g., [18,21,22,31]) obtained partial results on special subclasses of the involved families but the complete set of equivalences was lost.

In this paper we pursue a different approach to achieve the goal of restating the above equivalences in the larger context of structured CF languages. In essence, we go back to string languages as opposed to the approaches based on tree languages but we choose a family of languages where the tree structure is somewhat implicit in the string, namely *operator precedence languages (OPL)*. OPLs have been invented by Floyd to support efficient deterministic parsing [19]. We classify them as "structured but semivisible" languages since their structure is implicitly assigned by *precedence relations* between terminal characters which were inspired by the precedence rules between arithmetic operations: as an early intuition for readers who are not familiar with OPLs, the expression $a + b \cdot c$ "hides" the parenthetic structure $a + (b \cdot c)$ which is implied by the fact that multiplicative operations should be applied before the additive ones. Subsequent investigations characterized OPLs as the largest known family of structured CFLs that is closed under all fundamental language operations and can be defined through a natural extension of the classic MSO logic [26].

Our new results on the relations between aperiodicity, star-freeness, and FO-definability of OPLs are the following. We define *Operator precedence expressions (OPE)*, as a simple extension of regular expressions and show that they define exactly OPLs. We prove closure properties of NC OPLs and derive therefrom that the languages defined by star-free OPEs are NC. We show that star-free

OPEs and FO formulas define exactly the same subfamily of OPLs. We conjecture that all NC OPLs are FO-definable.

2 Background

For brevity, we just list our notations for the basic concepts we use from formal language and automata theory. The terminal alphabet is denoted by Σ, and the empty string is ε. We also take for granted the traditional logic and set theoretic abbreviations. For a string x, $|x|$ denotes its length. The character $\#$, not present in the terminal alphabet, is used as string *delimiter*, and we define the alphabet $\Sigma_\# = \Sigma \cup \{\#\}$.

Definition 1 (Regular expression and language). *A* regular expression *(RE) over an alphabet Σ is a well-formed formula made with the characters of Σ, $\emptyset$, ε, the Boolean operators $\cup, \neg, \cap$, the concatenation $\cdot$, the Kleene star * and plus $^+$ operators. When neither * nor $^+$ are used, the RE is called* star-free *(SF). An RE E defines a language over Σ, denoted by $L(E)$. REs define the language family of* regular *languages.*

A regular language L over Σ is called noncounting *or* aperiodic *if there exists an integer $n \geq 1$ such that for all $x, y, z \in \Sigma^*$, $xy^n z \in L$ iff $xy^{n+m}z \in L$, $\forall m \geq 0$.*

Proposition 2. *The family of aperiodic regular languages coincides with the family of languages defined by star-free REs.*

Definition 3 (Grammar and language). *A (CF)* grammar *is a tuple $G = (\Sigma, V_N, P, S)$ where Σ and V_N, with $\Sigma \cap V_N = \emptyset$, are resp. the terminal and the nonterminal alphabets, the total alphabet is $V = \Sigma \cup V_N$, $P \subseteq V_N \times V^*$ is the rule (or production) set, and $S \subseteq V_N$, $S \neq \emptyset$, is the axiom set. For a generic rule $B \to \alpha$, where B and α are resp. called the left/right hand sides (lhs/rhs) the following forms are relevant:*

axiomatic*: $B \in S$,* terminal*: $\alpha \in \Sigma^+$,* empty*: $\alpha = \varepsilon$,*
renaming*: $\alpha \in V_N$,*
operator*: $\alpha \notin V^* V_N V_N V^*$,*
parenthesized*: $\alpha = (\!|\beta|\!)$, with $(\!|$, $|\!)$ new terminals.*

G is called backward deterministic *(or BD-grammar) if $(B \to \alpha, C \to \alpha \in P)$ implies $B = C$.*
If all rules of G are in operator form, G is called an operator grammar *or O-grammar.*
$\tilde{G} = \left(\Sigma \cup \{(\!|, |\!)\}, V_N, \tilde{P}, S\right)$ is a parenthesis *grammar (Par-grammar) if the rhs of every rule is parenthesized. $\tilde{G}$ is called the* parenthesized version *of G, if $\tilde{P}$ consists of all rules $B \to (\!|\beta|\!)$ such that $B \to \beta$ is in P. For brevity we take for granted the usual definition of derivation; the* language *defined by a grammar starting from a nonterminal A is $L_G(A) = \left\{x \in \Sigma^* \mid A \overset{*}{\underset{G}{\Longrightarrow}} x\right\}$. The subscript*

G will be omitted whenever clear from the context. The string x is derivable *from A and we call x a* sentence *if $A \in S$. The union of $L_G(A)$ for all $A \in S$ is the language $L(G)$ defined by G. Two grammars defining the same language are* equivalent*. Two grammars such that their parenthesized versions are equivalent, are* structurally equivalent*. The language generated by a Par-grammar is called a* parenthesis language*, and its sentences are well-parenthesized strings.*

From now on we only consider w.l.o.g. [5,20] BD-, unless otherwise stated, O-grammars without renaming rules, and without empty rules except, if the empty string is in the language, the axiomatic rule $B \to \varepsilon$ where B does not appear in the rhs of any rule.

Definition 4 (Backward deterministic reduced grammar [27,33]). *A* context *over an alphabet Σ is a string in $\Sigma^*\{-\}\Sigma^*$, where the character '$-$' $\notin \Sigma$ and is called a* blank*. We denote by $\alpha[x]$ the context α with its blank replaced by the string x. Two nonterminals B and C of a grammar G are termed* equivalent *if, for every context α, $\alpha[B]$ is derivable exactly in case $\alpha[C]$ is derivable (not necessarily from the same axiom). A nonterminal B is* useless *if there is no context α such that $\alpha[B]$ is derivable or B generates no terminal string. A terminal b is useless if it does not appear in any sentence of $L(G)$. A grammar is* clean *if it has no useless nonterminals and terminals. A grammar is* reduced *if it is clean and no two nonterminals are equivalent. A BDR-grammar is both backward deterministic and reduced.*

From [27], every parenthesis language is generated by a unique, up to an isomorphism of its nonterminal alphabet, Par-grammar that is BDR.

2.1 Operator-Precedence Languages

Intuitively, operator precedence grammars (OPG) are based on three precedence relations, called *equal*, *yield* and *take*, included in $\Sigma_\# \times \Sigma_\#$. For a given O-grammar, a character a is *equal in precedence* to b iff some rhs contains as substring ab or a string aBb, where B is a nonterminal; in fact, when evaluating the relations between terminal characters for OPG, nonterminals are "transparent". A character a *yields precedence* to b iff a can occur immediately to the left of a syntax subtree whose leftmost *terminal* character is b. Symmetrically, a *takes precedence* over b iff a can occur as the rightmost *terminal* character of a subtree and b is the immediately following terminal character.

Definition 5 [19]. *Let $G = (\Sigma, V_N, P, S)$ be an O-grammar. Let a, b denote elements in Σ, A, B in V_N, C in V_N or ε, and α, β range over $(V_N \cup \Sigma)^*$. The left and right terminal sets of nonterminals are respectively:*

$$\mathcal{L}_G(A) = \left\{ a \mid \exists C : A \overset{*}{\underset{G}{\Longrightarrow}} Ca\alpha \right\} \quad \text{and} \quad \mathcal{R}_G(A) = \left\{ a \mid \exists C : A \overset{*}{\underset{G}{\Longrightarrow}} \alpha a C \right\}.$$

(The grammar name will be omitted unless necessary to prevent confusion.)

The operator precedence relations *(OPRs) are defined over $\Sigma_\# \times \Sigma_\#$ as follows:*

- *equal in precedence:* $a \doteq b \iff \exists A \to \alpha a C b \beta \in P, \# \doteq \#$
- *takes precedence:* $a \gtrdot b \iff \exists A \to \alpha B b \beta \in P, a \in \mathcal{R}(B); a \gtrdot \# \iff a \in \mathcal{R}(B), B \in S$
- *yields precedence:* $a \lessdot b \iff \exists A \to \alpha a B \beta \in P, b \in \mathcal{L}(B); \# \lessdot b \iff b \in \mathcal{L}(B), B \in S$.

The OPRs can be collected into a $|\Sigma_\#| \times |\Sigma_\#|$ *array, called the* operator precedence matrix *of the grammar,* $\mathrm{OPM}(G)$*: for each (ordered) pair* $(a, b) \in \Sigma_\# \times \Sigma_\#$, $\mathrm{OPM}_{a,b}(G)$ *contains the OPRs holding between* a *and* b.

Consider a square matrix: $M = \{M_{a,b} \subseteq \{\doteq, \lessdot, \gtrdot\} \mid a, b \in \Sigma_\#\}$. Such an OPM matrix, is called *conflict-free* iff $\forall a, b \in \Sigma_\#$, $0 \leq |M_{a,b}| \leq 1$. A conflict-free matrix is called *total* or *complete* iff $\forall a, b \in \Sigma_\#$, $M_{a,b} \neq \emptyset$. A matrix is $\doteq$*-acyclic* if $\nexists a_i \in \Sigma$ such that $a_i \doteq \ldots \doteq a_i$.

In this we assume that an OPM is $\doteq$-acyclic. Such a hypothesis is stated for simplicity despite the fact that, rigorously speaking, it affects the expressive power of OPLs. It could be avoided if we adopted OPGs extended by the possibility of including regular expressions in production rhs [12,14], which however would require a much heavier notation.

We extend the set inclusion relations and the Boolean operations in the obvious cell by cell way, to any two matrices having the same terminal alphabet. Two matrices are *compatible* iff their union is conflict-free.

Definition 6 (Operator precedence grammar). *An O-grammar* G *is an* operator precedence *grammar (OPG) iff the matrix* $\mathrm{OPM}(G)$ *is conflict-free, i.e., the three OPRs are pairwise disjoint. Then the language generated by* G *is an* operator precedence language (OPL)*. An OPG is* $\doteq$-acyclic *if* $\mathrm{OPM}(G)$ *is so.*

It is known that the OPL family is strictly included within the deterministic and reverse-deterministic CF family and strictly includes the VPL one [12].

Example 7. For the grammar GAE_1 (see Fig. 1, left), the left and right terminal sets of nonterminals E, T and F are, respectively: $\mathcal{L}(E) = \{+, *, e\}$, $\mathcal{L}(T) = \{*, e\}$, $\mathcal{L}(F) = \{e\}$, $\mathcal{R}(E) = \{+, *, e\}$, $\mathcal{R}(T) = \{*, e\}$, and $\mathcal{R}(F) = \{e\}$.

Figure 1 (center) displays the conflict-free OPM associated with the grammar GAE_1; for instance $\mathrm{OPM}_{*,e} = \lessdot$ tells that $*$ yields precedence to e.

$S = \{E, T, F\}$
$E \to E + T \mid T * F \mid e$
$T \to T * F \mid e$
$F \to e$

	$+$	$*$	e	$\#$
$+$	$\gtrdot$	$\lessdot$	$\lessdot$	$\gtrdot$
$*$	$\gtrdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
e	$\gtrdot$	$\gtrdot$		$\gtrdot$
$\#$	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$

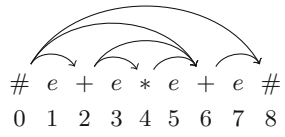


Fig. 1. GAE_1 (left) and its OPM (center); the string $e + e * e + e$, with relation $\curvearrowright$ (right).

Unlike the arithmetic relations having similar typography, the OPRs do not enjoy any of the transitive, symmetric, reflexive properties.

A conflict-free matrix associates to every string at most one structure, i.e., a unique parenthesization. This aspect, paired with a way of deterministically choosing rules' rhs to be reduced, are the basis of Floyd's natural bottom-up deterministic parsing algorithm. E.g., the following BD version of GAE_1 (axioms and OPM are unchanged) drives the parsing of the string $e + e * e + e$ to the unique structure $(\!(\!(\!(e)\!) + (\!(\!(e)\!) * (\!(e)\!)\!)\!) + (\!(e)\!)\!)$:

$$E \to E + T \mid E + F \mid T + T \mid F + F \mid F + T \mid T + F$$
$$T \to T * F \mid F * F \qquad F \to e.$$

Various formal properties of OPGs and languages are documented in the literature, chiefly in [12,13,26]. For convenience, we just recall and collect the ones that are relevant for this article in the next proposition.

Proposition 8. (Relevant properties of OPGs and OPLs).
Let M be a conflict-free OPM over $\Sigma_\# \times \Sigma_\#$.

1. *The class of OPGs and OPLs* compatible *with M are:*
   $\mathscr{C}_M = \{G \mid G \text{ is an OPG}, \mathrm{OPM}(G) \subseteq M\}$, $\mathscr{L}_M = \{L(G) \mid G \in \mathscr{C}_M\}$.
2. *The class $\mathscr{C}_M$ contains a unique grammar, called the* maxgrammar *of M, denoted by $G_{max,M}$, such that for all grammars $G \in \mathscr{C}_M$, the inclusion holds $L(G) \subseteq L(G_{max,M})$. $L(G_{\max,M})$ is called* max-language. *If M is total, then $L(G_{\max,M}) = \Sigma^*$.*
3. *Let M be total. With a natural overloading, we define the function $M : \Sigma^* \to (\Sigma \cup \{(\!|, |\!)\})^*$ as $M(x) = y$, if $A \xRightarrow[G_{max,M}]{*} x$, $A \xRightarrow[\tilde{G}_{max,M}]{*} y$ are corresponding derivations.*
   E.g. with M such that $a \lessdot a$, $a \doteq b$, $b \gtrdot b$, $M(aaaabbb) = (\!|a(\!|a(\!|a(\!|ab|\!)b|\!)b|\!)|\!)$.
4. *Let M be total.*
   - *$\mathscr{L}_M$ is closed under all set operations, therefore it is a Boolean algebra.*
   - *$\mathscr{L}_M$ is closed under concatenation and Kleene star.*

In summary, an OPM assigns a universal structure to strings in Σ^*, thus we call the pair (Σ, M) an *OP alphabet*. The following characterizations of OPLs, in terms of logic and expressions, are bound to the OP alphabet.

Logic Characterization. In [25] the traditional monadic second order logic (MSO) characterization of regular languages by Büchi, Elgot, and Trakhtenbrot [8,17,36] is extended to the case of OPL. To deal with the typical tree structure of CF languages the original MSO syntax is augmented with the predicate $\curvearrowright$, based on the OPL precedence relations: informally, $\boldsymbol{x} \curvearrowright \boldsymbol{y}$ holds between the rightmost and leftmost positions of the context encompassing a subtree, i.e., respectively, of the character that yields precedence to the subtree's leftmost leaf, and of the one over which the subtree's rightmost leaf takes precedence.

Unlike similar but simpler relations introduced, e.g., in [23] and [2], the $\curvearrowright$ relation is not one-to-one. For instance, Fig. 1 (right) displays the $\curvearrowright$ relation

holding for the sentence $e + e * e + e$ generated by grammar GAE_1: we have $0 \curvearrowright 2$, $2 \curvearrowright 4$, $4 \curvearrowright 6$, $6 \curvearrowright 8$, $2 \curvearrowright 6$, $0 \curvearrowright 6$, and $0 \curvearrowright 8$. Such pairs correspond to contexts where a reduce operation is executed during the parsing of the string (they are listed according to their execution order).

Formally, we define a countable infinite set of first-order variables $\boldsymbol{x}, \boldsymbol{y}, \ldots$ and a countable infinite set of monadic second-order (set) variables $\boldsymbol{X}, \boldsymbol{Y}, \ldots$. We adopt the convention to denote first and second-order variables in boldface font.

Definition 9 (Monadic Second-order Logic over (Σ, M)). *Let $\mathcal{V}_1$ be a set of first-order variables, and $\mathcal{V}_2$ be a set of second-order (or set) variables. The $MSO_{\Sigma,M}$ (*monadic second-order logic *over (Σ, M)) is defined by the following syntax (symbols Σ, M will be omitted unless necessary to prevent confusion), where $c \in \Sigma_{\#}$, $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{V}_1$, and $\boldsymbol{X} \in \mathcal{V}_2$:*

$$\varphi := c(\boldsymbol{x}) \mid \boldsymbol{x} \in \boldsymbol{X} \mid \boldsymbol{x} < \boldsymbol{y} \mid \boldsymbol{x} \curvearrowright \boldsymbol{y} \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists \boldsymbol{x}.\varphi \mid \exists \boldsymbol{X}.\varphi.$$

A MSO formula is interpreted over a (Σ, M) string w, with respect to assignments $\nu_1 : \mathcal{V}_1 \to \{0, 1, \ldots |w| + 1\}$ and $\nu_2 : \mathcal{V}_2 \to \wp(\{0, 1, \ldots |w| + 1\})$, in this way:

- $\#w\#, M, \nu_1, \nu_2 \models c(\boldsymbol{x})$ *iff* $\#w\# = w_1 c w_2$ *and* $|w_1| = \nu_1(\boldsymbol{x})$.
- $\#w\#, M, \nu_1, \nu_2 \models \boldsymbol{x} \in \boldsymbol{X}$ *iff* $\nu_1(\boldsymbol{x}) \in \nu_2(\boldsymbol{X})$.
- $\#w\#, M, \nu_1, \nu_2 \models \boldsymbol{x} < \boldsymbol{y}$ *iff* $\nu_1(\boldsymbol{x}) < \nu_1(\boldsymbol{y})$.
- $\#w\#, M, \nu_1, \nu_2 \models \boldsymbol{x} \curvearrowright \boldsymbol{y}$ *iff* $\#w\# = w_1 a w_2 b w_3$, $|w_1| = \nu_1(\boldsymbol{x})$, $|w_1 a w_2| = \nu_1(\boldsymbol{y})$, *and w_2 is the frontier of a subtree of the syntax tree of w.*
- $\#w\#, M, \nu_1, \nu_2 \models \neg\varphi$ *iff* $\#w\#, M, \nu_1, \nu_2 \not\models \varphi$.
- $\#w\#, M, \nu_1, \nu_2 \models \varphi_1 \vee \varphi_2$ *iff* $\#w\#, M, \nu_1, \nu_2 \models \varphi_1$ *or* $\#w\#, M, \nu_1, \nu_2 \models \varphi_2$.
- $\#w\#, M, \nu_1, \nu_2 \models \exists \boldsymbol{x}.\varphi$ *iff* $\#w\#, M, \nu_1', \nu_2 \models \varphi$, *for some ν_1' with* $\nu_1'(\boldsymbol{y}) = \nu_1(\boldsymbol{y})$ *for all* $\boldsymbol{y} \in \mathcal{V}_1 - \{\boldsymbol{x}\}$.
- $\#w\#, M, \nu_1, \nu_2 \models \exists \boldsymbol{X}.\varphi$ *iff* $\#w\#, M, \nu_1, \nu_2' \models \varphi$, *for some ν_2' with* $\nu_2'(\boldsymbol{Y}) = \nu_2(\boldsymbol{Y})$ *for all* $\boldsymbol{Y} \in \mathcal{V}_2 - \{\boldsymbol{X}\}$.

To improve readability, we drop M, ν_1, ν_2 and the delimiters $\#$ from the notation whenever there is no risk of ambiguity; furthermore we use some standard abbreviations in formulas, e.g., $\wedge$, $\forall$, $\boldsymbol{x}+1$, $\boldsymbol{x}-1$, $\boldsymbol{x} = \boldsymbol{y}$, $\boldsymbol{x} \leq \boldsymbol{y}$.

A sentence *is a formula without free variables. The language of all strings $w \in \Sigma^*$ such that $w \models \varphi$ is $L(\varphi) = \{w \in \Sigma^* \mid w \models \varphi\}$.*

The above MSO logic describes exactly the OPL family [25]. We denote the restriction of the MSO logic to the first-order as FO. Whenever we deal with logic definition of languages we implicitly exclude from such languages the empty string, according with the traditional convention adopted in the literature (e.g., [28]); thus, when talking about MSO or FO definable languages we exclude empty rules from their grammars.

2.2 Parenthesis and OPLs and the Noncounting Property

We briefly recall the standard definition of the NC property for CF parenthesis languages [10] and its decidability, then we apply it to OPLs.

Definition 10 (NC parenthesis language and grammar [10]**).** *A parenthesis language L is NC (or* aperiodic*) iff $\exists n > 1$ such that, for all strings x, u, w, v, y in $(\Sigma \cup \{(\!|, |\!)\})^*$ where w and uwv are well parenthesized, $xu^n wv^n y \in L$ iff $xu^{n+m} wv^{n+m} y \in L$, $\forall m \geq 0$.*

A derivation *of a Par-grammar is* counting *iff it has the form $B \overset{*}{\Longrightarrow} u^m B v^m$, with $m > 1$, and there is not a derivation $B \overset{*}{\Longrightarrow} uBv$. A Par-grammar is NC iff none of its derivations is counting.*

The next proposition ensures the decidability of the NC property.

Theorem 11. *[NC language and grammar (Th. 1 of [10])] A parenthesis language is NC iff its BDR grammar has no counting derivation.*

Definition 12 (NC OPLs and grammars). *For a given OPL L with OPM M, L_p is the language of the parenthesized strings x_p uniquely associated to L's strings x by M. An OPL L is NC iff its corresponding parenthesized language L_p is NC.*

A derivation of a grammar G is counting iff the corresponding derivation of the associated Par-grammar G_p is counting.

Thus, an OPL is NC iff its BDR OPG (unique up to an isomorphim of nonterminal alphabets) has no counting derivations.

In the following, unless parentheses are explicitly needed, we refer to unparenthesized strings since their correspondence to parenthesized strings is one-to-one. It is also worth recalling [11] the following peculiar property of OPLs: such languages are NC or not independently on their OPM, in other words, although the NC property is defined for structured languages (parenthesis or tree languages [27,34]), in the case of OPLs this property does not depend on the structure given to the sentences by the OPM.

It is important to stress, however, that, despite the above peculiarity of OPLs, aperiodicity remains a property that makes sense only with reference to the structured version of languages. Consider the following languages, with the same OPM consisting of $\{c \lessdot c, c \doteq a, c \doteq b, a \gtrdot b, b \gtrdot a\}$ besides the implicit relations w.r.t. #: $L_1 = \{c^{2n}(ab)^n \mid n \geq 1\}$, $L_2 = (ab)^+$. They are both clearly NC and so is their concatenation $L_1 \cdot L_2$, according to Definition 12, which in its parenthesized version is $\{(\!|^{2(m-n)}((\!|c)^{2n}(a|\!)b|\!))^m \mid m > n \geq 1\}$, (see also Theorem 20); however, if we applied Definition 10 to $L_1 \cdot L_2$ without considering parentheses, we would obtain that, for every n, $c^{2n}(ab)^{2n} \in L_1 \cdot L_2$ but not so for $c^{2n+1}(ab)^{2n+1}$.

3 Expressions for OPLs

Operator Precedence Expressions (OPE) extend traditional REs in a similar way as in other cases such as, e.g., REs for tree-languages [35]. We show that OPEs

define exactly the OPL family, so that, by joining this result with the previous characterizations of OPL in terms of MSO definability and recognizability by *Operator Precedence Automata (OPA)* [25], we have that *the class OPL can be defined equivalently as the class of languages generated by OPGs, or described through MSO formulas, or recognized by OPAs, or defined by OPEs.*

As well as for MSO logic and OPA, OPE's definition is based on an OP alphabet.

Definition 13 (OPE). *Given an OP alphabet (Σ, M), where M is complete, an OPE E and its language $L_M(E) \subseteq \Sigma^*$ are defined as follows. The meta-alphabet of OPE uses the same symbols as REs, together with the two symbols '$[$', and '$]$'. Let E_1 and E_2 be OPEs:*

1. *$a \in \Sigma$ is an OPE with $L_M(a) = a$.*
2. *$\neg E_1$ is an OPE with $L_M(\neg E_1) = \Sigma^* - L_M(E_1)$.*
3. *$a[E_1]b$, called the* fence *operation, i.e., we say E_1 in the fence a, b, is an OPE with*
   if $a, b \in \Sigma$: $L_M(a[E_1]b) = a \cdot \{x \in L_M(E_1) \mid M(a \cdot x \cdot b) = (\!|a \cdot M(x) \cdot b|\!)\} \cdot b$
   if $a = \#, b \in \Sigma$: $L_M(\#[E_1]b) = \{x \in L_M(E_1) \mid M(x \cdot b) = (\!|M(x) \cdot b|\!)\} \cdot b$
   if $a \in \Sigma, b = \#$: $L_M(a[E_1]\#) = a \cdot \{x \in L_M(E_1) \mid M(a \cdot x) = (\!|a \cdot M(x)|\!)\}$
   where E_1 must not contain $\#$.
4. *$E_1 \cup E_2$ is an OPE with $L_M(E_1 \cup E_2) = L_M(E_1) \cup L_M(E_2)$.*
5. *$E_1 \cdot E_2$ is an OPE with $L_M(E_1 \cdot E_2) = L_M(E_1) \cdot L_M(E_2)$, where E_1 does not contain $a[E_3]\#$ and E_2 does not contain $\#[E_3]a$, for some OPE E_3, and $a \in \Sigma$.*
6. *E_1^* is an OPE defined by $E_1^* := \bigcup_{n=0}^{\infty} E_1^n$, where $E_1^0 := \{\varepsilon\}$, $E_1^1 = E_1$, $E_1^n := E_1^{n-1} \cdot E_1$; $E_1^+ := \bigcup_{n=1}^{\infty} E_1^n$.*

Among the operations defining OPEs, concatenation has the maximum precedence; set-theoretic operations have the usual precedences, the fence operation is dealt with as a normal parenthesis pair. A star-free (SF) OPE *is one that does not use * and $^+$.*

The conditions on # are due to the peculiarities of OPLs closure w.r.t. concatenation (see also Theorem 20). In 5. the # is not permitted within, say, the left factor E_1 because delimiters are necessarily positioned at the two ends of a string.

Besides the usual abbreviations for set operations (e.g., $\cap$ and $-$), we also use the following derived operators: $a \Delta b := a[\Sigma^+]b$, and $a \nabla b := \neg(a \Delta b) \cap a \cdot \Sigma^+ \cdot b$. It is trivial to see that the identity $a[E]b = a \Delta b \cap a \cdot E \cdot b$ holds.

The fact that in Definition 13 matrix M is complete is w.l.o.g.: to state that for two terminals a and b, $M_{a,b} = \emptyset$ (i.e. that there should be a "hole" in the OPM for them), we can use the short notations: $\text{hole}(a, b) := \neg(\Sigma^*(ab \cup a \Delta b)\Sigma^*)$, $\text{hole}(\#, b) := \neg(\# \Delta b \Sigma^*)$, $\text{hole}(a, \#) := \neg(\Sigma^* a \Delta \#)$ and intersect them with the OPE.

The following examples illustrate the meaning of the fence operation, the expressiveness of OPLs w.r.t. less powerful classes of CF languages, and how OPEs naturally extend REs to the OPL family.

Example 14. Let Σ be $\{a, b\}$, $\{a \lessdot a, a \doteq b, b \gtrdot b\} \subseteq M$. The OPE $a[a^*b^*]b$ defines the language $\{a^n b^n \mid n \geq 1\}$. In fact the fence operation imposes that any string $x \in a^*b^*$ embedded within the context a, b is well-parenthesized according to M.

The OPEs $a[a^*b^*]\#$ and $a^+a[a^*b^*]b \cup \{a^+\}$, instead, both define the language $\{a^n b^m \mid n > m \geq 0\}$ since the matrix M allows for, e.g., the string *aaabb* parenthesized as $\langle\!|a\langle\!|a\langle\!|ab|\!\rangle b|\!\rangle|\!\rangle$.

It is also easy to define Dyck languages with OPEs, as their parenthesis structure is naturally encoded by the OPM. Consider L_{Dyck} the Dyck language with two pairs of parentheses denoted by a, a' and b, b'. This language can be described simply through an incomplete OPM, reported in Fig. 2 (left). In other words it is $L_{\text{Dyck}} = L(G_{max,M})$ where M is the matrix of the figure. Given that, for technical simplicity, we use only complete OPMs, we must refer to the one in Fig. 2 (center), and state in the OPE that some OPRs are not wanted, such as a, b', where the open and closed parentheses are of the wrong kind, or $a, \#$, i.e. an open a must have a matching a'.

	a	a'	b	b'	$\#$
a	$\lessdot$	$\doteq$	$\lessdot$		
a'	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$	$\gtrdot$
b	$\lessdot$		$\lessdot$	$\doteq$	
b'	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$	$\gtrdot$
$\#$	$\lessdot$		$\lessdot$		$\doteq$

	a	a'	b	b'	$\#$
a	$\lessdot$	$\doteq$	$\lessdot$	$\gtrdot$	$\gtrdot$
a'	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$	$\gtrdot$
b	$\lessdot$	$\gtrdot$	$\lessdot$	$\doteq$	$\gtrdot$
b'	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$	$\gtrdot$
$\#$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$

	call	*ret*	*int*	$\#$
call	$\lessdot$	$\doteq$	$\gtrdot$	
ret	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
int	$\gtrdot$		$\gtrdot$	$\gtrdot$
$\#$	$\lessdot$		$\lessdot$	$\doteq$

Fig. 2. The incomplete OPM defining L_{Dyck} (left), a possible completion M_{complete} (center), and the OPM M_{int} for the OPE describing an interrupt policy (right).

The following OPE defines L_{Dyck} by suitably restricting the "universe" $L(G_{max,M_{\text{complete}}})$:

$$\text{hole}(a, b') \cap \text{hole}(b, a') \cap \text{hole}(\#, a') \cap \text{hole}(\#, b') \cap \text{hole}(a, \#) \cap \text{hole}(b, \#).$$

Example 15. For a more application-oriented case, consider the classical LIFO policy managing procedure calls and returns but assume also that interrupts may occur: in such a case the stack of pending calls is emptied and computation is resumed from scratch.

This policy is already formalized by the incomplete OPM of Fig. 2 (right), with $\Sigma = \{call, ret, int\}$ with the obvious meaning of symbols. For example, the string *call call ret call call int* represents a run where only the second call returns, while the other ones are interrupted. On the contrary, *call call int ret* is forbidden, because a return is not allowed when the stack is empty. If we further want to say that there must be at least one terminating procedure, we can use the OPE: $\Sigma^* \cdot call \Delta ret \cdot \Sigma^*$.

Another example is the following, where we state that the run must contain at least one sub-run where no procedures are interrupted: $\Sigma^* \cdot \text{hole}(call, int) \cdot \Sigma^*$.

Notice that the language defined by the above OPE is not a VPL since VPLs only allow for unmatched returns and calls at the beginning or at the end of a string, respectively.

Theorem 16. *For every OPE E on an OPM M, there is an OPG G, compatible with M, such that $L_M(E) = L(G)$.*

Proof. By induction on E's structure. The operations $\cup, \neg, \cdot$, and * come from the closures of OPLs (Proposition 8). The only new case is $a[E]b$ which is given by the following grammar. The function $\eta : \Sigma_\# \to \Sigma$, such that $\eta(\#) = \varepsilon$, $\eta(a) = a$ otherwise, is used to take borders into account. If, by induction, G defines the same language as E, with axiom set S_E, then build the grammar G' from G by adding the following rules, where A and A' are new nonterminals of G' not in G, A is an axiom of G', and $B \in S_E$:

- $A \to \eta(a)B\eta(b)$, if $a \doteq b$ in M;
- $A \to \eta(a)A'$ and $A' \to B\eta(b)$, if $a \lessdot b$ in M;
- $A \to A'\eta(b)$ and $A' \to \eta(a)B$, if $a \gtrdot b$ in M.

The grammar for $a[E]b$ is then obtained by applying the construction for $L(G') \cap L(G_{max,M})$. This intersection is to check that $a \lessdot \mathcal{L}(B)$ and $\mathcal{R}(B) \gtrdot b$; if it is not the case, according to the semantics of $a[E]b$, the resulting language is empty. □

Next, we show that OPEs can express any language that is definable through an MSO formula as defined in Sect. 2.1. Thanks to the fact that the same MSO logic can express exactly OPLs [25] and to Theorem 16 we obtain our first major result, i.e., the equivalence of MSO, OPG, OPA (see e.g., [26]), and OPE.

To construct an OPE from a given MSO formula we follow the traditional path adopted for regular languages (as explained, e.g., in [30]) and augment it to deal with the $\curvearrowright$ relation. For a MSO formula φ, let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_r$ be the set of first-order variables occurring in φ, and $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_s$ be the set of second order variables. We use the new alphabet $B_{p,q} = \Sigma \times \{0,1\}^p \times \{0,1\}^q$, where $p \geq r$ and $q \geq s$. The main idea is that the $\{0,1\}^p$ part of the alphabet is used to encode the value of the first-order variables (e.g. for $p = r = 4$, $(1,0,1,0)$ stands for both the positions $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$), while the $\{0,1\}^q$ part of the alphabet is used for the second order variables. Hence, we are interested in the language $K_{p,q}$ formed by all strings where the components encoding the first-order variables contain exactly one occurrence of 1. We also use this definition $C_k := \{c \in B_{p,q} \mid \text{ the } (k+1)\text{-st component of } c = 1\}$.

Theorem 17. *For every MSO formula φ on an OP alphabet (Σ, M) there is an OPE E on M such that $L_M(E) = L(\varphi)$.*

Proof. By induction on φ's structure; the construction is standard for regular operations, the only difference is $\boldsymbol{x}_i \curvearrowright \boldsymbol{x}_j$. $B_{p,q}$ is used to encode interpretations of free variables. The set $K_{p,q}$ of strings where each component encoding a first-order variable is such that there exists only one 1, is given

by: $K_{p,q} = \bigcap_{1 \le i \le p}(B^*_{p,q}C_iB^*_{p,q} - B^*_{p,q}C_iB^*_{p,q}C_iB^*_{p,q})$. Disjunction and negation are naturally translated into $\cup$ and $\neg$; like in Büchi's theorem, for OPE $\exists \boldsymbol{x}_i \psi$ (resp. $\exists \boldsymbol{X}_j \psi$), first the expression E_ψ for ψ on an alphabet $B_{p,q}$ is built, then E for $\exists \boldsymbol{x}_i \psi$ is obtained from E_ψ by erasing the component i (resp. j) from $B_{p,q}$; $\boldsymbol{x}_i < \boldsymbol{x}_j$ is represented by $K_{p,q} \cap B^*_{p,q}C_iB^*_{p,q}C_jB^*_{p,q}$. Last, the OPE for $\boldsymbol{x}_i \curvearrowright \boldsymbol{x}_j$ is: $B^*_{p,q}C_i[B^+_{p,q}]C_jB^*_{p,q}$. □

4 Closure Properties of Noncounting OPLs and Star-Free OPEs

Thanks to the fact that an OPM implicitly defines the structure of an OPL, i.e., its parenthesization, aperiodic OPLs inherit from the general class the same closure properties w.r.t. the basic algebraic operations. Such properties are proved in this section under the same assumption as in the general case (see Proposition 8), i.e., that *the involved languages have compatible OPMs*. As a major consequence we derive that star-free OPEs define aperiodic OPLs.

Theorem 18. *Counting and NC parenthesis languages are closed w.r.t. complement. Thus, for any OPM M, counting and NC OPLs in the family $\mathscr{L}_M$ (Proposition 8) are closed w.r.t. complement w.r.t. the max-language defined by M.*

Proof. We give the proof for counting languages which also implies the closure of NC ones.

By definition of counting parenthesis language and from Theorem 11, if L is counting there exist strings x, u, v, z, y and integers n, m with $n > 1, m > 1$ such that $xv^{n+r}zu^{n+r}y \in L$ for all $r = km > 0$, $k > 0$, but not for all $r > 0$. Thus, the complement of L contains infinitely many strings $xv^{n+i}zu^{n+i}y \in L$ but not all of them since for some i, $i = km$. Thus, for $\neg L$ too there is no n such that $xv^nzu^ny \in L$ iff $xv^{n+r}zu^{n+r}y \in L$ for all $r \ge 0$. □

Theorem 19. *NC parenthesis languages and NC OPLs in the same family $\mathscr{L}_M$ are closed w.r.t. union and therefore w.r.t. intersection.*

Proof. Let L_1, L_2 be two NC parenthesis languages (resp. OPLs). Assume by contradiction that $L = L_1 \cup L_2$ is counting. Thus, there exist strings x, u, v, z, y such that for infinitely many m, $xv^mzu^my \in L$ but for no n $xv^nzu^ny \in L$ iff $xv^{n+r}zu^{n+r}y \in L$ for all $r \ge 0$. Hence, the same property must hold for at least one of L_1 and L_2 which therefore would be counting. □

Notice that, unlike the case of complement, counting languages are not closed w.r.t. union and intersection, whether they are regular or parenthesis or OPLs.

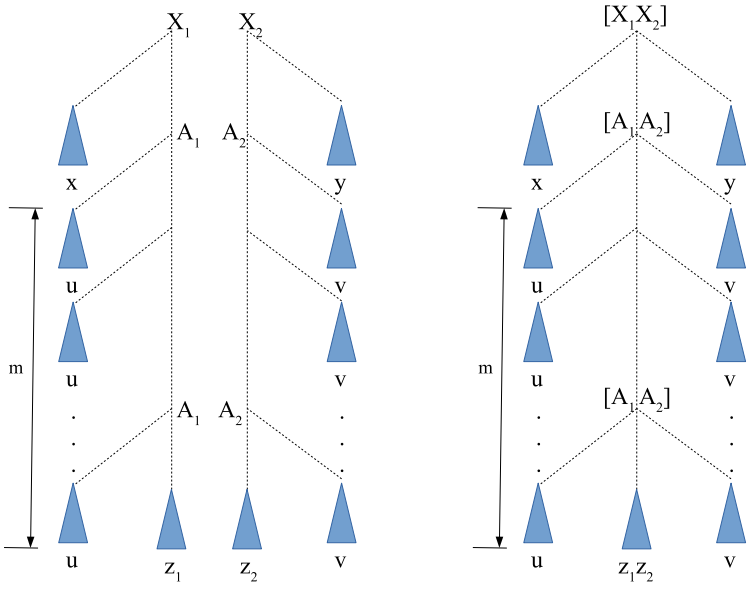


Fig. 3. An example of paired derivations combined by the concatenation construction. In this case the last character of u is in $\doteq$ relation with the first character of v.

Theorem 20. *NC OPLs are closed w.r.t. concatenation.*

Proof. Let $L_i = L(G_i)$, $i = 1, 2$, be NC OPLs with $G_i = (\Sigma, V_{Ni}, P_i, S_i)$ BDR OPGs. Let also $L_{pi} = L(G_{pi})$ be the parenthesized languages and grammars. We exploit the proof in [12] that OPLs with compatible OPM are closed w.r.t. concatenation. In general the parenthesized version L_p of $L = L_1 \cdot L_2$ is not the parenthesized concatenation of the parenthesized versions of L_1 and L_2, i.e., L_p may differ from $(\!|L'_{p1} \cdot L'_{p2}|\!)$, where $(\!|L'_{p1}|\!) = L_{p1}$ and $(\!|L'_{p2}|\!) = L_{p2}$, because the concatenation may cause the syntax trees of L_1 and L_2 to coalesce.

The construction given in [12] builds a grammar G whose nonterminal alphabet includes V_{N1}, V_{N2} and a set of pairs $[A_1, A_2]$ with $A_1 \in V_{N1}$, $A_2 \in V_{N2}$; the axioms of G are the pairs $[X_1, X_2]$ with $X_1 \in S_1$, $X_2 \in S_2$.[1] In essence (Lemmas 18 through 21 of [12]) G's derivations are such that $[X_1, X_2] \overset{*}{\underset{G}{\Longrightarrow}} x[A_1, A_2]y$, $[A_1, A_2] \overset{*}{\underset{G}{\Longrightarrow}} u$ implies $u = w \cdot z$ for some w, z and $X_1 \overset{*}{\underset{G_1}{\Longrightarrow}} xA_1$, $A_1 \overset{*}{\underset{G_1}{\Longrightarrow}} w$, $X_2 \overset{*}{\underset{G_2}{\Longrightarrow}} A_2 y$, $A_2 \overset{*}{\underset{G_2}{\Longrightarrow}} z$. Notice that some substrings of $x \cdot w$, resp. $z \cdot y$, may be derived from nonterminals belonging to V_{N1}, resp. V_{N2}, as the consequence of rules of type $[A_1, A_2] \to \alpha_1[B_1, B_2]\beta_2$ with $\alpha_1 \in V_1^*$, $\beta_2 \in V_2^*$, where $[B_1, B_2]$

[1] This is a minor deviation from [12], where it was assumed that grammars have only one axiom.

could be missing; also, any string γ derivable in G contains at most one nonterminal of type $[A_1, A_2]$.[2]

Suppose, by contradiction, that G has a counting derivation[3] $[X_1, X_2] \underset{G}{\overset{*}{\Longrightarrow}} x[A_1, A_2]y \underset{G}{\overset{*}{\Longrightarrow}} xu^m[A_1, A_2]v^m y \underset{G}{\overset{*}{\Longrightarrow}} xu^m z v^m y$ (one of u^m, v^m could be empty either in L or in L_p) whereas $[A_1, A_2]$ does not derive $u[A_1, A_2]v$: this would imply the derivations $A_1 \underset{G_1}{\overset{*}{\Longrightarrow}} u^m A_1$, $A_2 \underset{G_2}{\overset{*}{\Longrightarrow}} A_2 v^m$ which would be counting in G_1 and G_2 since they would involve the same nonterminals in the pairs $[A_i, A_j]$. If instead the counting derivation of G were derived from nonterminals belonging to V_{N1}, (resp. V_{N2}) that derivation would exist identical for G_1 (resp. G_2). Figure 3 shows a counting derivation of G derived by the concatenation of two counting derivations of G_1 and G_2; in this case neither u^m nor v^m are empty. □

Theorem 21. *The OPLs defined through star-free OPEs are NC.*

Proof. We need to show that if the language defined by the SF expression E is NC, so is the language defined by $a[E]b$. This follows by the identity $a[E]b = a\Delta b \cap aEb = a[\Sigma^+]b \cap aEb$. □

5 FO-Definable OPLs and SF OPEs

We now prove that SF OPE-definable languages coincide with FO-definable OPLs which are therefore NC as well; a result in sharp contrast with the negative results for NC tree-languages [35]. In Sect. 5.1 we show that NC linear OPLs are FO-definable too.

Lemma 22 (Flat Normal Form). *Any star-free OPE can be written in the following form, called* flat normal form: $\bigcup_i \bigcap_j t_{i,j}$, *where the elements* $t_{i,j}$ *have either the form* $L_{i,j}a_{i,j}\Delta b_{i,j}R_{i,j}$, *or* $L_{i,j}a_{i,j}\nabla b_{i,j}R_{i,j}$, *or* $H_{i,j}$, *for* $a_{i,j}$, $b_{i,j} \in \Sigma_{\#}$, *and* $L_{i,j}$, $R_{i,j}$, $H_{i,j}$ *star-free REs.*

Proof. The lemma is a consequence of the distributive and De Morgan properties, together with the following identities, where $\circ_1, \circ_2 \in \{\Delta, \nabla\}$, and L_k, $1 \leq k \leq 3$ are star-free REs:

$$a[E]b = a\Delta b \cap aEb$$

$$L_1 a_1 \circ_1 a_2 L_2 a_3 \circ_2 a_4 L_3 = (L_1 a_1 \circ_1 a_2 L_2 a_3 \Sigma^+ a_4 L_3) \cap (L_1 a_1 \Sigma^+ a_2 L_2 a_3 \circ_2 a_4 L_3)$$

$$\neg(L_1 a_1 \Delta a_2 L_2) = L_1 a_1 \nabla a_2 L_2 \cup \neg(L_1 a_1 \Sigma^+ a_2 L_2)$$

$$\neg(L_1 a_1 \nabla a_2 L_2) = L_1 a_1 \Delta a_2 L_2 \cup \neg(L_1 a_1 \Sigma^+ a_2 L_2)$$

[2] See Fig. 3.

[3] Note that the G produced by the construction is BD if so are G_1 and G_2, but it could be not necessarily BDR; however, if a BDR OPG has a counting derivation, any equivalent BD grammar has also a counting derivation.

The first two identities are immediate, while the last two are based on the idea that the only non-regular constraints of the left-hand negations are respectively $a_1 \nabla a_2$ or $a_1 \Delta a_2$, that represent strings that are not in the set only because of their structure. □

Theorem 23. *For every FO formula φ on an OP alphabet (Σ, M) there is a star-free OPE E on M such that $L_M(E) = L(\varphi)$.*

Proof. Consider the formula φ, and its set of first-order variables: like in Sect. 3, $B_p = \Sigma \times \{0,1\}^p$ (the q components are absent, φ being a first-order formula), and the set K_p of strings where each component encoding a variable is such that there exists only one 1.

First, K_p is star-free: $K_p = \bigcap_{1 \leq i \leq p} (B_p^* C_i B_p^* - B_p^* C_i B_p^* C_i B_p^*)$.

Disjunction and negation are naturally translated into $\cup$ and $\neg$; $\boldsymbol{x}_i < \boldsymbol{x}_j$ is covered by the star-free OPE $K_p \cap B_p^* C_i B_p^* C_j B_p^*$. The $\boldsymbol{x}_i \curvearrowright \boldsymbol{x}_j$ formula is like in the second order case, i.e. is translated into $B_p^* C_i [B_p^+] C_j B_p^*$, which is star-free.

For the existential quantification, the problem is that star-free (OP and regular) languages are not closed under projections. Like in the regular case, the idea is to leverage the encoding of the evaluation of first-order variables, because there is only one position in which the component is 1 (see K_p), to use the bijective renamings $\pi_0(a, v_1, v_2, ..., v_{p-1}, 0) = (a, v_1, v_2, ..., v_{p-1})$, and $\pi_1(a, v_1, v_2, ..., v_{p-1}, 1) = (a, v_1, v_2, ..., v_{p-1})$, where the last component is the one encoding the quantified variable. Notice that the bijective renaming does not change the Σ component of the symbol, thus maintaining all the OPRs.

Let E_φ be the star-free OPE on the alphabet B_p for the formula φ, with x a free variable in it. Let us assume w.l.o.g. that the evaluation of x is encoded by the last component of B_p; let $B = \Sigma \times \{0,1\}^{p-1} \times \{0\}$, and $A = \Sigma \times \{0,1\}^{p-1} \times \{1\}$.

The OPE for $\exists x \varphi$ is obtained from the OPE for φ through the bijective renaming π, and considering all the cases in which the symbol from A can occur.

First, let E' be an OPE in flat normal form, equivalent to E_φ (Lemma 22). The FO semantics is such that $L(\varphi) = L_M(E') = L_M(E') \cap B^* A B^*$.

By construction, E' is a union of intersections of elements $L_{i,j} a_{i,j} \Delta b_{i,j} R_{i,j}$, or $L_{i,j} a_{i,j} \nabla b_{i,j} R_{i,j}$, or $H_{i,j}$, where $a_{i,j}, b_{i,j} \in \Sigma$, and $L_{i,j}$, $R_{i,j}$, $H_{i,j}$ are star-free regular languages.

In the intersection of E' and $B^* A B^*$, all the possible cases in which the symbol in A can occur in E''s terms must be considered: e.g. in $L_{i,j} a_{i,j} \Delta b_{i,j} R_{i,j}$ it could occur in the $L_{i,j}$ prefix, or in $a_{i,j} \Delta b_{i,j}$, or in $R_{i,j}$. More precisely, $L_{i,j} a_{i,j} \Delta b_{i,j} R_{i,j} \cap B^* A B^* = (L_{i,j} \cap B^* A B^*) a_{i,j} \Delta b_{i,j} R_{i,j} \cup L_{i,j}$ $(a_{i,j} \Delta b_{i,j} \cap B^* A B^*)$ $R_{i,j} \cup L_{i,j} a_{i,j} \Delta$ $b_{i,j} (R_{i,j} \cap B^* A B^*)$ (the ∇ case is analogous, $H_{i,j}$ is immediate, being regular star-free).

The cases in which the symbol from A occurs in $L_{i,j}$ or $R_{i,j}$ are easy, because they are by construction regular star-free languages, hence we can use one of the standard regular approaches found in the literature (e.g. by using the *splitting lemma* in [16]). The only differences are in the factors $a_{i,j} \Delta b_{i,j}$, or $a_{i,j} \nabla b_{i,j}$.

Let us consider the case $a_{i,j} \Delta b_{i,j} \cap B^* A B^*$. The cases $a_{i,j} \in A$ or $b_{i,j} \in A$ are like $(L_{i,j} \cap B^* A B^*)$ and $(R_{i,j} \cap B^* A B^*)$, respectively, because $L_{i,j} a_{i,j}$ and $b_{i,j} R_{i,j}$ are also regular star-free (∇ is analogous).

The remaining cases are $a_{i,j} \Delta b_{i,j} \cap B^+AB^+$ and $a_{i,j} \nabla b_{i,j} \cap B^+AB^+$. By definition of Δ, $a_{i,j} \Delta b_{i,j} \cap B^+AB^+ = a_{i,j}[B^*AB^*]b_{i,j}$, and its bijective renaming is $\pi_0(a_{i,j})[\pi_0(B^*)\pi_1(A)\ \pi_0(B^*)]\pi_0(b_{i,j}) = a'_{i,j}[B^+_{p-1}]b'_{i,j}$, where $\pi_0(a_{i,j}) = a'_{i,j}$, and $\pi_0(b_{i,j}) = b'_{i,j}$, which is a star-free OPE. By definition of ∇, $a_{i,j} \nabla b_{i,j} \cap B^+AB^+ = \neg(a_{i,j}[B^+_p]b_{i,j}) \cap a_{i,j}B^+_p b_{i,j} \cap B^+AB^+ = \neg(a_{i,j}[B^+_p]b_{i,j}) \cap a_{i,j}B^*AB^*b_{i,j}$. Hence, its renaming is $\neg(\pi_0(a_{i,j})[\pi_0(B^*_p)\pi_1(B_p)\pi_0(B^*_p)]\ \pi_0(b_{i,j})) \cap \pi_0(a_{i,j}B^*)\pi_1(A)\pi_0(B^*\ \ b_{i,j}) = \neg(a'_{i,j}[B^+_{p-1}]b'_{i,j}) \cap a'_{i,j}B^+_{p-1}b'_{i,j}$, a star-free OPE. □

Theorem 24. *For every star-free OPE E on an OP alphabet (Σ, M), there is a FO formula φ on (Σ, M) such that $L_M(E) = L(\varphi)$.*

Proof. The proof is by induction on E's structure. Of course, singletons are easily first-order definable; for negation and union we use $\neg$ and $\vee$ as natural. Like in the case of star-free regular languages, concatenation is less immediate, and it is based on formula *relativization*. Consider two FO formulas φ and ψ, and assume w.l.o.g. that their variables are disjunct, and let $\boldsymbol{x}$ be a variable not used in either of them. To construct a relativized variant of φ, called $\varphi_{<\boldsymbol{x}}$, proceed from the outermost quantifier, going inward, and replace every subformula $\exists \boldsymbol{z} \lambda$ with $\exists \boldsymbol{z}((\boldsymbol{z} < \boldsymbol{x}) \wedge \lambda)$. Variants $\varphi_{\geq \boldsymbol{x}}$ and $\varphi_{>\boldsymbol{x}}$ are analogous. We also call $\varphi_{\boldsymbol{x},\boldsymbol{y}}$ the relativization where quantifications $\exists \boldsymbol{z} \lambda$ are replaced by $\exists \boldsymbol{z}((\boldsymbol{x} < \boldsymbol{z} < \boldsymbol{y}) \wedge \lambda)$. The language $L(\varphi) \cdot L(\psi)$ is defined by the following formulas: $\exists \boldsymbol{x}(\varphi_{<\boldsymbol{x}} \wedge \psi_{\geq \boldsymbol{x}})$ if $\varepsilon \notin L(\psi)$; otherwise $\exists \boldsymbol{x}(\varphi_{<\boldsymbol{x}} \wedge \psi_{\geq \boldsymbol{x}}) \vee \varphi$.

The last part we need to consider is the fence operation, i.e. $a[E]b$. Let φ be a FO formula such that $L(\varphi) = L_M(E)$, for a star-free OPE E. Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be two variables unused in φ. Then the language $L(a[E]b)$ is the one defined by $\exists \boldsymbol{x} \exists \boldsymbol{y}(a(\boldsymbol{x}) \wedge b(\boldsymbol{y}) \wedge \boldsymbol{x} \curvearrowright \boldsymbol{y} \wedge \varphi_{\boldsymbol{x},\boldsymbol{y}})$. □

5.1 Linear Noncounting OPLs Are First-Order Definable

Definition 25. *Let $G = (\Sigma, V_N, P, S)$ be a linear grammar, i.e., a grammar where all rule rhs $\in \Sigma^+ V_N \Sigma^* \cup \Sigma^* V_N \Sigma^+ \cup \Sigma^+$. The finite state* stencil automaton *associated to G is $\mathcal{A}^{\Xi}_G = (Q, \Xi, \delta, S, \{q_F\})$, where $Q = V_N \cup \{q_F\}$ are its states, $\Xi \subseteq (\Sigma^* \times \Sigma^*) \cup \Sigma^+$, its input alphabet, is the set of stencils of G,[4] and the transition relation $\delta \subseteq Q \times \Xi \times Q$ is defined as follows:*

$$\delta = \{A \xrightarrow{(x,y)} B \mid A \to xBy \in P\} \cup \{A \xrightarrow{z} q_F \mid A \to z \in P\}.$$

Lemma 26. *Let G be a linear OPG, with OPM M, and Ξ the set of stencils of G. $L(G)$ is NC iff $L(\mathcal{A}^{\Xi}_G)$ is a NC regular language.*

Proof. By construction, every derivation $A_0 \overset{*}{\Longrightarrow} x_1 A_1 y_1 \overset{*}{\Longrightarrow} x_1 x_2 A_2 y_2 y_1 \overset{*}{\Longrightarrow} x_1 x_2 \dots x_k A_k\ y_k y_{k-1} \dots y_1 \overset{*}{\Longrightarrow} x_1 x_2 \dots x_k\ z y_k y_{k-1} \dots y_1$ of G corresponds to a run $A_0 \vdash A_1 \vdash \dots \vdash A_k \vdash q_F$ of $\mathcal{A}^{\Xi}_G$, reading the word $(x_1, y_1)(x_2, y_2) \dots (x_k, y_k)z$, and vice versa.

[4] $\Xi := \{(x, y) \mid A \to xBy \in P\} \cup \{z \mid A \to z \in P\}$.

According to Definitions 10 and 12, if it is $x_1x_2\dots\ x_kzy_ky_{k-1}\dots\ y_1 = ou^nwv^np$, for some strings o,u,w,v,p, then $ou^{n+1}wv^{n+1}\ p \in L(G)$, i.e., assuming that $x_i\dots x_j = u$ and $y_j\dots y_i = v$, $ou^{n+1}wv^{n+1}p = x_1x_2\dots(x_i\dots x_j)^2x_{j+1}\dots x_kz\ y_ky_{k-1}\dots\ (y_j\ \dots y_i)^2y_{i-1}\dots\ y_1$.
Hence, $(x_1,y_1)(x_2,y_2)\dots(x_k,y_k)z = o'u'^nw'$, where $u' = (x_i,y_i)\dots(x_j,y_j)$, and $(x_1,y_1)\dots\ ((x_i,y_i)\dots(x_j,y_j))^2\ (x_{j+1},\ y_{j+1})\dots(x_k,y_k)z = o'u'^{n+1}w' \in L(\mathcal{A}_G^{\Xi})$. The other direction is analogous. □

Theorem 27. *Let G be a linear NC OPG with OPM M, $L(G)$ is expressible in $FO_{(\Sigma,M)}$.*

Proof. From Lemma 26, we can build $\mathcal{A}_G^{\Xi}$, which defines a NC regular language. It is known from [28] that there exists an equivalent first-order formula $\varphi(\mathcal{A}_G^{\Xi})$ on the alphabet Ξ.

We can build a $\mathrm{FO}_{(\Sigma,M)}$ formula defining $L(G)$ in this way. First, for each $p \in \Xi$, where $p = (u,v)$, $u = u_1u_2\dots u_m$, $v = v_1v_2\dots v_n$, $m,n \geq 0$, $m+n > 0$, we introduce the following formula σ_p:

$$\sigma_p(\boldsymbol{x}) := \begin{pmatrix} u_1(\boldsymbol{x}) \wedge u_2(\boldsymbol{x}+1) \wedge \\ \dots \wedge u_n(\boldsymbol{x}+m-1) \end{pmatrix} \wedge \exists \boldsymbol{y} \begin{pmatrix} \boldsymbol{x}+m-1 \curvearrowright \boldsymbol{y} \wedge v_1(\boldsymbol{y}) \wedge v_2(\boldsymbol{y}+1) \wedge \\ \dots \wedge v_n(\boldsymbol{y}+n-1) \end{pmatrix}$$

(the case $p \in \Sigma^+$ is trivial.) E.g. for $p = (\varepsilon, ab)$, $\sigma_p(\boldsymbol{x}) = \exists \boldsymbol{y}(\boldsymbol{x} \curvearrowright \boldsymbol{y} \wedge a(\boldsymbol{y}) \wedge b(\boldsymbol{y}+1))$.

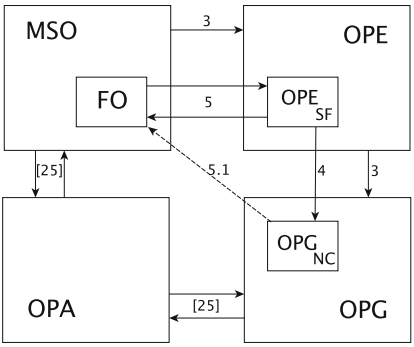
It is easy to see that σ_p is a straightforward way to encode a stencil p into a $\mathrm{FO}_{(\Sigma,M)}$ formula defining its structure.

Let us consider $\varphi(\mathcal{A}_G^{\Xi})$, and obtain a trivially equivalent formula $\varphi'(\mathcal{A}_G^{\Xi})$ by substituting each quantified subformula $\exists \boldsymbol{x}(\rho)$ in it with $\exists \boldsymbol{x}\left(\left(\bigvee_{p\in\Xi} p(\boldsymbol{x})\right) \wedge \rho\right)$. Hence, we are stating that each quantified variable in $\varphi(\mathcal{A}_G^{\Xi})$ correspond to a position in which there is a stencil.

To obtain a $\mathrm{FO}_{(\Sigma,M)}$ formula ξ, such that $L(\xi) = L(G)$, we take $\varphi'(\mathcal{A}_G^{\Xi})$, and substitute in it every subformula $p(\boldsymbol{x})$ with $\sigma_p(\boldsymbol{x})$, for each $p \in \Xi$, and variable $\boldsymbol{x}$. □

6 Conclusion

To the best of our knowledge OPLs are the largest family among the many families of structured CFLs to enjoy closure w.r.t. most fundamental language operations and to be characterized in terms of a MSO logic that naturally extends the classic one for regular languages. In this paper we have introduced OPEs as an extension of Kleene's REs. We have shown that languages defined by OPEs coincide with OPLs and that FO-definable OPLs are those defined by SF OPEs and are NC, in sharp contrast with comparable results framed in the context of tree-languages [21,35]. Together with previous partial results [15,24] and the fact that linear NC OPLs are first-order definable (Sect. 5.1), they support our conjecture that OPLs jointly with our MSO and FO logics, perfectly extend the classic results of regular languages. Figure 4 summarizes past, present and "future" results on OPLs and their logics.



Legend
All boxes denote classes of OPLs with a common OPM:

- MSO denotes languages defined through MSO formulas
- FO denotes languages defined through FO formulas
- OPA denotes languages defined through OP automata [25]
- OPE_{SF} denotes languages defined through star-free OPEs
- OPG_{NC} denotes aperiodic OPLs, i.e., languages defined through NC OPGs

Arrows between boxes denote language family inclusion; they are labeled by the reference pointing to where the property has been proved, either to previous literature or to a section of this paper. The dashed arrow stands for the inclusion of linear NC OPLs, and for our conjecture.

Fig. 4. The relations among the various characterizations of OPLs and their aperiodic subclass.

The figure immediately suggests a first further research step, i.e., making the internal triangle a square, as well as the external one: we conjecture that once the concept of NC OPLs has been put in the appropriate framework, a further characterization thereof in terms of a suitable subclass of OPAs should be possible but so far we did not pursue such an option.

The further goal that we wish to pursue is the complete reproduction of the historical path that, for regular languages, lead from the first characterization in terms of MSO logic to the restricted case of FO characterization of NC regular languages, to the temporal logic case which in turn is first-order complete, and, ultimately, to the success of model checking techniques.

Some proposals of temporal logic extension of the classical linear or branching time ones to cope with the typical nesting structure of CF languages have been already offered in the literature. E.g., [1,4,6] present different cases of temporal logics extended to deal with VPLs; they also prove FO-completeness of such logics but do not afford the relation between FO and MSO versions of their logics; see also [3]. A first example of temporal logic for OPLs and a related model checking algorithm have also been provided in [9].

Given that most, if not all, of the CF languages for practical applications are aperiodic, the final goal of building verification tools that cover a much wider application field than that of regular languages –and of VPLs too– does not seem unreachable.

Acknowledgments. We are grateful to the reviewers for their careful reading and suggestions.

References

1. Alur, R., Arenas, M., Barceló, P., Etessami, K., Immerman, N., Libkin, L.: First-order and temporal logics for nested words. In: Logical Methods in Computer Science, vol. 4, no. 4 (2008)
2. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM **56**(3), 1–43 (2009)
3. Alur, R., Bouajjani, A., Esparza, J.: Model checking procedural programs. Handbook of Model Checking, pp. 541–572. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_17
4. Alur, R., Chaudhuri, S., Madhusudan, P.: Software model checking using languages of nested trees. ACM Trans. Program. Lang. Syst. **33**(5), 15:1–15:45 (2011). https://doi.org/10.1145/2039346.2039347
5. Autebert, J.-M., Berstel, J., Boasson, L.: Context-free languages and pushdown automata. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, pp. 111–174. Springer, Heidelberg (1997). https://doi.org/10.1007/978-3-642-59136-5_3
6. Bozzelli, L., Sánchez, C.: Visibly linear temporal logic. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS (LNAI), vol. 8562, pp. 418–433. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08587-6_33
7. von Braunmühl, B., Verbeek, R.: Input-driven languages are recognized in log n space. In: Karpinski, M. (ed.) FCT 1983. LNCS, vol. 158, pp. 40–51. Springer, Heidelberg (1983). https://doi.org/10.1007/3-540-12689-9_92
8. Büchi, J.R.: Weak second-order arithmetic and finite automata. Math. Log. Q. **6**(1–6), 66–92 (1960)
9. Chiari, M., Mandrioli, D., Pradella, M.: Temporal logic and model checking for operator precedence languages. In: Orlandini, A., Zimmermann, M. (eds.) Proceedings Ninth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2018, Saarbrücken, Germany, 26–28th September 2018. EPTCS, vol. 277, pp. 161–175 (2018). https://doi.org/10.4204/EPTCS.277.12
10. Crespi Reghizzi, S., Guida, G., Mandrioli, D.: Noncounting context-free languages. J. ACM **25**, 571–580 (1978)
11. Crespi Reghizzi, S., Guida, G., Mandrioli, D.: Operator precedence grammars and the noncounting property. SICOMP: SIAM J. Comput. **10**, 174–191 (1981)
12. Crespi Reghizzi, S., Mandrioli, D.: Operator precedence and the visibly pushdown property. J. Comput. Syst. Sci. **78**(6), 1837–1867 (2012)
13. Crespi Reghizzi, S., Mandrioli, D., Martin, D.F.: Algebraic properties of operator precedence languages. Inf. Control **37**(2), 115–133 (1978)
14. Crespi Reghizzi, S., Pradella, M.: Beyond operator-precedence grammars and languages. J. Comput. Syst. Sci. (2020). to appear
15. Crespi Reghizzi, S., Mandrioli, D.: A class of grammar generating non-counting languages. Inf. Process. Lett. **7**(1), 24–26 (1978). https://doi.org/10.1016/0020-0190(78)90033-9
16. Diekert, V., Gastin, P.: First-order definable languages. In: Logic and Automata: History and Perspectives, Texts in Logic and Games, pp. 261–306. Amsterdam University Press (2008)
17. Elgot, C.C.: Decision problems of finite automata design and related arithmetics. Trans. Am. Math. Soc. **98**(1), 21–52 (1961)
18. Ésik, Z., Iván, S.: Aperiodicity in tree automata. In: Bozapalidis, S., Rahonis, G. (eds.) CAI 2007. LNCS, vol. 4728, pp. 189–207. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75414-5_12

19. Floyd, R.W.: Syntactic analysis and operator precedence. J. ACM **10**(3), 316–333 (1963)
20. Harrison, M.A.: Introduction to Formal Language Theory. Addison Wesley, Boston (1978)
21. Heuter, U.: First-order properties of trees, star-free expressions, and aperiodicity. ITA **25**, 125–145 (1991). https://doi.org/10.1051/ita/1991250201251
22. Langholm, T.: A descriptive characterisation of linear languages. J. Log. Lang. Inf. **15**(3), 233–250 (2006). https://doi.org/10.1007/s10849-006-9016-z
23. Lautemann, C., Schwentick, T., Thérien, D.: Logics for context-free languages. In: Pacholski, L., Tiuryn, J. (eds.) CSL 1994. LNCS, vol. 933, pp. 205–216. Springer, Heidelberg (1995). https://doi.org/10.1007/BFb0022257
24. Lonati, V., Mandrioli, D., Panella, F., Pradella, M.: First-order logic definability of free languages. In: Beklemishev, L.D., Musatov, D.V. (eds.) CSR 2015. LNCS, vol. 9139, pp. 310–324. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20297-6_20
25. Lonati, V., Mandrioli, D., Panella, F., Pradella, M.: Operator precedence languages: their automata-theoretic and logic characterization. SIAM J. Comput. **44**(4), 1026–1088 (2015)
26. Mandrioli, D., Pradella, M.: Generalizing input-driven languages: theoretical and practical benefits. Comput. Sci. Rev. **27**, 61–87 (2018). https://doi.org/10.1016/j.cosrev.2017.12.001
27. McNaughton, R.: Parenthesis grammars. J. ACM **14**(3), 490–500 (1967)
28. McNaughton, R., Papert, S.: Counter-Free Automata. MIT Press, Cambridge (1971)
29. Nowotka, D., Srba, J.: Height-deterministic pushdown automata. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 125–134. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74456-6_13
30. Pin, J.: Logic on words. In: Current Trends in Theoretical Computer Science, pp. 254–273 (2001)
31. Floyd, C.: Theory and practice of software development. In: Mosses, P.D., Nielsen, M., Schwartzbach, M.I. (eds.) CAAP 1995. LNCS, vol. 915, pp. 25–41. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-59293-8_185
32. Rabinovich, A.: A proof of Kamp's theorem. Log. Methods Comput. Sci. **10**(1), 1–16 (2014). https://doi.org/10.2168/LMCS-10(1:14)2014
33. Salomaa, A.K.: Formal Languages. Academic Press, New York (1973)
34. Thatcher, J.: Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. J. Comput. Syst. Sci. **1**, 317–322 (1967)
35. Thomas, W.: Logical aspects in the study of tree languages. In: Courcelle, B. (ed.) 9th Colloquium on Trees in Algebra and Programming, CAAP 1984, Bordeaux, France, March 5–7, 1984, Proceedings, pp. 31–50. Cambridge University Press (1984)
36. Trakhtenbrot, B.A.: Finite automata and logic of monadic predicates. Doklady Akademii Nauk SSR **140**, 326–329 (1961). (in Russian)