Beyond operator-precedence grammars and languages[☆]

Stefano Crespi Reghizzi[a,b], Matteo Pradella[a,b]

[a]*Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo da Vinci 32, Milano, Italy*
[b]*CNR IEIIT-MI, Milano, Italy*

Abstract

Operator Precedence Languages (OPL) are deterministic context-free and have desirable properties. OPL are parallely parsable, and, when structurally compatible, are closed under Boolean operations, concatenation and star; they include the Input Driven languages. OPL use three relations between two terminal symbols, to assign syntax structure to words. We extend such relations to k-tuples of consecutive symbols, in agreement with strictly locally testable regular languages. For each k, the new corresponding class of Higher-order Operator Precedence languages properly includes the OPL and enjoy many of their properties. OPL are a strict hierarchy based on k, which contains maximal languages.

Keywords Operator Precedence Languages, Input-Driven Languages, Visibly Pushdown languages, Deterministic Context-Free Languages, Syntactic Tags, Boolean Closure, Locally Testable Languages, Local Parsability, Grammar Inference.

1. Introduction

We improve on the classic language family of *operator-precedence* (OP) languages (OPL), originally invented by R. Floyd [2] for efficient parsing and still used for fast compilation (e.g., [3]). Shortly after invention and more so in recent years, their mathematical and algorithmic properties have been investigated with quite a variety of motivations and results that are worth mentioning (chronologically), before we introduce our current research.

OP languages have been found suitable for grammar inference [4, 5] thanks to their lattice-theoretical properties [6]. Years later, the focus of formal language research on *Input-Driven* (ID) [7, 8] (or "Visibly Push-down" [9]) languages has prompted reexamination of OP languages, resulting [10] in the awareness that OP closure properties imply ID closure properties and that the ID languages are a class of OP languages characterized by restricted OP relations. Motivated by the progress of parallel computers, the local parsability property of OP languages has been exploited to generate fast

[☆]Part of this work was published preliminarily in [1].

Email addresses: stefano.crespireghizzi@polimi.it (Stefano Crespi Reghizzi), matteo.pradella@polimi.it (Matteo Pradella)

parallel parsers [11]. OP languages offer promise for model-checking of infinite-state systems due to the Boolean closure, ω-languages, logic and automata characterizations, and the ensuing decidability of relevant problems [12].

But, in spite of such remarkable properties, OP languages are less widely referenced than other deterministic families, which is perhaps due to some of their expressive limitations, in particular the operator form required for productions. A recent survey of the theory of OP languages is in [13].

To introduce the present development, we need to recall the essential idea of OPL, by examining how its bottom-up deterministic parser operates. Such parser localizes the left/right edge of the handle (i.e., a factor to be reduced by a grammar rule) by means of three precedence relations, represented by the *tags* $\lessdot, \gtrdot, \doteq$. Such OP relations are defined between two consecutive terminals, possibly separated by a nonterminal. E.g., the *yield precedence* relation $a \lessdot b$ says that b is the leftmost terminal of the handle and a is the last terminal of the left context. The no-conflict condition of OP grammars permits to find the handle positions by means of a local inspection of terminals, and ensures unambiguity. An OP parser configuration is essentially a word consisting of alternated terminals and tags, i.e., a *tagged word*; notice that nonterminal symbols, although available in the configuration, play no role in determining the handle positions, but are of course necessary for checking syntactic correctness. In general, any language having the property that handles can be localized by a local test is called *locally parsable* and its parser is amenable to parallelization [14].

If the parser is modeled as a push-down automaton, each pair of terminals associated to a left or to a right edge of a handle, respectively triggers a push or a pop move; i.e., the move choice is driven by two consecutive input symbols. From this viewpoint, the well-known model of input-driven languages is a special case of the OP model, since just one terminal suffices to choose the move. This is shown in [10], where the OP relations characterizing the input-driven languages are studied and called *partitioned* since their structure mirrors the alphabet threefold partition. The syntax structures compatible with such partitioned relations are often too restrictive for the constructs of computer languages (besides LISP), making the ID model unsuitable for defining technical languages, e.g., a markup language like HTML5 has special rules that allow dropping some closing tags.

On the other hand, OP grammars have been used in many compilers though they are less expressive than LR(1) grammars; in practice (e.g., in [11]) grammar adjustment and lexical transformations are sometimes needed in order to obtain an OP grammar for languages having complex syntax. From this a natural question: can we improve the generative capacity and structural adequacy of OP grammars, without jeopardizing their nice properties, by letting the parser examine more than two consecutive terminals to determine the handle position? Quite surprisingly, to our knowledge the question remained unanswered until now, but in the Section 5 we mention some related research.

Next, we intuitively present the new family of languages and grammars called *Higher-order Operator Precedence* (HOP), which is the union of subfamilies identified by an integer parameter $k = 3, 5, \ldots$, which is an odd integer specifying the number of consecutive terminals and intervening tags, to be examined for localizing handles of reductions. As said, the value $k = 3$ is for OP languages, which therefore are almost the same as the HOP(3) subfamily, apart the permission to use regular expressions in the

HOP grammar rules. Now, the set of OP relations traditionally visualized in a $|\Sigma| \times |\Sigma|$ matrix, is represented by a set of tagged words of length k.

This article develops the theory of HOP grammars and languages to a degree comparable with the theory of OP grammars. Our main contributions are: a definition of HOP(k) grammars and their parsing algorithm, a decidable condition for testing whether a grammar has the HOP(k) property, the proof that the OP family is properly included into the HOP(3) one, and that parameter k induces a strict hierarchy of language families. We show that most formal properties of OPL carry over to HOP languages: closure under intersection with regular languages, under reversal and, for structurally compatible grammars, also under Boolean operations. Another preserved OPL property is that, within each HOP(k) subfamily, the class of languages having the same set of tagged words of length k has a unique maximal element, called *max-language*. Interestingly, max-languages can be defined by a simple cancellation rule that applies to tagged words, and iteratively cancels innermost handles by a process called a *reduction*. Before each cancellation, the word, completed with tags, has to pass local tests, defined by means of a *strictly locally testable* [15] regular language of order k. We mention several properties of the max-language family, in particular that they form a strict infinite hierarchy ordered by parameter k. Open questions concerning HOP languages are discussed in the conclusion.

For applications, we view the HOP model as a possible upgrade of the OP model, whenever the latter lacks the expressiveness needed for defining a technical language; we show a realistic toy language that has a straightforward HOP grammar whereas an equivalent OP grammar is less convenient. In particular, HOP grammars have the advantage of permitting regular expressions in the right parts of rules. Moreover, it would be straightforward to adapt to HOP the parallel OP parsing algorithm [11]. But of course ours is a theoretical contribution, and further engineering and experimental work is needed to assess practicality of the HOP model.

Paper organization. Section 2, after the basic notation and definitions, defines the OP grammars, the strictly locally testable languages, the tagged words and their conflicts. Section 3 introduces the HOP(k) languages, compares that OP and HOP(3) languages, and presents a deterministic PDA for such languages. Section 4 proves some closure properties of the families of conflict-free HOP(k) languages, defines the max-language for each family, and proves two hierarchy results. Section 5 compares HOP with some related existing models, and mentions open problems and future research directions. Appendix 1 lists a grammar for a Pascal-like language having OP conflicts resolvable in HOP. Appendix 2 proves Boolean closure for extended context-free parenthesis languages.

2. Basic Definitions

For terms not defined here, we refer to any textbook on formal languages, e.g. [16]. The alphabets that we consider include the terminal symbols, denoted by Σ, and some special symbols: the symbols "[", "]", and $\odot$ called *tags* and denoted by Δ. We also include into Σ the special symbol # to mark the start and end of a word. Since we will need to refer to different terminal alphabets in automata and grammars depending on their usage, e.g. Σ or $\Sigma \cup \Delta$, we use in the following definitions the neutral symbol Υ to denote a generic alphabet. The empty word is ε; unless stated otherwise, all languages considered are free from the empty word.

For any $k \geq 1$, for a word w, $|w| \geq k$, let $i_k(w)$ and $t_k(w)$ be the prefix and, respectively, the suffix of w of length k. If a word w has length at least k, $f_k(w)$ denotes the set of factors of w of length k, otherwise the empty set. The notations i_k, t_k and f_k can be applied in the obvious way to languages. The i-th character of w is denoted by $w(i), 1 \leq i \leq |w|$.

A (nondeterministic) *finite automaton* (FA) is a 5-tuple $M = (\Upsilon, Q, \delta, I, T)$, where $I, T \subseteq Q$ are respectively the initial and final states and δ is a relation (or its graph) over $Q \times \Upsilon \times Q$. A (labeled) *path* is a sequence $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} q_n$, such that, for each $1 \leq i < n$, $(q_i, a, q_{i+1}) \in \delta$. The *path states* are the sequence $q_1 q_2 \dots q_n$, the *path label* is the word $w = a_1 a_2 \dots a_{n-1}$. Word w is recognized by M, i.e., $w \in L(M)$ if $q_1 \in I$ and $q_n \in T$ for some path. An FA is *unambiguous* if each word in language $L(M)$ is the label of exactly one path, i.e., it is recognized by just one computation.

Now we move to context-free grammars. Since the new models to be introduced are directly related to grammar rules having regular languages in their right hand sides, we start from the definition of such grammars, recalling that they are diffusely applied in language reference manuals and in compilers (e.g., see [17]).

Definition 2.1 (Extended and non-extended CF grammars)**.** An *extended context-free* (ECF) grammar is a 4-tuple $G = (V_N, \Upsilon, P, S)$, where Υ is the terminal alphabet, V_N is the nonterminal alphabet, P is the set of rules, and $S \subseteq V_N$ is the set of axioms. Each rule has the form $X \to R_X$, where $X \in V_N$ and R_X is a *regular language* over the alphabet $V = \Upsilon \cup V_N$. Language R_X will be defined by means of an FA, $M_X = (V, Q_X, \delta_X, I_X, T_X)$, that we safely assume to be unambiguous. Without loss of generality, each nonterminal X has exactly one rule, to be written as $X \to M_X$ or $X \to R_X$, understanding that $R_X = L(M_X)$.

We say that a rule $X \to R_X$ *includes a copy rule* if language R_X includes a word made by just one nonterminal symbol, i.e., there exists $Y \in V_N$ such that $Y \in R_X$; we can w.l.o.g. assume that the grammar does not include any such copy rules.

Without loss of generality, we can also assume that for any two rules $X \to R_X$, $Y \to R_Y$ the regular languages are disjoint, i.e., $R_X \cap R_Y = \emptyset$.

A (non-extended) *context-free* (CF) grammar is an ECF grammar such that for each rule $X \to R_X$, R_X is a finite language over V. For a CF grammar, with an abuse of notation we also say that $X \to x$ is in P if $x \in R_X$.

The *derivation relation* $\Rightarrow \subseteq (V^+ \times V^*)$ is defined as follows $u \Rightarrow v$ if $u = u'Xu''$, $v = u'wu''$, $X \to R_X \in P$, and $w \in R_X$. The reflexive and transitive closure of $\Rightarrow$ is denoted by $\overset{*}{\Rightarrow}$.
A word is X-*grammatical* if it derives from a nonterminal symbol X. If X is an axiom, the word is *sentential*. The language generated by G starting from a nonterminal X is denoted by $L(G, X) \subseteq \Upsilon^+$ and $L(G) = \bigcup_{X \in S} L(G, X)$.

The usual assumption that all parts of a CF grammar are productive can be reformulated for ECF grammars by combining the following two assumptions: (i) for each rule $X \to M_X$ the FA M_X is trim (i.e. without unreachable states and unusable transitions), and (ii) for each nonterminal X there exists a sentential word containing X and there exists an X-grammatical word $w \in \Upsilon^*$.

To linearly represent syntactic structures and to define ambiguity and structural equivalence, it is convenient to use parenthesis grammars [18]. We need the following (alphabetic) projections: $\nu : \Upsilon \cup V_N \to \Upsilon$, and $\pi : \Upsilon \cup \{(,)\} \to \Upsilon$, where we assume that Υ does not contain the round parentheses "(" and ")".

Let $G = (V_N, \Upsilon, P, S)$ be a grammar. The corresponding *parenthesis grammar*, denoted by $G_{()}$, is defined by the 4-tuple $(V_N, \Upsilon \cup \{(,)\}, P', S)$ where $P' = \{X \to (R_X) \mid X \to R_X \in P\}$. A grammar G is *structurally ambiguous* if there exist $w, z \in L(G_{()}), w \neq z$, such that $\pi(w) = \pi(z)$. Two grammars G' and G'' are *structurally equivalent* if $L(G'_{()}) = L(G''_{()})$.

Operator grammars and precedences. A well-known normal form for ECF and CF grammars is the operator normal form, which we use throughout the paper. A word w over $V_N \cup \Upsilon$ is in *operator form* if it contains at least one element of Υ and it does not contain two adjacent nonterminals, i.e., if $f_2(w) \cap V_N V_N = \emptyset$. An ECF grammar is in *operator form* if, for all rules $X \to R_X$ and for each word $x \in R_X$, x is in operator form.

The operator precedence (OP) grammars, introduced by Floyd [2], define a family of deterministic languages that have a very efficient and widely used parsing algorithm, see [3] for a practical presentation. Later studies [19, 6, 10, 20] have focused in particular on structural and closure properties, on partial ordering and lattices of OP languages, on decidability and complexity of classical problems, and on logic and automata characterizations.

Definition 2.2 (Operator precedence grammars)**.** Let G be a (non-extended) CF grammar. The characters $\{\lessdot, \gtrdot, \doteq\}$, called *precedence tags*, are assumed to be distinct from all others. They are used to represent three *precedence relations* over the terminal alphabet $\Sigma \times \Sigma$, respectively called: *yield* precedence, *take* precedence, and *equal* in precedence. Let u, v, x be, possibly empty, words over $\Sigma \cup V_N$, $A, B, C \in V_N$ and $a, b \in \Sigma$.

$$\begin{array}{rcc}
\text{yields precedence: } a \lessdot b & \iff & A \to uaBv \in P \text{ and } \left(B \overset{*}{\Rightarrow} bx \text{ or } B \overset{*}{\Rightarrow} Cbx\right) \\
\text{takes precedence: } a \gtrdot b & \iff & A \to uBbv \in P \text{ and } \left(B \overset{*}{\Rightarrow} xa \text{ or } B \overset{*}{\Rightarrow} xaC\right) \\
\text{equal in precedence: } a \doteq b & \iff & A \to uaBbv \in P \text{ or } A \to uabv \in P
\end{array}$$

The *operator precedence matrix* M is a $|\Sigma| \times |\Sigma|$ array that to each ordered pair (a, b) associates the (possibly empty) set, denoted by M_{ab}, of the precedence relations holding between a and b.

Grammar G has the *operator precedence property* (OP) if, for each pair of terminal characters a, b, at most one precedence relation holds, i.e., $|M_{ab}| \leq 1$. In that case we say that matrix M is *conflict-free*. Two OP grammars G' and G'' are *compatible* if the union, case by case, of their matrices is conflict-free.

Example 2.3. The following grammar G_1 generates $L(G_1) = \{a^n \left(cb^+\right)^n \mid n \geq 1\}$

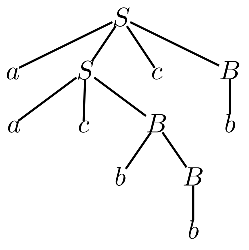
and has the precedence relations shown in the conflict-free OP matrix M:

$$G_1 = \{S \to aScB \mid acB\,,\; B \to bB \mid b\} \qquad M = \begin{array}{c|c|c|c|c} & a & b & c & \# \\ \hline a & \lessdot & & \doteq & \\ \hline b & & \lessdot & \gtrdot & \gtrdot \\ \hline c & & \lessdot & \gtrdot & \gtrdot \\ \hline \# & \lessdot & & & \doteq \end{array}$$

Notice that conventionally character # is used to delimit any input word. The precedence relations involving # can be defined as those arising from the special grammar rules $S_0 \to \#X\#$ for each $X \in S$, where S_0 is a new nonterminal.

Driven by the precedence relations, the classical bottom-up parsing algorithm [3] executes a series of reductions (i.e. of derivations in reverse), shown in Figure 1 (i) for the input $aacbbcb$ together with the resulting tree. Sentential forms are interspersed with the appropriate precedence relation between the corresponding terminal symbols, to show how precedences in a sense "represent" the syntactic structure of a sentence. At each step, an innermost substring $\lessdot \ldots \gtrdot$ which may only contain $\doteq$ as tag, reduces

$$\begin{array}{l}
\# \lessdot a \lessdot a \doteq c \lessdot b \lessdot b \gtrdot c \lessdot b \gtrdot \# \Longleftarrow_{G_1} \\
\# \lessdot a \lessdot a \doteq c \lessdot bB \gtrdot c \lessdot b \gtrdot \# \Longleftarrow_{G_1} \\
\# \lessdot a \lessdot a \doteq cB \gtrdot c \lessdot b \gtrdot \# \Longleftarrow_{G_1} \\
\# \lessdot aS \doteq c \lessdot b \gtrdot \# \Longleftarrow_{G_1} \\
\# \lessdot aS \doteq cB \gtrdot \# \Longleftarrow_{G_1} \\
\#S \doteq \#
\end{array}$$



(i)

$$\begin{array}{l}
\#[a[a \odot c[b[b]c[b]\#, \\
\#[a[a \odot c[b]c[b]\#, \\
\#[a[a \odot c]c[b]\#, \\
\#[a \odot c[b]\#, \\
\#[a \odot c]\#, \\
\# \odot \#
\end{array}$$

(ii)

Figure 1: OP parsing: (i) Reductions and syntax tree for grammar G_1 of Example 2.3; (ii) Tagged forms corresponding to the reductions (discussed later).

to a nonterminal symbol, and a precedence relation between terminal characters is inserted into the sentential form. Notice that such algorithm not necessarily operates from left to right, but it may perform reductions in any order and also in parallel.

OP languages having compatible matrices are closed with respect to Boolean operations, concatenation, Kleene star, reversal, prefix, suffix, homomorphisms preserving the OP property, intersection with regular sets [6, 10]. The partial orders and lattices

[6] of certain subfamilies of OP languages having the *non-counting property* [21, 22] have been exploited for grammatical inference algorithms.

The classical definition of OP does not deal with ECF grammars, but it will be reformulated for that case. Such extension and especially a more substantial one that uses precedence relations between more than two characters, are presented in Sect. 3 as the main contribution of this paper.

Strict local testability and tagged languages. To extend the OP property, we will move from a binary relation such as $c \lessdot b$ encoded by a word of length 3, to higher order relations encoded by longer words such as $a \lessdot a \doteq c \lessdot b$, which has length 7, for short called a 7-word. We have chosen to encode the latter relation with different tags, as $a[a \odot c[b$, for no other reason but to avoid confusion between the existing OP theory and the new more general higher order theory.

Observing in Figure 1 how a given input text is parsed, we see that the 3-words encoding precedence relations occur as factors within a text obtained by applying reductions, i.e., they are the content of a window of width 3 which slides over the text. Such texts alternate terminals and tags, and are called *tagged words* or, more specifically, *tagged forms* of a given grammar. An example of tagged forms is in Figure 1 (ii). The idea of defining a language by listing the permitted contents of a sliding window of fixed width is the foundation of the classical family of strictly locally testable languages [15]. Next, we list their essential definitions as in [23], with minor differences. We assume that any input word $x \in \Upsilon^+$ is enclosed between two special words of sufficient length, called *end-words* and both denoted by the same symbol $\textcircled{\#}$, as their difference is always clear from the context.

Let $\#$ be a character, tacitly assumed to be in Υ and used only in the end-words. We actually use two different end-words, without or with tags, depending on circumstances: $\textcircled{\#} \in \#^+$ (e.g. in Definition 2.4) or $\textcircled{\#} \in (\#\odot)^*\#$, (e.g. in Definition 2.5).

Definition 2.4. Let $k \geq 2$ be an integer, called *width*. A language L is *k-strictly locally testable*, if there exists a k-word set $F \subseteq \Upsilon^k$ such that $L = \{x \in \Upsilon^* \mid f_k(\textcircled{\#}\, x\, \textcircled{\#}) \subseteq F\}$; then we write $L = \mathrm{SLT}(F)$. A language is *strictly locally testable* (*SLT*) if it is k-strictly locally testable for some k.

We need to define the words that contain tags and alternate tags and terminals, in order to apply the SLT definition to such words.

Definition 2.5 (tagged word and tagged language)**.** Let here and throughout the paper $k \geq 3$ be an odd integer. Let Σ be the terminal alphabet. A *tagged word* is a word w in the set $\Sigma(\Delta\Sigma)^*$, denoted by $\Sigma^{\square}$. A *tagged sub-word* of w is a factor of w that is a tagged word. A *tagged language* is a set of tagged words. Let $\Sigma^{\square k} = \{w \in \Sigma^{\square} \mid |w| = k\}$. We call *tagged k-word* any word in $\Sigma^{\square k}$. The set of all tagged k-words that occur as sub-word in w is denoted by $\varphi_k(w)$.

A language $L \subseteq \Sigma^{\square}$ is a *k-strictly locally testable tagged language* if there exists a set of tagged k-words $\Phi \subseteq \Sigma^{\square k}$ such that $L = \left\{w \in \Sigma^{\square} \mid \varphi_k(\textcircled{\#}\, [\ w\ ]\, \textcircled{\#}) \subseteq \Phi\right\}$. In that case we write $L = \mathrm{SLT}(\Phi)$. The k-word set $F \subseteq (\Sigma \cup \Delta)^k$ *derived from* Φ is $F = \bigcup_{x \in \mathrm{SLT}(\Phi)} f_k(x)$.

The projection $\sigma : \Sigma \cup \Delta \to \Sigma$ erases all the tags and preserves any other character. A tagged k-word set Φ is *conflictual* if $\exists x, y \in \Phi, x \neq y$, such that $\sigma(x) = \sigma(y)$, otherwise it is conflict-free.

For a conflict-free set Φ, we define the partial *tagging function* $\tau : \Sigma^+ \to \Sigma^{\Box}$ as $\tau(w) = \sigma^{-1}(w) \cap \mathrm{SLT}(\Phi)$, and we call $\tau(w)$ the *tagged word corresponding to* w.

We illustrate the previous definitions and some straightforward consequences.

Example 2.6. Let $k = 3$ and $\Phi = \{\#[a,\ a \odot b,\ b \odot a,\ a]\#\}$. Then the 3-SLT language is $\mathrm{SLT}(\Phi) = (a \odot b \odot)^* a$.

We observe that, for each tagged word $w \in \Sigma^{\Box}$, the set $\varphi_k(w)$ is included in the set $f_k(w)$. For instance $\varphi_3(a \odot b \odot a \odot c) \subset f_3(a \odot b \odot a \odot c)$, since f_3 contains also the 3-words $\odot b \odot, \odot a \odot$. In particular, the 3-word set derived from Φ is

$$F = \Phi \cup \{[a\odot,\ [a],\ \odot b\odot,\ \odot a\odot,\ \odot a]\}\,.$$

Yet, although $\Phi \subset F$, the languages defined by strict local testing coincide: $\mathrm{SLT}(F) = \mathrm{SLT}(\Phi)$.

In what follows all tagged word sets considered are conflict-free, unless stated otherwise. The next technical lemma and its corollary are used in later proofs.

Lemma 2.7. *Let $w \in \Sigma^{\Box k}$; let $s', s'' \in \Delta$ be two distinct tags. Then, for every $3 \leq h \leq k + 2$, the tagged word set $\varphi_h(ws'ws''w)$ is conflictual.*

Proof. Let $w = a_1 s_2 a_3 \dots s_{k-1} a_k$. It suffices to observe that the conflicting tagged h-words $t_h(a_1 s_2 a_3 \dots s_{k-1}\ a_k s' a_1)$ and $t_h(a_1 s_2 a_3 \dots s_{k-1} a_k s'' a_1)$ are contained in $\varphi_h(ws'ws''w)$. $\Box$

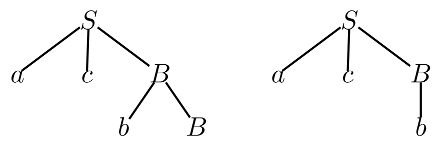
Corollary: when $w = a \in \Sigma$, for any sufficiently long word $z \in a(\Delta a)^*$, if z contains two distinct tags, the set $\varphi_k(z)$ is conflictual.

3. Higher order operator precedence

We introduce the new concept of *higher order operator precedence* (HOP) grammar, first intuitively, then formally. Then we show that OP grammars are a subcase of the lowest subfamily HOP(3) $\subset$ HOP, within the infinite hierarchy induced by the width parameter $k \in 3, 5, 7, \dots$; value 3 corresponds to the information contained in each entry of an OP matrix. The section then continues with the presentation of a deterministic push-down machine for parsing (more precisely recognizing) HOP languages.

3.1. Intuitive presentation

For a given grammar, each one of the OP relations *yield*, *take* and *equal* of Definition 2.2 essentially summarizes some patterns that may occur in a syntax tree. For instance the relation $c \lessdot b$ for the grammar G_1 of Example 2.3 says that the (partial) trees



may occur in the language. Using the parenthesis grammar, $S \to (aScB) \mid (acB)$, $B \to (bB) \mid (b)$, the same trees are represented by the words

$$(ac(bB)) \qquad (ac(b))$$

which occur as factors of some sentential forms. Thus, it is immediate to see that the factor $c(b$ represents the relation $c \lessdot b$. Similarly, relation $b \lessdot b$ would be represented by factor $b(b$, which occurs in a parenthesized word such as $(ac(b(b)))$. Notice that the relation $b \gtrdot \#$ is represented by a longer factor $b)))\#$ occurring in $\#(ac(b(b)))\#$; and, for longer words of the form $\#acbb\ldots b\#$, by a factor $b))\ldots)\#$. Clearly, as in this case, a run of two or more closed parentheses is represented by a single $\gtrdot$ tag; and the same is true for open parentheses and the $\lessdot$ tag. Therefore, when converting from parenthesized words to OP relations, we have to condense a run of identical parentheses into a single OP tag.

To avoid confusion, in our formal definition we use the syntactical tags "[" and "]" instead of $\lessdot$ and $\gtrdot$.

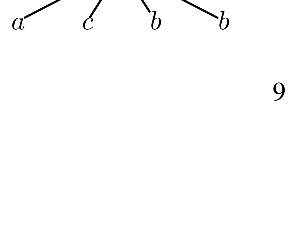
Next we examine how the relation of equal precedence appears in parenthesized sentential forms. Clearly, the relation $a \doteq c$ is represented in certain parenthesized sentential forms by the factor ac or by the factor aSc where the two terminals are separated by a nonterminal. We decide to represent the $\doteq$ tag by $\odot$. Altogether, our syntactical tags are denoted by $\Delta = \{[,], \odot\}$.

We shift now from CF to ECF grammars. To see how the concept of OP relations continues to work for ECF grammars, we define the same language of Example 2.3 by means of the equivalent ECF grammar G_{1E} and we show the corresponding new OP matrix:

$$G_{1E} = \{S \to aScb^+ \cup acb^+\} \qquad M_{1E} = \begin{array}{c|c|c|c|c} & a & b & c & \# \\ \hline a & [& & \odot & \\ \hline b & & \odot &] &] \\ \hline c & & \odot & & \\ \hline \# & [& & & \odot \end{array}$$

For brevity, in the rule $S \to R_S$, language R_S is specified by a regular expression.

To explain the change of entries of the OP matrix M_{1E} (apart from the use of Δ symbols instead of OP tags) with respect to matrix M of Example 2.3, we examine for word $aacbbcb$ the syntax tree, the parenthesized word, and the parenthesized sentential forms:



$(a(acbb)cb)\,, \quad (aScb)$

By converting as explained from parenthesized sentential forms to OP relations, we see that factor $a(a$ expresses the relation $a[a$, (i.e., $a \lessdot a$); factors aSc, cb, bb and ac express the relations $a \odot c$, $c \odot b$, $b \odot b$; factor $b)c$ expresses relation $b]c$. It is important to notice that all such factors contain exactly two terminal characters. Since each factor encodes an OP relation, which is a 3-tuple of the form $\langle a, s, b\rangle$, where $a, b \in \Sigma$ and $s \in \Delta$, the OP model for ECF grammars will be called a higher order precedence relation model of order 3, $\text{HOP}(3)$ for short.

So far we have simply shown how OP relations also for ECF grammars are visible in parenthesized sentential forms. The next fundamental step considers factors that contain more than two terminal characters, and thus permits to overcome many OP conflicts. We introduce the idea informally on the next example.

Example 3.1. We list a grammar G_2, with $L(G_2) = \{a^n \left(ba^2c\right)^n \mid n \geq 1\}$, and its conflictual precedence relations:

$$G_2 = \{S \to aSbaac \mid abaac\} \qquad M_2 = \begin{array}{c|c|c|c|c} & a & b & c & \# \\ \hline a & [,\odot & \odot & \odot & \\ \hline b & \odot & & & \\ \hline c & &] & &] \\ \hline \# & [& & & \odot \end{array}$$

The conflict $a[a$ vs $a \odot a$ in matrix M_2 is due to the factors $a(a$ and aa that occur in both the parenthesized sentential forms: $(a(a(aSbaac)\ baac)\ baac)$ and $(a(a(abaac)$ $baac)\ baac)$. Such conflict is remedied if we look at the syntactical tags that occur in longer factors, containing 3 terminal characters. We list all such factors:

Factors of parenthesis grammar	Tagged k-factors ($k = 5$)
$a(a(a$	$a[a[a$
$a(ab$ and $a(aSb$	$a[a \odot b$
aac	$a \odot a \odot c$
baa	$b \odot a \odot a$

The tagged 5-word in the right column are conflict-free, because their projections on Σ are all different: aaa, aab, aac, baa. Completing the example, we list all tagged 5-words that may occur for grammar G_2, denoted by $\varphi_5(G_2)$:

$$\varphi_5(G_2) = \left\{ \begin{array}{l} a \odot c]\#,\ \# \odot \#[a,\ c]b \odot a,\ \#[a \odot b,\ b \odot a \odot a,\ a[a \odot b, \\ a \odot b \odot a,\ a \odot c]b,\ \#[a[a,\ \# \odot \# \odot \#,\ a \odot a \odot c,\ a[a[a,\ c]\# \odot \# \end{array} \right\}$$

Since no conflict occurs in $\varphi_5(G_2)$, we say that grammar G_2 has the higher OP property of order 5, in short $\text{HOP}(5)$.

We are now ready to formalize the above concepts.

3.2. Definition of higher order operator-precedence grammar

Instead of passing through the intermediate step of the parenthesis grammar as we did in the intuitive presentation, we directly define a new grammar generating the

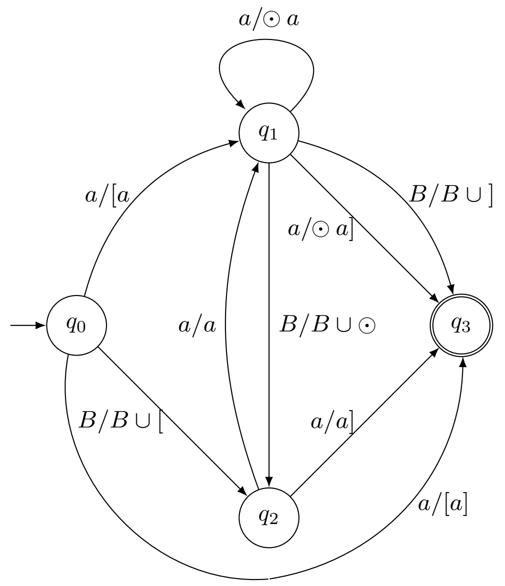


Figure 2: Transducer for the finite-state translation $f : V^+ \to 2^{(V \cup \Delta)^+}$ used in Definition 3.2 to construct the tagged grammar rules.

tagged language, whose sentential forms contain exactly all the tagged words of the original grammar. Informally, the main idea is to insert the tags $\odot$ as separators between terminal symbols in rules r.h.s., and to insert the brackets, to make structure visible, at the beginning and end of a rule, only when strictly needed.

Definition 3.2 (Tagged grammar)**.** Let $G = (V_N, \Sigma, P, S)$ be an ECF grammar in operator form. The *tagged grammar* associated to G, denoted by $\overline{G}$, is defined as $\overline{G} = (V_N, \Sigma \cup \Delta, \overline{P}, S)$ where each rule $X \to R_X \in P$ is replaced in $\overline{P}$ by the rule $X \to \overline{R}_X$ such that the regular language $\overline{R}_X \subseteq (V \cup \Delta)^*$ is defined as $\overline{R}_X = f\,(R_X)$ and $f : V^+ \to 2^{(V \cup \Delta)^+}$ is the multi-valued finite-state translation, specified in Figure 2.
A word of the language $\text{⊕}L(\overline{G})\text{⊕}$ is called a *tagged form* generated by grammar G.

Notice in Figure 2 that each nonterminal in the input can be left unchanged or replaced by a tag, in accordance with the following criterion. The tag is "[" if first character, "]" if last character, otherwise $\odot$. We notice that an output word must start with "[" or a nonterminal; and it must end with "]" or a nonterminal. The input terminals are left unchanged and are separated by $\odot$ in the output. Clearly, no word in $\overline{R}_X$ may contain adjacent terminals, adjacent tags, or adjacent nonterminals. Therefore, grammar $\overline{G}$ is in operator form.

Tagged grammars are akin to the classical parenthesis grammars [24], yet their representation of nested structures is more parsimonious, since a single "[" tag (analogously a "]") can represent a run of many open (resp. closed) parentheses.

We extend the operator φ_k of Definition 2.5 to ECF grammars in the following definition.

Definition 3.3. Let $k \geq 3$. The *set of tagged k-words of grammar G* is denoted by $\varphi_k(G) \subseteq \Sigma^{\Box k}$ and defined as $\varphi_k(G) = \varphi_k\left(\circledast L(\overline{G}) \circledast\right)$.

Thus, each word of $\varphi_k(G)$ is obtained by considering the k-tagged words occurring in the tagged forms of G.

The set $\varphi_k(G)$ plays the role the OP matrix had for OP grammars, in the next central definition.

Definition 3.4. Let $k \geq 3$. Grammar G and its language $L(G)$ have the *higher operator precedence property of order k*, in short HOP(k), if the tagged k-word set of grammar G, $\varphi_k(G)$, is conflict-free, i.e.

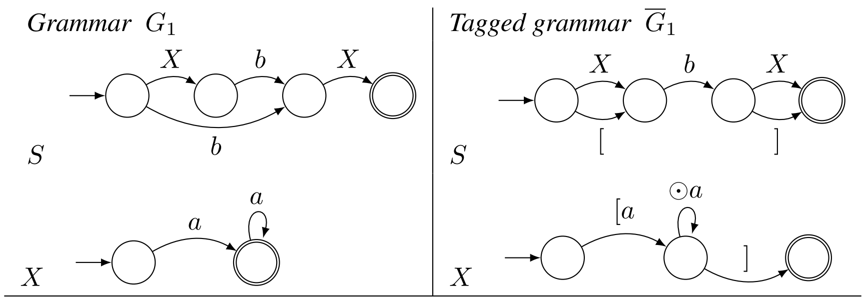
$$\nexists u, v \in \varphi_k(G) \text{ such that } u \neq v \text{ and } \sigma(u) = \sigma(v). \tag{1}$$

The union, for all finite k, of the families HOP(k) is denoted by HOP. The family of grammars in HOP(k) having the same set Φ of tagged k-words is denoted by HOP(Φ). Identical notations denote the corresponding language families, when no confusion arises.

Clearly, if a grammar has the HOP property of order k, then it has the same property for any larger odd value of the parameter.

The decidability of Condition (1) for any given grammar G and for a fixed value of k, is an immediate consequence of the following question, which is a special case of a well-known decidable question for context-free grammars [25]. Let $w \in \Sigma^{\Box k}$ and X be a nonterminal of the tagged grammar $\overline{G}$; $\exists u$ and $\exists v$, such that $X \overset{*}{\Longrightarrow}_{\overline{G}} uwv$, where u and v are possibly empty terminal words?

However, it is not known whether for a given grammar it is decidable if there exists a value k such that G has the HOP(k) property.



$$\varphi_3(G_1) = \{a \odot a, a[b, b]a, \#[a, a]\#, \# \odot \#, \#[b, b]\#\}$$

Figure 3: Top: grammar and tagged grammar of Example 3.5. Bottom: set of tagged 3-words.

Example 3.5. We show in Figure 3 a grammar G_1 and the associated tagged grammar $\overline{G}_1$, where for conciseness we permit transitions reading more than one symbol, with the obvious meaning. Notice that in general $\sigma(L(\overline{G}_1)) \supseteq L(G_1)$, since extra terminal words may be generated by tagged grammar rules where a tag replaces a nonterminal. In the example, the word $[b]$ is grammatical for $\overline{G}_1$, while $\sigma([b]) = b \notin L(G_1)$.

It is easy to check that the set $\varphi_3(G_1)$ defined in Definition 3.3, listed at the bottom of Figure 3, coincides with the set of tagged subwords of length 3 that occur in the words of $\#L(\overline{G}_1)\#$. Since such set is conflict-free, G_1 is a HOP(3) grammar.

We have argued that the precedence relations of OP grammars can be represented by tagged 3-words, therefore such grammars have the HOP(3) property. Then, a more subtle question comes: is the family of OP languages strictly included within the family of HOP(3) languages? Since HOP(3) grammars only differ from OP grammars by having ECF instead of CF rules, the question is whether there exists an ECF grammar in HOP(3) that does not have an equivalent CF grammar in OP. Surprisingly, the answer is affirmative, in contrast to what happens with the deterministic CF grammar families LR(1) and LL(1), where ECF rules may be used for convenience but do not enlarge the class of languages (see, e.g., [26]).

Theorem 3.6 (Inclusion of OP)**.** *Every operator precedence (OP) grammar has the HOP*(3) *property. The family of OP languages is strictly included within the family HOP*(3).[1]

Proof. Having already shown that the OP grammar family coincides with the family of (not extended) CF grammars having the HOP(3) property, it remains to prove the second proposition. Let $L_{\text{loop}} = L_1 \cup L_2 \cup L_3 \cup L_4$ where $L_1 = \{a^n(bc)^n \mid n > 0\}$, $L_2 = \{b^n(ca)^n \mid n > 0\}$, $L_3 = \{c^n(ab)^n \mid n > 0\}$, $L_4 = (abc)^+$. First, we prove that any CF grammar for L_{loop} necessarily has OP conflicts,[2] then we show an ECF grammar with the HOP(3) property

For each language L_i, $i \in 1 \ldots 4$, we apply the pumping lemma to determine all valid factorizations of sentences. Then, for each factorization we compute the corresponding OP relations. From the pumping lemma, the words of L_1 can be factorized in two ways: $a^n(bc)^n$ or $a^n b(cb)^m c$ for suitable values of m and n. The syntax trees corresponding to the two factorizations impose the OP relations shown in rows L_1 of Table 1. For the similar cases L_2, L_3, Table 1 lists the OP relations.

Case L_4 is a regular language that admits only three factorizations having a Chomsky type 3 structure: $(abc)^n$, $(bca)^n$, $(cab)^n$. Different OP relations are imposed by each factorization depending on whether left-linear or right-linear structure is chosen. The possible cases are listed in the table.

Notice that other factorizations are possible which contain as iterated factors two different circular permutations of abc; an example is $(abc)^m a(bca)^n bc$. By Lemma 2.7,

[1]The proof of this proposition is based on an idea by Dino Mandrioli.

[2]We observe that this is not in contrast with the closure properties of OP grammars under union, concatenation and Kleene star, because the latter closure (Theorem 16 of [10]) requires that the equal in precedence relation be acyclic, whereas language L_{loop} in our proof necessarily has a cycle in the $\odot$ relation.

Table 1: OP relations imposed by factorizations compatible with pumping lemma.

L_1	
$a^n(bc)^n$	$a[a$ and $c]b$ or
$a^n b(cb)^m c$	$a[a$ and $b]c$
L_2	$b[b$ and $($ $c]a$ or $a]c$ $)$
L_3	$c[c$ and $($ $a]b$ or $b]a$ $)$
L_4	
$(abc)^n$	$a \odot b$ and $b \odot c$ and $($ $c]a$ or $c[a$ $)$
$(bca)^n$	$b \odot c$ and $c \odot a$ and $($ $a]b$ or $a[b$ $)$
$(cab)^n$	$c \odot a$ and $a \odot b$ and $($ $b]c$ or $b[c$ $)$

all such factorizations present OP conflicts and can be excluded. Since all possibilities displayed in Table 1 are conflictual, no OP grammar for L_{loop} exists.

On the other hand, it can be checked that the following ECF grammar generates L_{loop} and has the set of tagged 3-words of matrix (2):

$$X_1 \to aX_1bc \mid abc, \quad X_2 \to bX_2ca \mid bca, \quad X_3 \to cX_3ab \mid cab, \quad X_4 \to (abc)^+$$

$$\begin{array}{c|c|c|c} & a & b & c \\ \hline a & [& \odot &] \\ \hline b &] & [& \odot \\ \hline c & \odot &] & [\end{array} \tag{2}$$

□

This theorem has practical import for the definition of technical languages by means of OP grammars: it says that permitting regular expressions in their rules may increase generative capacity not just expressiveness or conciseness.

Suitability for practical use. We spend a few words on the potential advantage of using HOP grammars instead of OP grammars for defining and parsing technical languages. OP grammars have been and are still preferred to LR(1)/LL(1) grammars in compilers that require very fast parsing. But adapting a CF, or worse an Extended CF, grammar for OP has some cost. Quite often, the reference grammars of technical languages, even when already in operator form, have OP conflicts. In most cases it is possible to eliminate conflicts by manual grammar transformations, which however may damage grammar terseness and structural adequacy. In such cases, if the original grammar meets the HOP(k) condition for a value $k > 3$, it can be used without change. We show an example of such situation in Appendix 1 for a toy language including some typical constructs of a programming language.

3.3. Parsing algorithm

To extend to HOP the traditional parsing algorithm for OP languages, we define here a model of deterministic push-down automaton (DPDA) and we prove that it is able to recognize any HOP language.

Consider a HOP(Φ) ECF grammar $G = (\Sigma, V_N, P, S)$, with rules $X \to M_X$, equivalently written as $X \to R_X$ where R_X is the language recognized by M_X. We assume, without loss of generality, that each automaton $M_X = (V, Q_X, \delta_X, q_{0,X}, T_X)$ is deterministic, with δ_X total. We also recall the assumptions that, for any two rules $X \to R_X$ and $Y \to R_Y$, $R_X \cap R_Y = \emptyset$ and $V_N \cap R_X = \emptyset$.

We loosely describe the DPDA, before entering the formal definition. The states of the push-down automaton are of two kinds, which respectively correspond to input reading moves, and to ε-moves. A state of the former kind remembers the last $\lfloor k/2 \rfloor$ input symbols read by the automaton. A state of the latter kind is entered right after a pop move, and contains two pieces of information, (X, w): the name X of the recognized nonterminal, and a string w of $\lceil k/2 \rceil$ terminal symbols, that will be used to select the next move.

Each stack symbol carries two pieces of information: the first piece is made of $\lceil k/2 \rceil$ terminal symbols and stores the state at the push, while the second component is a state of a deterministic FA (defined below) called *driver*, which is used to recognize a handle.

The idea is that, in each input reading move, the DPDA checks its state w, reads a terminal a, then performs a move depending on the tagging imposed on string wa by the HOP grammar G, i.e., $\tau(wa)$. More precisely, the last tag decides the move as follows: "[" for a pushing move, $\odot$ for a move just changing the driver state stored on the stack, "]" for a popping move. On a ε-move, i.e. a move from a state (X, w), the DPDA uses the driver to check the found nonterminal X, and performs a push/pop/internal move depending on the last tag of $\tau(w)$, like in reading moves. The string w represents the state at the last push move, together with the last read symbol; informally, w stores the "context" in which X was found.

The driver automaton is formally defined next.

Definition 3.7. The *driver automaton* $D = (V, Q_D, \delta_D, q_{D,0}, T_D)$ is the classical product machine that accepts the language $\bigcup_{X \in V_N} L(M_X)$. More precisely, Q_D, and T_D, are, respectively, the Cartesian product, for all $X \in V_N$, of the sets Q_X, and of the sets T_X. The transition function δ_D is defined in the obvious way.

To properly handle reductions, we also introduce a *labeling function* $\lambda : T_D \to V_N$, which associates to each final state of the driver the (unique thanks to the previous assumptions) nonterminal X of the right part R_X recognized.

For simplicity, in the next definition we present only the language recognizer; of course, it can be extended as usual to compute parse trees.

Definition 3.8 (parser DPDA)**.** We build a DPDA $P_G = (\Sigma, Q, \Gamma, Z_0, \delta, q_0, F)$. We assume that the stack grows from left to right. For convenience, we set $h = \lfloor k/2 \rfloor$.

Here it is how P_G is constructed:

- set of states $Q = \Sigma^h \cup (T_D \times \Sigma^{h+1})$; notice that states are identified either by a terminal h-word, or by a final state of the driver coupled with a terminal $(h+1)$-word;
- stack alphabet $\Gamma = \Sigma^h \times V_N \cup \{Z_0\}$;

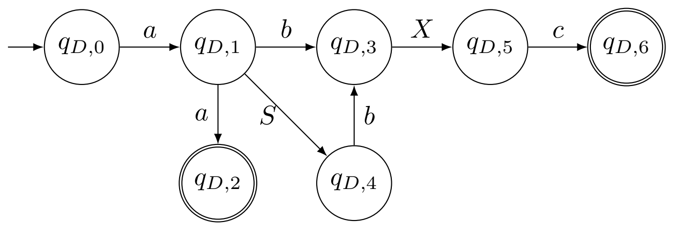


Figure 4: Driver automaton of Example 3.9; where $\lambda(q_{D,2}) = X$, and $\lambda(q_{D,6}) = S$.

- initial stack symbol Z_0;
- initial state $q_0 = \#^h$;
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \times \Gamma \to Q \times \Gamma^*$, with the following constraints, for some $a \in \Sigma$, $C \in \Gamma$, $w, w' \in \Sigma^h$, $q_{D,i}, q_{D,j} \in Q_D$, $q_{D,j} \in F_D$, $Z, Y \in V_N$, $w_1 \in \Sigma^{\square k-2}$:
    1. $\delta(w, a, C) = (t_h(wa), C\ (w, \delta_D(q_{D,0}, a)))$,
       when $\tau(wa) = w_1[a$;
    2. $\delta(w, a, (w', q_{D,i})) = (t_h(wa), (w', \delta_D(q_{D,i}, a)))$,
       when $\tau(wa) = w_1 \odot a$;
    3. $\delta(w, a, (w', q_{D,j})) = ((\lambda(q_{D,j}), w'a), \varepsilon)$,
       when $\tau(wa) = w_1]a$;
    4. $\delta((Y, wa), \varepsilon, C) = (t_h(wa), C\ (t_h(wa), \delta^*_D(q_{D,0}, Ya)))$,
       when $\tau(wa) = w_1[a$;
    5. $\delta((Y, wa), \varepsilon, (w', q_{D,i})) = (t_h(wa), (w', \delta^*_D(q_{D,i}, Ya)))$,
       when $\tau(wa) = w_1 \odot a$;
    6. $\delta((Z, wa), \varepsilon, (w', q_{D,j})) = ((\lambda(q_{D,j}), w'a), \varepsilon)$,
       when $\tau(wa) = w_1]a$;
- $F = \{(q_{D,j}, \#^{h+1}) \mid \lambda(q_{D,j}) \in S\}$.

Acceptance is by final state and empty stack; in addition, for convenience, the language accepted by P_G is end-marked, i.e., it is $L(G) \cdot \{\#\}$.

Example 3.9. Consider a variant of G_2 of Example 3.1, slightly enriched to illustrate more features:

$$G'_2 = \{S \to aSbXc \mid abXc, X \to aa\}\,.$$

The resulting tagged word set is the following:

$$\varphi_5(G'_2) = \left\{ \begin{array}{c} b \odot c]b,\ \# \odot \#[a,\ a]c]b,\ c]b[a,\ \#[a \odot b,\ c]b \odot c, \\ a[a \odot b,\ a \odot b[a,\ b[a \odot a,\ a \odot a]c,\ a]c]\#,\ b \odot c]\#, \\ \#[a[a,\ \# \odot \# \odot \#,\ a[a[a,\ c]\# \odot \#,\ a \odot b \odot c \end{array} \right\}$$

The driver automaton is reported in Figure 4. We now illustrate the behavior of DPDA $P_{G'_2}$, by tracing the steps for input $aabaacbaac\#$.

State	Input	Stack contents	Tagged word
$\#\#$	$aabaacbaac\#$	Z_0	$\# \odot \#[a$
$\#a$	$abaacbaac\#$	$Z_0\ (\#\#, q_{D,1})$	$\#[a[a$
aa	$baacbaac\#$	$Z_0\ (\#\#, q_{D,1})(\#a, q_{D,1})$	$a[a \odot b$
ab	$aacbaac\#$	$Z_0\ (\#\#, q_{D,1})(\#a, q_{D,3})$	$a \odot b[a$
ba	$acbaac\#$	$Z_0\ (\#\#, q_{D,1})(\#a, q_{D,3})(ab, q_{D,1})$	$b[a \odot a$
aa	$cbaac\#$	$Z_0\ (\#\#, q_{D,1})(\#a, q_{D,3})(ab, q_{D,2})$	$a \odot a]c$
(X, abc)	$cbaac\#$	$Z_0\ (\#\#, q_{D,1})(\#a, q_{D,3})$	$a \odot b \odot c$
bc	$baac\#$	$Z_0\ (\#\#, q_{D,1})(\#a, q_{D,6})$	$b \odot c]b$
$(S, \#ab)$	$aac\#$	$Z_0\ (\#\#, q_{D,1})$	$\#[a \odot b$
ab	$aac\#$	$Z_0\ (\#\#, q_{D,3})$	$a \odot b[a$
ba	$ac\#$	$Z_0\ (\#\#, q_{D,3})(ab, q_{D,1})$	$b[a \odot a$
aa	$c\#$	$Z_0\ (\#\#, q_{D,3})(ab, q_{D,2})$	$a \odot a]c$
(X, abc)	$c\#$	$Z_0\ (\#\#, q_{D,3})$	$a \odot b \odot c$
bc	$\#$	$Z_0\ (\#\#, q_{D,6})$	$b \odot c]\#$
$(S, \#\#\#)$	ε	Z_0	accept

Theorem 3.10. *For any k, for any grammar G in HOP(k), let P_G be the DPDA of Definition 3.8. Then $L(P_G) = L(G) \cdot \{\#\}$.*

Proof. The proof is by induction on derivation steps of the grammar. Without loss of generality, only rightmost derivations are assumed. We also use a modified version of G with the new axiom S' and a rule $S' \to S\#$ to take the ending symbol into account.
Base case:

$$x = w_1 w a w_2 \Longleftarrow_G w_1 W a w_2,$$

$w_1, w_2 \in \Sigma^*$, $w \in \Sigma^+$, for some rule $W \to M_W$ and $w \in L(M_W)$ iff

$$\langle \#^h, w_1 w a w_2, Z_0 \rangle \overset{+}{\vdash}_{P_G} \langle (W, t_h(w_1)a), w_2, \gamma(t_h(w_1'), \delta_D^*(q_{D,0}, w_1'')) \rangle,$$

where $\gamma \in \Gamma^+$, for $w_1 = w_1' w_1''$, such that $\tau(w_1) = \tau(w_1')[\, \tau(w_1'')$, and $\tau(w_1'')$ does not contain [.

Being the grammar HOP(k), it is $\tau(w_1 w a) = \tau(w_1)[\tau(w)]a$, where no tag in $\tau(w_1)$ is], and all tags in $\tau(w)$ are $\odot$. Hence, P_G reads w_1 and performs either moves in which a new symbol is pushed on the stack (whenever a [tag with the next read symbol is encountered), or moves in which the top stack symbol is updated (for a tag $\odot$). By reading w, P_G follows the driver automaton (which has a state component encoding M_W) to update its stack's top-most right symbol, with a left component storing the state (i.e. the look-back of h symbols) in which P_G was when the [corresponding to the handle for w was found.
It is easy to see that also the vice versa holds, since those P_G moves correspond to the structure in which w is the first handle found.
Induction case:

$$w_1 w a w_2 \Longleftarrow_G w_1 W a w_2,$$

$w_1, w_2 \in V^*$, $w \in V^+$, for some rule $W \to M_W$ and $w \in L(M_W)$ iff

$$\langle q_1, w a w_2, \gamma(q_2, q_D') \rangle \overset{+}{\vdash}_{P_G} \langle (W, q_1 a), w_2, \gamma(q_2, q_D') \rangle,$$

where $\gamma \in \Gamma^+$.

There are two cases: either $q_1 \in \Sigma^h$, or $q_1 = (X, q_3 b)$, for some $q_3 \in \Sigma^h$, $b \in \Sigma$.
Case 1: Being G HOP(k), it is $\tau(q_1 w) = \tau(q_1)[\tau(w)$; hence, if $w = bw_3$,
$\langle q_1, waw_2, \gamma(q_2, q'_D)\rangle \vdash_{P_G} \langle t_h(q_1 b), w_3 a w_2, \gamma(q_2, q'_D)(q_1, \delta_D(q_{D,0}, b))\rangle$.
Case 2: Being $q_1 = (X, q_3 b)$, there will be an ε-move depending on the last tag in $\tau(q_3 b)$: for [and $\odot$ we go back to Case 1; for] we continue the reductions (i.e., Case 2), until we reach Case 1.

Next, P_G reads w_3 following the tags in $\tau(w_3)$: by inductive hypothesis, all the handles found in there are reduced, recognizing the correct nonterminals, and then read, updating the right component in the top of the stack:
$\langle t_h(q_1 b), w_3 a w_2, \gamma(q_2, q'_D)(q_1, \delta_D(q_{D,0}, b))\rangle \overset{+}{\vdash}_{P_G} \langle q'_1, aw_2, \gamma(q_2, q'_D)(q_1, q_{D,i})\rangle$,
for some $q'_1, q_{D,i}$. Hence, P_G read w by following D, which encodes the rule $W \to M_W$. The reached state $q_{D,i}$ must be accepting, with $\lambda(q_{D,i}) = W$. Next, from the HOP(k) property, it follows $\tau(wa) = \tau(w)]a$. This means that the next step for P_G is the following: $\langle q'_1, aw_2, \gamma(q_2, q'_D)(q_1, q_{D,i})\rangle \vdash_{P_G} \langle (W, q_1 a), w_2, \gamma(q_2, q'_D)\rangle$.

The vice versa is simpler, since every move of P_G reading w is done according to the tags in $\tau(w)$, and by the induction hypothesis, every handle found in w is reduced to the correct nonterminal. □

Theorem 3.11. *The family of HOP languages is strictly contained in the family DET of deterministic context-free languages and in the family $\{L \mid L^R \in DET\}$ of deterministic in reverse languages.*

Proof. Consider the language $L = \{a^n b a^n \mid n \geq 1\}$, which is deterministic and deterministic in reverse, and assume that $L = L(G)$ for some G in HOP(k). By the pumping lemma, it is clear that, for any value of k, both the tagging $a[a$ and $a]a$ must be present in some tagged k-words of G. Clearly, we can find a sufficiently long word $w \in L$ such that both $a[a$ and $a]a$ occur in the corresponding tagged word $\tau(w)$. From Lemma 2.7 it follows that $\varphi_k(G)$ is conflictual. □

4. Closure properties, maximality and hierarchies

It is natural to ask whether the rich closure properties of OP languages continue to hold for HOP languages. We prove that the answer is positive in all but one of the cases investigated, which remains open in some sense. We start from the reversal operation.

Theorem 4.1. *For every k, HOP(k) is closed under reversal.*

Proof. Given an HOP(k) grammar G, we construct G' such that $L(G') = L(G)^R$. For every rule of G, $X \to R_X$, G' contains rule $X \to R'_X$, where R'_X is the reversal of R_X. It is easy to see that all the factors of $L(G)$ are reversed in $L(G')$. Consider the homomorphism $\rho : \Sigma \cup \Delta \to \Sigma \cup \Delta$, $\rho([) =]$, $\rho(]) = [$, $\rho(a) = a$ otherwise. Clearly, $\rho(\varphi_k(G')) = \rho(\varphi_k(G)^R)$. Hence, $\varphi_k(G')$ is conflict-free, and G' is in HOP(k). □

4.1. Boolean closure

To prove the Boolean closure of HOP(Φ) for each conflict-free set Φ of order k, we need the following lemma which extends Theorem 5 of Knuth [18] from CF to Extended CF grammars.

Lemma 4.2. *Let $G_{(\,),1}$ and $G_{(\,),2}$ be ECF parenthesis grammars. Then there exists an ECF parenthesis grammar $G_{(\,)}$ such that $L(G_{(\,)}) = L(G_{(\,),1}) - L(G_{(\,),2})$.*

The proof differs from the one in [18] just in some details and is in Appendix 2.

Theorem 4.3. *For every k and for every conflict-free set $\Phi \subset \Sigma^{\Box k}$, the language family HOP(Φ) is closed under union, intersection and set difference.*

Proof. Let $L_i = L(G_i)$ where for $i = 1, 2$ grammar $G_i = (V_{N_i}, \Sigma, P_i, S_i)$ is in HOP(Φ) and without loss of generality we assume that V_{N_1} and V_{N_2} are disjoint.

Union Define the grammar $G = (V_{N_1} \cup V_{N_2}, \Sigma, P_1 \cup P_2, S_1 \cup S_2)$. Clearly, G generates $L(G_1) \cup L(G_2)$ and belongs to family HOP(Φ) since its set of tagged k words is Φ.

Set Difference Let $G_{(\,),i}$ be the parenthesis grammar of G_i, $i = 1, 2$, and by Lemma 4.2 let $G_{(\,)} = (V_N, \Sigma, P, S)$ be the parenthesis grammar such that $L(G_{(\,)}) = L(G_{(\,),1}) - L(G_{(\,),2})$. Preliminarily, we observe that, for each structurally unambiguous grammar G and for the associated parenthesis grammar $G_{(\,)}$, the correspondence between their respective sentences x and y induced by the identity $x = \pi(y)$ is bijective.
Define the grammar $G = (V_N, \Sigma, P, S)$ obtained by erasing the parentheses from each rule of $G_{(\,)}$. We prove that $L(G) = L(G_1) - L(G_2)$.
First, if $x \in L(G)$ then there exists a word $y \in L(G_{(\,)}) = L\left(G_{(\,),1}\right) - L\left(G_{(\,),2}\right)$ such that $\pi(y) = x$. From the bijective correspondence between the sentences of a grammar and of the corresponding parenthesis grammar, it follows that $x \in L(G_1) - L(G_2)$.

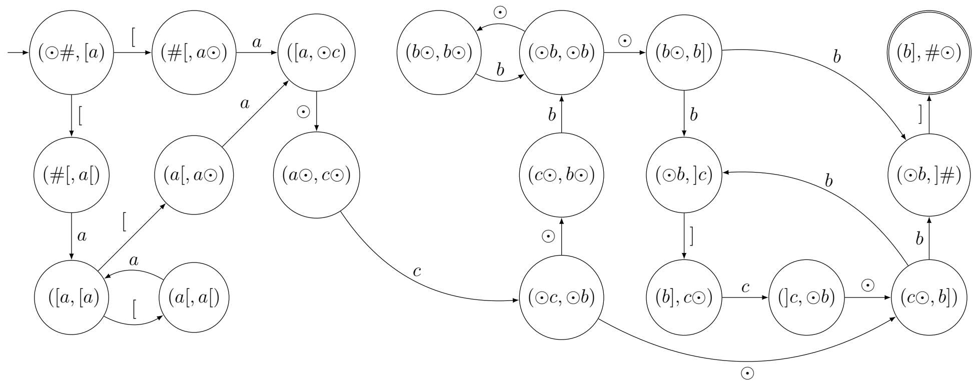
Second, let $x \in L(G_1) - L(G_2)$ and let y be the word of $L(G_{(\,),1})$ such that $x = \sigma(y)$. Since $x \notin L(G_2)$, there is no word y in $L(G_{(\,),2})$ such that $x = \pi(y')$. Therefore y is in $L(G_{(\,)})$ and $x = \sigma(y) \in L(G)$.

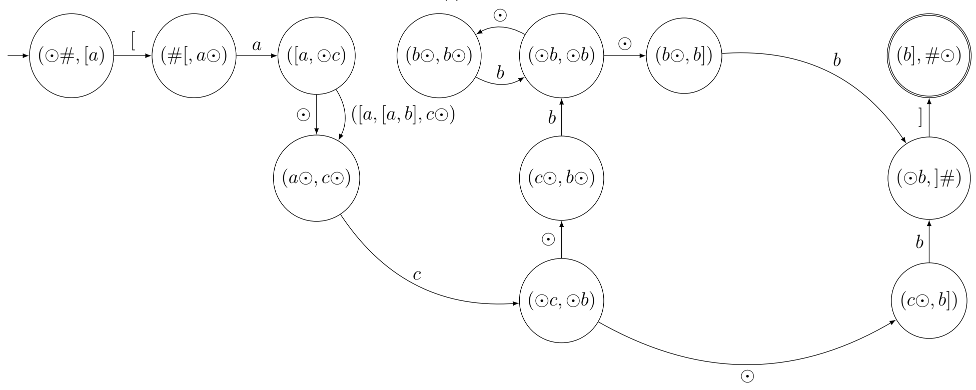
Intersection From the equality $A \cap B = A - (A - B)$. □

The preceding proof is analogous to the proof in [6] that each family of OP languages having the same set of operator precedences is closed under the three Boolean operations. Both the new and the old proofs exploit the property of structural unambiguity and the fact that the family of parentheses languages is closed under the same operations. Thus, the only difference is that here we have dealt with ECF grammars.

4.2. Maximal language

Refining the preceding theorem, we proceed to show that each language family HOP(Φ) has a unique maximal element with respect to set inclusion, to be called *max-language*. To prove this and other properties we need a new nondeterministic version of the finite-state machine that recognizes an SLT language.


(i)


(ii)

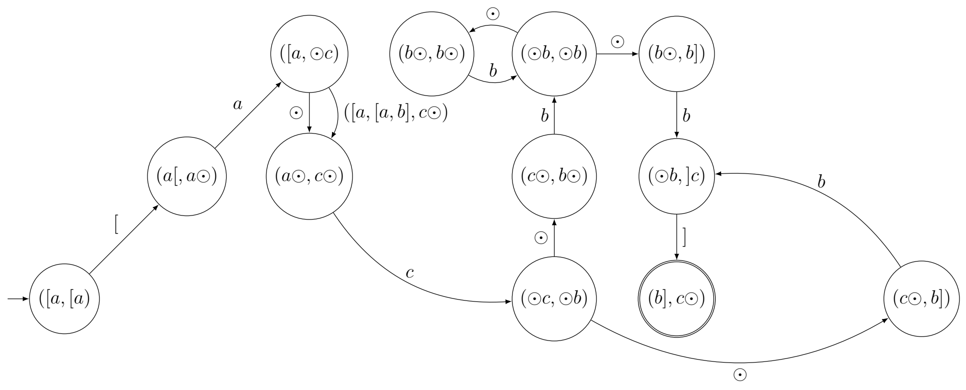

(iii)

Figure 5: (i) Symmetrical FA A_Φ of Example 4.6. (ii) Automaton of the rule $X \to [a(\odot \cup Y)c \odot (b\odot)^* b]$ of Example 4.10. (iii) Automaton of the rule $Y \to [a(\odot \cup Y)c \odot (b\odot)^* b]$ of Example 4.10.

4.2.1. Symmetric recognizer of SLT languages

To recognize an arbitrary SLT language (Definition 2.4), we define a nondeterministic FA which is symmetrical w.r.t. the scanning direction. This property differentiates such FA from the standard sliding-window DFA commonly used to recognize SLT languages (see, e.g., [23]).

Definition 4.4 (symmetrical FA)**.** Let Υ be a generic alphabet and $F \subseteq \Upsilon^k$ be a k-word set, $k \geq 2$. The *symmetrical automaton* A associated to F is obtained by trimming the FA A_0 defined as follows:

- $A_0 = (\Upsilon, Q, \delta, I, T)$, where:
- $Q = \big\{(\beta, \alpha) \in \Upsilon^{k-1} \times \Upsilon^{k-1} \mid \beta, \alpha \in f_{k-1}(F)\big\}$
- $(\beta, \alpha) \xrightarrow{a} (\beta', \alpha') \in \delta$ if, and only if, $\beta' = t_{k-1}(\beta\, a)$ and $\alpha = i_{k-1}(a\, \alpha')$
- $I = \{(t_{k-1}(\circledast), \alpha) \in Q\},\ T = \{(\beta, i_{k-1}(\circledast)) \in Q\}$.

Thus, each state is identified by two $(k-1)$-words β and α, which respectively represent the look-back and look-ahead of the state, i.e., the last $k-1$ characters already read or the next $k-1$ characters to be read by the computation traversing the state. We show that the symmetrical FA is correct and unambiguous.

Lemma 4.5 (correctness and unambiguity)**.** *Let F be a k-word set, and $A = (\Upsilon, Q, \delta,$ I, $T)$ be the* symmetrical FA *associated to F. Then A recognizes* $\mathrm{SLT}(F)$ *and is unambiguous.*

Proof. First, the identity $L(A) = \mathrm{SLT}(F)$ is immediate since, on each accepting path, the k-factors are by construction those of F.
To prove unambiguity, consider a word $x = uyv \in \Upsilon^+$, where u and v may be empty, and assume by contradiction that two computation paths accept x, respectively traversing the state sequences $\pi_u \pi_y \pi_v$ and $\pi_u \pi'_y \pi_v$, with π_u and π_v possibly empty. By construction of A, paths π_y and π'_y have equal length. Moreover, if $\pi_y = q_1 q_2 \ldots q_t$ and $\pi'_y = q'_1 q'_2 \ldots q'_t$, then $q_1 = (t_{k-1}(u), i_{k-1}(y))$ and also $q'_1 = (t_{k-1}(u), i_{k-1}(y))$ and the two states coincide. Then, such identity holds for every subsequent step, hence $\pi'_y = \pi_y$ and the two paths are identical, a contradiction. $\square$

To illustrate, we present the symmetrical FA associated to a set $\Phi_k \subseteq \Sigma^{\square k}$ of k-words that are tagged, since that is what we need later. We recall from Definition 2.5 that, given Φ, the *k-strictly locally testable tagged language* defined by Φ coincides with the SLT language defined by the set $F \subset (\Sigma \cup \Delta)^k$ such that $\Phi \subset F$ and $\sigma(\Phi) = \sigma(F)$, in formula $\mathrm{SLT}(F) = \mathrm{SLT}(\Phi)$. Therefore, the construction of Definition 4.4 holds also for a tagged k-word set.

Example 4.6. Figure 5 (i) shows the symmetrical FA A_Φ recognizing the language $\mathrm{SLT}(\Phi)$ defined by the conflict-free set

$$\Phi = \{\# \odot \#,\ \#[a,\ b]\#,\ b]c,\ c \odot b,\ b \odot b,\ a \odot c,\ a[a\}.$$

It is easy to see that A_Φ is unambiguous since, from any state q, any two computations of length 3 respectively reading the words $a_1a_2a_3$ and $b_1b_2b_3$ such that $\delta(q,a_1) \neq \delta(q,b_1)$ and $\delta^*(q,a_1a_2) \neq \delta^*(q,b_1b_2)$, necessarily differ in their third character, i.e., $a_3 \neq b_3$. E.g., observe the computations:

$$([a, [a) \xrightarrow{[a[} (a[, a[), \quad \text{and} \quad ([a, [a) \xrightarrow{[a\odot} (a\odot, c\odot).$$

4.2.2. Reductions and maximal grammar

We show that a language of tagged words, $SLT(\Phi)$, defined by a set of conflict-free k-words, can be interpreted as defining another language over terminal alphabet Σ; such language is context-free but not necessarily regular, and is called a *maximal language or max-language*, denoted by $L(\Phi)$. We anticipate the reason of the name "max-language": such language belongs to family HOP(Φ) and includes any other language in the same family.

First we present the intuition, then we formally define max-languages. A word w is in $L(\Phi)$ if its corresponding tagged word $y = \tau(w)$ can be reduced to the tag $\odot$ by applying a series of canceling operations called reductions. A reduction cancels a substring of the form $[a_1\odot\ldots\odot a_m]$ and replace it by a single tag s, under the condition that the resulting tagged word is in the regular language SLT(Φ). We formalize such reduction process.

Definition 4.7 (reduction and max-language)**.** Let $\Phi \subseteq \Sigma^{\square k}$ be a conflict-free set. A *handle* is a word of the form $[u] = [a_1 \odot \ldots \odot a_m]$ where $a_i \in (\Sigma - \{\#\})$. Notice that u is a tagged word not containing $[,], \#$.
Let $[u]$ be a handle, and $s \in \Delta$ be a tag. A *reduction* is a binary relation $\rightsquigarrow_\Phi \subseteq \Sigma^\square \times \Sigma^\square$, defined as:

$$w[u]z \rightsquigarrow_\Phi wsz \text{ if, and only if, } w[u]z \in \text{SLT}(\Phi) \text{ and } wsz \in \text{SLT}(\Phi). \tag{3}$$

A reduction is *leftmost* if no handle occurs in w. Subscript Φ may be dropped from $\rightsquigarrow_\Phi$ when clear from context. The reflexive and transitive closure of $\rightsquigarrow$ is denoted by $\overset{*}{\rightsquigarrow}$.
The *tagged max-language* and the *max-language* defined by Φ are respectively:

$$\bar{L}(\Phi) = \left\{ w \in \Sigma^\square \mid \textcircled{\#}[w]\textcircled{\#} \overset{*}{\rightsquigarrow}_\Phi \textcircled{\#} \odot \textcircled{\#} \right\} \text{ and } L(\Phi) = \sigma\left(\bar{L}(\Phi)\right).$$

It is important to observe that at most one tag s may occur in relation (3) since Φ is conflict-free. Clearly, the max-language does not depend on the order of application of reductions, and we may choose the leftmost order.

Example 4.8. The Dyck language (without ε) over the alphabet $\{a, a', b, b'\}$ is the max-language $L(\Phi)$ defined by the tagged 3-word set: $\Phi = \{\# \odot \#,\ a \odot a',\ b \odot b'\} \cup \{\#[B,\ W]Z,\ B[C,\ W[B,\ W]\# \mid B, C \in \{a, b\},\ W, Z \in \{a', b'\}\}$.
Word $aaa'a'aa'$ is recognized by the following leftmost reduction:

$$\textcircled{\#}[a[a \odot a']a'[a \odot a']\textcircled{\#} \rightsquigarrow \textcircled{\#}[a \odot a'[a \odot a']\textcircled{\#} \rightsquigarrow \textcircled{\#}[a \odot a']\textcircled{\#} \rightsquigarrow \textcircled{\#} \odot \textcircled{\#}.$$

4.2.3. Max-grammar construction

Given a set Φ of tagged k-words, we present a novel[3] construction of an HOP(k) grammar, denoted by $G(\Phi)$, that generates the max-language $L(\Phi)$.

Definition 4.9 (max-grammar construction)**.** Given a set of tagged words $\Phi \subseteq \Sigma^{\square k}$ and the symmetrical FA $A_\Phi = (\Sigma \cup \Delta, Q, \delta, I, T)$ recognizing SLT(Φ), we construct two grammars, respectively called *tagged max-grammar* and just *max-grammar*, denoted by $\overline{G}_\Phi$ and G_Φ. We first construct a temporary grammar $\overline{G}'_\Phi = (V'_N, \Sigma \cup \Delta, \bar{P}', S')$, which may contain useless parts. By reducing the temporary grammar we then obtain the tagged max-grammar $\overline{G}_\Phi = (V_N, \Sigma \cup \Delta, \bar{P}, S)$.
The temporary grammar is constructed as follows.

- The nonterminal alphabet V'_N is a subset of $Q \times Q$ such that

  $(q_1, q_2) \in V'_N$ if $q_1 = (\beta_1, [\gamma_1)$ and $q_2 = (\gamma_2], \alpha_2)$, for some $\beta_1, \gamma_1, \gamma_2, \alpha_2$.

  Notice that each nonterminal is identified by a pair of states such that the look-ahead of q_1 starts with "[" and the look-back of q_2 ends with "]". Equivalently, the same nonterminal is identified by the 4-tuple of $(k-1)$-words: $(\beta_1, [\gamma_1, \gamma_2], \alpha_2)$.

- The axiom set is $S' = I \times T$.

- For each nonterminal $X = (\beta_X, \alpha_X, \beta'_X, \alpha'_X)$, there is a rule $X \to \bar{R}_X \in \bar{P}'$, such that $\bar{R}_X$ is the regular language recognized by the FA $\bar{M}_X$ defined as follows:

  - $\bar{M}_X = (V \cup \Delta, Q_X, \delta_X, \{p_I\}, \{p_T\})$
  - state set: $Q_X = Q$ (useless states will be trimmed)
  - initial state $p_I = (\beta_X, \alpha_X)$; final state $p_T = (\beta'_X, \alpha'_X)$
  - the transition relation δ_X is computed in three steps: (i) add to the relation δ of the symmetric automaton A_Φ a set δ' of edges labeled with nonterminals, (ii) delete a set δ'' of edges labeled with "[" or "]" that are not initial or final, respectively, (iii) delete the set δ''' of edges that enter the initial state or that exit from the final state. Thus $\delta_X = \delta \cup \delta' - \delta'' - \delta'''$ (sets $\delta', \delta'', \delta'''$ are disjoint). The sets are:

[3]Although the notion of max-grammar originates from the theory of OP languages in [6], the approach there used cannot be easily extended to the HOP case; the following construction and proofs are completely new.

$$\delta' = \left\{ (\beta_1, \alpha_3) \overset{(\beta_1,\alpha_1,\beta_2,\alpha_2)}{\longrightarrow} (\beta_3, \alpha_2) \;\middle|\; \begin{array}{l} (\beta_1, \alpha_1, \beta_2, \alpha_2) \in V_N, \\ (\beta_1, \alpha_3) \overset{\odot}{\longrightarrow} (\beta_3, \alpha_2) \in \delta \vee \\ (\beta_1, \alpha_3) = p_I \wedge (\beta_1, \alpha_3) \overset{[}{\longrightarrow} (\beta_3, \alpha_2) \in \delta \vee \\ (\beta_3, \alpha_2) = p_T \wedge (\beta_1, \alpha_3) \overset{]}{\longrightarrow} (\beta_3, \alpha_2) \in \delta \end{array} \right\}$$

$$\delta'' = \left\{ q' \overset{[}{\longrightarrow} q'' \in \delta \;\middle|\; q' \neq p_I \right\} \cup \left\{ q' \overset{]}{\longrightarrow} q'' \in \delta \;\middle|\; q'' \neq p_T \right\}$$

$$\delta''' = \left\{ q' \overset{x \in \Sigma}{\longrightarrow} p_I \in \delta \right\} \cup \left\{ p_T \overset{x \in \Sigma}{\longrightarrow} q' \in \delta \right\}.$$

Notice that each $\bar{M}_X$ may contain useless states, which disappear when the temporary grammar $\overline{G}'_\Phi$ is reduced to obtain the tagged grammar $\overline{G}_\Phi$.
Given the tagged max-grammar $\overline{G}_\Phi = (V_N, \Sigma \cup \Delta, \bar{P}, S)$, the *max-grammar* is defined as:

$$G_\Phi = \left(V_N, \Sigma, \left\{X \to \sigma(\bar{R}_X) \mid X \to \bar{R}_X \in \bar{P}\right\}, S\right).$$

Explanations. In the tagged max-grammar each rule right part is a subgraph starting with an edge labeled "[", ending with a label "]", and containing as labels only terminals, nonterminals and $\odot$ tags; the rule left part is the nonterminal denoted by the pair of initial and final states of the subgraph. Thus, each word in language R_X has the format of a potential handle for parsing.
Intuitively, set δ' adds transitions with nonterminal labels between any two states already linked by a tag-labeled edge, provided such transitions are "compatible" with the nonterminal name, in the sense of having the same look-back and look-ahead.
Set δ'' acts as a filter to remove the edges of δ labeled by tags "[" or "]" that are not initial or final. Acting also as filter, set δ''' removes the edges of δ that reenter the initial state or exit from the final state.
We observe that, although each $\bar{M}_X$ has a unique final state $p_T = (\beta'_X, \alpha'_X)$, this does not reduce generality since such FA is used to specify the right part of a grammar rule; an FA with two or more final states would be represented by as many separate rules.

By construction, for each rule $X \to \bar{M}_X$ of a tagged max-grammar $\overline{G}_\Phi$, the language $\bar{R}_X$ is included into $\bar{R}_X \subseteq (V_N \cup \{[\}) \cdot \Sigma \cdot ((V_N \cup \{\odot\}) \cdot \Sigma)^* \cdot (V_N \cup \{]\})$, implying that grammars $\overline{G}_\Phi$ and G_Φ are in operator form.

For convenience, we also define the *grammar graph*, denoted by $\Gamma(\overline{G}_\Phi)$, associated to the tagged grammar $\overline{G}_\Phi$. The grammar graph includes as subgraph the graph of the symmetrical automaton A_Φ, and in addition includes all the edges labeled with nonterminals, resulting from the above definition of set δ'. The *grammar graph* is just a synthetic representation of all the rules of the max-grammar $\overline{G}_\Phi$ and will help in the proof of the next lemma.

We illustrate the construction with an example.

Example 4.10. We show the construction of the max-grammar for the set $\Phi = \{\#\odot\#,$ $\#[a,\ b]\#,\ b]c,\ c\odot b,\ b\odot b,\ a\odot c,\ a[a\}$ of Example 4.6. Its symmetrical automaton A_Φ is shown in Figure 5 (i). The nonterminal alphabet V'_N of the temporary grammar $\overline{G}'_\Phi$ is included in the Cartesian product $\{(\odot\#, [a), ([a, [a)\} \times \{(b], c\odot), (b], \#\odot)\}$. But the nonterminals $(\odot\#,\ [a,\ b],\ c\odot)$ and $([a,\ [a,\ b],\ \#\odot)$ are useless, more precisely unreachable, because they are neither axioms nor they are transition labels in the grammar graph. Therefore, after reducing the temporary grammar, only two nonterminals are left in the tagged grammar:

$$\text{the axiom } X = (\odot\#,\ [a,\ b],\ \#\odot) \quad \text{and nonterminal } Y = ([a,\ [a,\ b],\ c\odot)$$

which occurs on the edge from $([a, \odot c)$ to $(a\odot, c\odot)$. The tagged max-grammar rules are

$$X \to [a\,(\odot \cup Y)\,c \odot (b\odot)^*\,b]$$
$$Y \to [a\,(\odot \cup Y)\,c \odot (b\odot)^*\,b]$$

and their FAs are shown in Figure 5 (ii) and (iii), respectively.

Now we state and prove that the language defined by a max-grammar coincides with the one defined by means of iterated reductions in Definition 4.7.

Lemma 4.11. *Let $\Phi \subseteq \Sigma^{\square k}$, and let $\overline{G}_\Phi$ and G_Φ be the max-grammars of Definition 4.9. Then the language identities hold: $L(\overline{G}_\Phi) = \overline{L}(\Phi)$ and $L(G_\Phi) = L(\Phi)$. Moreover, grammar G_Φ has the HOP(k) property.*

Proof. Preliminarily, we notice that the symmetrical FA A_Φ and the grammar graph $\Gamma(\overline{G}_\Phi)$ have the same set of nodes (states), and that A_Φ is a sub-graph of the grammar graph, since it only differs by the absence of nonterminally-labeled edges.
We say that two words w and w' are equivalent on a sequence of states $\nu = q_1, q_2, \ldots, q_n$ (or *path equivalent*), written $w \equiv_\nu w'$, if and only if in $\Gamma(\overline{G}_\Phi)$ two paths exist, both with the state sequence ν, such that w and w' are their labels.

Claim $L(\overline{G}_\Phi) = \overline{L}(\Phi)$. We start from a string $w^{(0)} \in \mathrm{SLT}(\Phi)$ and we prove by induction on the reduction steps that, for some $m > 0$ and for some axiom $W \in S$:

$$w^{(0)} \rightsquigarrow_\Phi w^{(1)} \rightsquigarrow_\Phi \ldots \rightsquigarrow_\Phi w^{(m)} = \textcircled{\#} \odot \textcircled{\#} \text{ if and only if}$$
$$\tilde{w}^{(0)} \Longleftarrow_{\overline{G}_\Phi} \tilde{w}^{(1)} \Longleftarrow_{\overline{G}_\Phi} \ldots \Longleftarrow_{\overline{G}_\Phi} \tilde{w}^{(m)} = W, \text{ where}$$
$$\tilde{w}^{(0)} = w^{(0)}, \text{ and } \forall i, \exists \nu_i : \tilde{w}^{(i)} \equiv_{\nu_i} w^{(i)}.$$

Base case: Consider $\tilde{w}^{(0)}$: it is by definition $\tilde{w}^{(0)} = w^{(0)}$, hence $\exists \nu : \tilde{w}^{(0)} \equiv_\nu w^{(0)}$.
Induction case: First, we prove that $w^{(t)} \rightsquigarrow_\Phi w^{(t+1)}$ implies $\tilde{w}^{(t)} \Longleftarrow_{\overline{G}_\Phi} \tilde{w}^{(t+1)}$ with $\tilde{w}^{(t+1)} \equiv_{\nu_{t+1}} w^{(t+1)}$. To perform the reduction, we need a handle, let it be called x, such that $w^{(t)} = uxv \rightsquigarrow w^{(t+1)} = usv$, $s \in \Delta$. By induction hypothesis, we know that $\tilde{w}^{(t)} \equiv_{\nu_t} w^{(t)} = uxv$, therefore $\tilde{w}^{(t)} = \tilde{u}\tilde{x}\tilde{v}$ with $\tilde{u} \equiv_{\nu'_t} u$, $\tilde{x} \equiv_{\nu''_t} x$, and $\tilde{v} \equiv_{\nu'''_t} v$, with $\nu_t = \nu'_t \nu''_t \nu'''_t$. The equivalence $\tilde{x} \equiv_{\nu''_t} x$, with x handle, implies that there is a right part of a rule $X \to M_X \in P$, such that $\tilde{x} \in R_X$. Hence, $\tilde{w}^{(t)} \Longleftarrow_{\overline{G}_\Phi} \tilde{w}^{(t+1)} = \tilde{u}X\tilde{v}$ and $X = (t_{k-1}(\textcircled{\#}u), i_{k-1}(xv\textcircled{\#}), t_{k-1}(\textcircled{\#}ux), i_{k-1}(v\textcircled{\#}))$.
The reduction relation implies that in A_Φ (and therefore also in $\Gamma(\overline{G}_\Phi)$) there is a path with states ν_{t+1} and labels $w^{(t+1)}$. Call ν'_{t+1} the states of the path prefix having u as label, and call ν''_{t+1} those of the path suffix with label v. Let us call q_u the last state of ν'_{t+1} and q_v the first state of ν''_{t+1}.
By construction of $\overline{G}_\Phi$, in $\Gamma(\overline{G}_\Phi)$ there is a transition $q_u \xrightarrow{X} q_v$, while in A_Φ there is a reduction $q_u \xrightarrow{s} q_v$. From this it follows $\tilde{w}^{(t+1)} \equiv_{\nu'_{t+1}\nu''_{t+1}} w^{(t+1)}$.

We now prove that

$$\tilde{w}^{(t)} \Longleftarrow_{\overline{G}_\Phi} \tilde{w}^{(t+1)} \text{ implies } w^{(t)} \rightsquigarrow_\Phi w^{(t+1)}, \text{ with } w^{(t+1)} \equiv_{\nu_{t+1}} \tilde{w}^{(t+1)}.$$

By definition of derivation, it is $\tilde{w}^{(t)} = \tilde{u}\tilde{x}\tilde{v} \Longleftarrow_{\overline{G}_\Phi} \tilde{w}^{(t+1)} = \tilde{u}X\tilde{v}$ for some $X \in V_N$. By induction hypothesis, we know that $\tilde{u}\tilde{x}\tilde{v} = \tilde{w}^{(t)} \equiv_{\nu_t} w^{(t)}$, hence $w^{(t)} = uxv$ with $\tilde{u} \equiv_{\nu'_t} u$, $\tilde{x} \equiv_{\nu''_t} x$, and $\tilde{v} \equiv_{\nu'''_t} v$, with $\nu_t = \nu'_t\nu''_t\nu'''_t$. From this it follows that

$$X = (t_{k-1}(\#u), i_{k-1}(xv\#), t_{k-1}(\#ux), i_{k-1}(v\#)), \text{ and } x \text{ must be an handle.}$$

Therefore, $w^{(t)} = uxv \rightsquigarrow w^{(t+1)} = usv$, $s \in \Delta$, and in A_Φ (and in $\Gamma(\overline{G}_\Phi)$) there is a path with states ν_{t+1} and labels $w^{(t+1)}$: call ν'_{t+1} the states of its prefix with label u, and ν''_{t+1} those of its suffix with label v. Let us call q_u the last state of ν'_{t+1} and q_v the first state of ν''_{t+1}. By construction of $\overline{G}_\Phi$, in $\Gamma(\overline{G}_\Phi)$ there is a transition (q_u, X, q_v), while in A_Φ there is (q_u, s, q_v). Hence $\tilde{w}^{(t+1)} \equiv_{\nu'_{t+1}\nu''_{t+1}} w^{(t+1)}$.

Claim $L(G_\Phi) = L(\Phi)$ and G_Φ is HOP(k). In the previous part of the proof, we showed that there is a bijection between all $\overline{G}_\Phi$'s sentential forms and the reductions defining $L(\Phi)$. Hence, $\varphi_k(\#L(\overline{G}_\Phi)\#) = \Phi$.

By construction, G_Φ and $\overline{G}_\Phi$ share the same structure: the rules of G_Φ are defined on those of $\overline{G}_\Phi$, by deleting tags. It is immediate to see that, if we apply the construction of Definition 3.2 to G_Φ we obtain $\overline{G}_\Phi$. Therefore, $\varphi_k(\#L(\overline{G_\Phi})\#) = \varphi_k(\#L(\overline{G}_\Phi)\#) = \Phi$, which means that G_Φ is HOP(k).

Also, for each derivation $W \overset{+}{\Longrightarrow}_{\overline{G}_\Phi} w$ there is a derivation $W \overset{+}{\Longrightarrow}_{G_\Phi} \sigma(w)$. From the bijection from $\overline{G}_\Phi$'s sentential forms to the reductions defining $L(\Phi)$, it follows that $L(G_\Phi) = L(\Phi)$. □

We are ready to prove that each max-language is indeed maximal.

Theorem 4.12. *Let G be any grammar in the family HOP(Φ) and $L(\Phi)$ be the max-language. Then the language inclusion holds $L(G) \subseteq L(\Phi)$.*

Proof. (hint) Let $G = (V_N, \Sigma, P, S)$ with $\overline{G} = (V_N, \Sigma \cup \Delta, \overline{P}, S)$ the tagged grammar. Let $G_\Phi = (V'_N, \Sigma, P', S')$ and $\overline{G}_\Phi = (V'_N, \Sigma \cup \Delta, \overline{P}', S')$ be, respectively, the max-grammar and the tagged max-grammar.
First, we establish language inclusion for tagged grammars by proving the following:

$$\text{if, for } X \in S, X \overset{+}{\Longrightarrow}_{\overline{G}} w \text{ then, for some } Y' \in S', Y' \overset{+}{\Longrightarrow}_{\overline{G}_\Phi} w. \tag{4}$$

We safely assume that both derivations are leftmost. If $X \overset{+}{\Longrightarrow}_{\overline{G}} uX_1v \Longrightarrow_{\overline{G}} uw_1v = w$ then w_1 is the leftmost handle in w, and by definition of max-grammar, there exists a derivation $uZ'v \Longrightarrow_{\overline{G}} uw_1v$ where nonterminal Z' is identified by the 4-tuple $(t_k(\#u), i_k(w_1v\#), t_k(\#uw_1), i_k(v\#))$.
Therefore, after the reduction (corresponding to the derivation step), the positions of the leftmost handle in uX_1v and in $uZ'v$ coincide, and we omit the simple inductive argument that completes the proof of the inclusion $L(\overline{G}) \subseteq \overline{L}(\Phi)$.

Second, consider grammars G and G_Φ. Clearly, the two derivations (4) for $\overline{G}$ and for $\overline{G}_\Phi$ have the same length and create isomorphic trees, which only differ in the nonterminal names. By applying projection σ to both derivations, the thesis follows. $\square$

As a corollary, for any subset $\Phi' \subset \Phi$ and $G' \in$ HOP(Φ') the inclusions $L(\overline{G}') \subseteq \overline{L}(\Phi)$ and $L(G') \subseteq L(\Phi)$ hold.

We observe that the concept of max-language can be naturally used to define the *complement* of a language. More precisely, given a grammar G which is in family HOP(Φ), for some Φ, the complement of language $L(G)$ is defined as $L(G_\Phi) - L(G)$.

Corollary 4.13. *For every k and for every conflict-free set $\Phi \subset \Sigma^{\square k}$, the language family HOP(Φ) is closed under* complement.

Verification and Model Checking. The next statement is a direct consequence of the decidability of the emptiness problem for CFLs and of the closure of HOP(Φ) languages under set difference.

Corollary 4.14. *For every Φ, the inclusion problem between languages in HOP(Φ) is decidable.*

These properties, possibly paired with automata and logic characterizations such as those for OP [12], make HOP languages suitable for devising automatic verification techniques, most notably model checking.

4.3. Intersection with regular languages

Building on preceding results, the closure property of HOP languages under intersection with regular languages is next established.

Theorem 4.15. *For every k and tagged set $\Phi \in \Sigma^{\square k}$, the language family HOP(Φ) is closed under intersection with regular languages.*

Proof. Consider a language $L = L(G)$ where grammar $G = (V_N, \Sigma, P, S)$ is in HOP(Φ). For each rule $X \to R_X$ of P, we may safely assume that the recognizer M_X of R_X is deterministic. Moreover we assume that M_X has a single final state; if not we can split the rule $X \to R_X$ into as many nonterminals and rules, as there are final states in M_X. Then, let $M_X = (\Sigma \cup V_N, Q_X, \delta_X, q_{0,X}, \{q_{t,X}\})$.

Let $R \subseteq \Sigma^+$ be recognized by the deterministic FA $M = (\Sigma, Q, \delta, p_0, T)$ where the set $T \subseteq Q$ of final states is for now a singleton $T = \{p_t\}$. At the end of the proof we deal with the case of multiple final states.
We construct a grammar $\hat{G} = (\hat{V}_N, \Sigma, \hat{P}, \hat{S})$, then we prove that $L(\hat{G}) = L(G) \cap R$.

The nonterminals of $\hat{G}$ are 3-tuples in $\hat{V}_N \subseteq Q \times V_N \times Q$. The axiom set $\hat{S}$ contains, for all $X \in S$, the nonterminal $\langle p_0, X, p_t\rangle$. The rules of $\hat{G}$ are constructed as follows.

$$\begin{aligned}
&\forall p, p' \in Q,\ \forall X \to M_X \in P:\\
&\text{create all the rules } \langle p', X, p''\rangle \to M_{\langle p',X,p''\rangle} \text{ such that}\\
&M_{\langle p',X,p''\rangle} = \left(\Sigma \cup \hat{V}_N, Q_{\langle p',X,p''\rangle}, \delta_{\langle p',X,p''\rangle}, \langle q_{0,X}, p'\rangle, \langle q_{t,X}, p''\rangle\right)\\
&\text{the states are } Q_{\langle p',X,p''\rangle} \subseteq Q_X \times Q, \text{ and the transitions are:}\\
&\delta_{\langle p',X,p''\rangle}(\langle q, p\rangle, a) = \langle \delta_X(q,a), \delta(p,a)\rangle \text{ with } a \in \Sigma,\\
&\delta_{\langle p',X,p''\rangle}(\langle q, p\rangle, \langle r', Y, r''\rangle) = \langle \delta_X(q,Y), r''\rangle \text{ where } Y \in V_N, r', r'' \in Q, \text{ and } r' = p.
\end{aligned}$$

Clearly, grammar $\hat{G}$ may contain useless nonterminals and some FA may be not trim. Then for each left derivation $X \overset{m}{\Longrightarrow}_G w$ where $w \in L(G) \cap R$ there is an obvious left derivation $(p_0, X, p_t) \overset{m}{\Longrightarrow}_{\hat{G}} w$, and the two derivations have identical structure, i.e., their syntax trees are isomorphic.
It remains to show that $\hat{G}$ is in HOP(Φ). Since for each $w \in L(G) \cap R$ the syntax trees of the derivations for G and $\hat{G}$ are isomorphic, the set of tagged k-words $\Phi(\hat{G})$ is included in $\Phi(G)$.
At last, consider the general case that machine M has multiple final states, therefore R is the disjoint union of the languages recognized by each final state. Since for each final state, the resulting grammar $\hat{G}$, computed above, is in family HOP(Φ), from the closure under union (Theorem 4.3) of language family HOP(Φ), we have that the union of all such grammars is in HOP(Φ). ☐

4.4. Strict infinite hierarchy

We already know that, if a grammar has the HOP(k) property, it also has it for any larger value of the parameter. Next, we show that the HOP(k) language family is a strict infinite hierarchy under set inclusion.

As a witness of the infinite hierarchy, we introduce the following language series:

$$L_{(h)} = \{a^n (ba^h c)^n \mid n \geq 1\} \text{ where } h \geq 1.$$

Lemma 4.16. *For every $h \geq 1$ there exists k such that $L_{(h)} \in$ HOP(k).*

Proof. It is easy to verify that each grammar $G_{(h)}$ with rules $S \to aba^hc \mid aSba^hc$ is in HOP($2h+1$). E.g., for $h = 2$ we obtain the following conflict-free tagged 5-words set: $\Phi = \{a \odot c]\#,\ \# \odot \#[a,\ c]b \odot a,\ \#[a \odot b,\ b \odot a \odot a,\ a[a \odot b,\ a \odot b \odot a,\ a \odot c]b,\ \#[a[a,\ \# \odot \# \odot \#,\ a \odot a \odot c,\ a[a[a,\ c]\# \odot \#\}$. ☐

On the other hand any value k smaller than the value in the preceding proof does not suffice.

Lemma 4.17. $L_{(h)} \notin$ *HOP(k), for $k < 2h+1$.*

Proof. Consider $k < 2h+1$. Let us suppose that there exists a grammar $G \in$ HOP(k) such that $L(G) = L_{(h)}$, let Φ be its associated set of tagged k-words.

First, by looking at all possible factors of size $k' = \lceil k/2 \rceil$, we notice that by Lemma 2.7, the only factor w such that $\sigma(w) = a^{k'}$, must be $w = (as)^{k'}a$, $s \in \{[,], \odot\}$, with a unique tag s. Moreover, the need to count a^n means that s must be $[$.

For simplicity and without loss of generality, we consider now only elements of Φ not containing borders, to analyze the main structure of G's sentences. There are only three possible cases:

1. $(a[)^{k'}a$
2. $(a[)^i as_1bs_2(a[)^j a$, $i + j + 2 = k'$.
3. $(a[)^i as_3cs_4bs_2(a[)^j a$, $i + j + 3 = k'$.

Let us now consider all the possible combinations for $s_1, s_2, s_3, s_4 \in \Delta$. We know that $s_1 = s_2 = s_3 = s_4 = [$ or $s_1 = s_2 = s_3 = s_4 = \odot$, or in general cases where there are no $s_i =]$ are impossible, since the resulting word structure would be right-linear (hence $L(G)$ regular).
The next point analyzes where to put $]$ tags. First, we can exclude $s_1 =]$, since handles would arise such that the factor a^n would be reduced alone, ruling out auto-inclusive rules including parts of the factor $(ba^hc)^n$.
Then, we are going to consider, in turn, $s_2 =]$, $s_3 =]$, and $s_4 =]$, starting from the phrase structure

$$\ldots a[a \ldots a[as_1bs_2a[a \ldots a[as_3cs_4bs_2a[a \ldots a[as_3cs_4bs_2a \ldots$$

Case $s_2 =]$. Since s_1 can be either $\odot$ or $[$, the first handle to be found is, respectively, $[a \odot b]$ or $[b]$. In both cases, after the handle reduction, the resulting structure is:

$$\ldots a[a \ldots a[as_3cs_4b]a[a \ldots a[as_3cs_4b]a \ldots$$

and we consider all the possible $s_3, s_4 \in \Delta$.

When $s_3 = [$, $s_4 = [$, we obtain $\ldots a[a \ldots a[a[c[b]a[a \ldots a[a[c[b]a \ldots$. After reducing all the $[b]$ handles, we need a tag, say s, between c and a. If $s \in \{\odot, [\}$, the resulting structure is right-linear, hence without auto-inclusive rules. If $s =]$, $[c]$ handles arise, but after their reduction we reach again a right-linear structure.

When $s_3 = [$, $s_4 = \odot$, we obtain $\ldots a[a \ldots a[a[c \odot b]a[a \ldots a[a[c \odot b]a \ldots$, and after reducing the $[c \odot b]$ handles, we reach a right-linear structure as well.

When $s_3 = [$, $s_4 =]$, we obtain $\ldots a[a \ldots a[a[c]b]a[a \ldots a[a[c]b]a \ldots$, and, after reducing the $[c]$ handles, $\ldots a[a \ldots a[as_1b]a[a \ldots a[as_1b]a \ldots$. Being $s_1 \in \{\odot, [\}$, the handles are either $[a \odot b]$ or $[b]$, but the resulting structure is in both cases right-linear: $\ldots a[a \ldots a[a \ldots$.

When $s_3 = \odot$, $s_4 = [$, we obtain $\ldots a[a \ldots a[a \odot c[b]a[a \ldots a[a \odot c[b]a \ldots$, which is analogous to the previous $s_3 = [$, $s_4 = [$ case.

When $s_3 = \odot$, $s_4 = \odot$, we obtain $\ldots a[a \ldots a[a \odot c \odot b]a[a \ldots a[a \odot c \odot b]a \ldots$, which, after the $[a \odot c \odot b]$ reductions, is again a right-linear structure.

When $s_3 = \odot$, $s_4 =]$, we obtain $\ldots a[a \ldots a[a \odot c]b]a[a \ldots a[a \odot c]b]a \ldots$, which, after the $[a \odot c]$ reductions, becomes $\ldots a[a \ldots a[as_1b]a[a \ldots a[as_1b]a \ldots$. Then, the handles are either $[a \odot b]$ or $[b]$, but the resulting structure is in both cases right-linear: $\ldots a[a \ldots a[a \ldots$.

When $s_3 =]$, we obtain $\ldots a[a \ldots a[a]cs_4b]a[a \ldots a[a]cs_4b]a \ldots$, and the value of s_4 is not important, since the structure is such that all the a symbols in the a^n prefix can be reduced.
Case $s_3 =]$. We have the structure

$$\ldots a[a \ldots a[as_1bs_2a[a \ldots a[a]cs_4bs_2a[a \ldots a[a]cs_4bs_2a \ldots,$$

where $s_1, s_2 \in \{\odot, [\}$. In both cases, all the right part of the structure, immediately precedingthe first c, can be reduced, losing the a^n prefix.
Case $s_4 =]$. We have the structure

$$\ldots a[a \ldots a[as_1bs_2a[a \ldots a[as_3c]bs_2a[a \ldots a[as_3c]bs_2a \ldots,$$

where $s_1, s_2, s_3 \in \{\odot, [\}$. Like in the previous case, all the right part of the structure, immediately preceding the second b, can be reduced, losing the a^n prefix. □

An immediate consequence of the previous two lemmata is the following.

Theorem 4.18. *For every $k \geq 5$ there exists a language L such that $L \in$ HOP(k) and $L \notin$ HOP$(k-2)$.*

It is interesting to observe that language $L_{(h)}$ is *input-driven* for $h = 0$, and it is OP for $h = 1$; if we take all possible values for h, i.e., $L_{(+)} = \{a^n(ba^+c)^n \mid n \geq 1\}$, the resulting language is not in HOP(k) for any k, but it is deterministic context-free.

4.5. Concatenation and star operations

Although the effect of concatenation on HOP languages is not completely understood, we report a partial negative result and we discuss an example suggestive of possible developments. It is known that the OP language family is closed under concatenation [10] (but it is not known whether the same property holds also for HOP(3)). However the proof in [10] involves transformations of OP grammars which are not easily extended to HOP(k) for $k > 3$.

Moving to larger k values, we exhibit a language in HOP(7) such that its self-concatenation is not in HOP(7) but it is in HOP for a much larger value. Such example raises the open question whether the concatenation of any two HOP(k) languages in the same HOP(Φ) family may always belong to a HOP(k') family, for some $k' > k$.

Since the same language is such that its concatenation closure, though not in HOP(7), is in HOP for a larger value, the same question arises for the star operation.

Example 4.19. The language $L = \{a^n(baab)^n \mid n \geq 1\}$ is generated by the following HOP(7) grammar G: $S \to aSbaab \mid abaab$. It can be proved, by an analysis similar to the proof of the family hierarchy Lemma 4.17, that the concatenation language $L \cdot L$ is not in HOP(7). On the other hand the grammar G_2

$$S_2 \to aSbaabS \mid abaabS, \ S \to aSbaab \mid abaab$$

generates $L \cdot L$ and has the HOP property for $k = 17$ but not for $k = 15$.

4.6. Further properties of max-languages

For completeness, we briefly report from [1] certain results on the subfamily of max-languages, but their systematic study, which would extend to HOP languages the algebraic and lattice-theoretical properties of OP max-languages [6], is out of scope for this paper.

Max-languages versus regular languages. A natural question is whether every regular language can be generated by a max-grammar. Not surprisingly, every *strictly locally testable* k-SLT language defined by a k-word set F is also the max-language defined by a tagged k-word set Φ, which can be easily derived from F. It follows that the family of max-languages includes the SLT family. A witness that such inclusion is strict is the non-SLT language $L = a^*ba^* \cup a^+$, which is the max-language defined by the set $\Phi = \{\#[b, a]b, \# \odot \#, \#[a, b]\#, a \odot a, b[a, a]\#\}$.

On the other hand, there exist very simple regular languages which are not max-languages. An example is $R = (aa)^+$. Assume by contradiction that R is defined by a tagged k-word set Φ. By Lemma 2.7, any conflict-free set Φ defining R may only use one tag, but it is easy to prove that all choices of the tag lead to contradiction.

Since there are obvious examples of non-regular max-languages, e.g., the Dyck languages of Example 4.8, we conclude that the family of max-languages and of regular languages are incomparable.

Hierarchy of max-languages. The max-languages can be classified into families, each family corresponding to all (conflict-free) sets of tagged k-words. Such families can be totally ordered by the value of k and the result is a strict infinite hierarchy. We illustrate by means of the following family of regular languages: $L_{(h)} = (a^h b)^+$, $h \geq 1$. It can be verified that language $L_{(h)}$ is a max-language with parameter $k = 2h + 1$, e.g.,

$$L_{(2)} = L\left(\left\{\begin{array}{c} \# \odot \#[a,\ b]\# \odot \#,\ b]a \odot a,\ a \odot b]a, \\ a \odot a \odot b,\ \#[a \odot a,\ a \odot b]\#,\ \# \odot \# \odot \# \end{array}\right\}\right)$$

On the other hand, it is proved in [1] that $L_{(h)}$ cannot be defined as max-language if $k < 2h + 1$.

Non-closure of max-languages. The max-language family is not closed under any of the basic language operations, as proved by witnesses:

Intersection: the intersection of the following max-languages is not context-free.

$\{a^n b^n c^* \mid n > 0\} = L(\Phi')$,
with $\Phi' = \{b[c,\ c]\#,\ a \odot b,\ c \odot c,\ \# \odot \#,\ \#[a,\ b]\#,\ b]b,\ a[a\ \}$
$\{a^* b^n c^n \mid n > 0\} = L(\Phi'')$,
with $\Phi'' = \{\#[b,\ a]b,\ c]\#,\ c]c,\ \# \odot \#,\ b[b,\ \#[a,\ a \odot a,\ b \odot c\}$.

Notice that the union of Φ' and Φ'' has conflicts.

Set difference: $(a^*ba^* \cup a^+) - a^+$. The first language is a max-language in HOP(3) and requires the 3-tagged words $\textcircled{\#}[a$ and $a]\textcircled{\#}$ (because words may begin and end with a). Since unlimited runs of a are possible, Lemma 2.7 imposes the same tag between any pair of a's, hence the resulting max-language necessarily contains a^+, a contradiction.

Concatenation: the witness is the concatenation of language

$$a^*b = L\left(\{\#[b,\ a \odot b,\ \# \odot \#,\ \#[a,\ b]\#,\ a \odot a\}\right)$$

and language a^+. The proof is similar to the one for set difference.

Intersection with regular sets: the witness is $\{a, b\}^+ \cap (aa)^+$, see above the reasoning for proving that $(aa)^+$ is not a max-language.

5. Conclusions

5.1. Related work

Very few attempts have been made in the past to improve the applicability of the *operator precedence* (OP) model, especially in the direction of definition and parsing of programming languages. Such paucity of efforts is easily understood if you consider that, shortly after the invention of OP grammars, two major advances on formal grammars and deterministic parsing took place, namely the LR(k) and LL(k) methods that we do not have to discuss; their widespread adoption shifted attention away from OPL for some years, until the different motivations mentioned in the Introduction, revived research. In the time span between [2] and such advances, two proposals are worth mentioning, the *simple-precedence grammars* of Wirth and Weber [27] and the *bounded-context* grammars [14] by Floyd himself. Each of them uses a different definition of the relations that, like OP relations, permit to find a reducible handle; in particular, such relations involve also nonterminal symbols. In both cases the operator normal form is not required, and neither model is a formal mathematical extension of OP grammars. Both language families strictly include the OPL family, but they do not have any of the characteristic closure properties of the latter. Some comparisons of generative capacity are in Fischer [19], but, what is more important, neither simple-precedence or bounded-context grammars (unless suitably restricted to avoid ambiguity), nor the LL(k) and LR(k) grammars, are locally parsable.

Very recently, we defined a class of deterministic grammars and automata, called *locally chain parsable* [28] (LCP) which is locally parsable and preserves the closure properties of OPL, except for concatenation and Kleene star. Moreover, any locally chain-parsable grammar is in operator form but it may have conflictual OP relations; notice that the Extended CF form has not been considered for LCP. The LCP family is not comparable with the HOP one; the LCP language $\{a^n b^n \mid n \geq 1\} \cup \{b^n a^n \mid n \geq 1\}$ is obviously not in HOP, on the other hand, the language L_{loop} of Theorem 3.6 is in HOP(3) but the same arguments that show it not to be OP can be easily applied to LCP. A limitation of the LCP approach is that the theory has been formulated only for contexts consisting of one character, and it would be complicated to deal with larger contexts, as we do here in the HOP(k) approach. A closer comparison of the parsing algorithms and automata for the two families remains to be done.

It is customary in deterministic parsing studies to consider grammar families indexed by an integer expressing the length of the look-ahead, i.e., the right context, needed by the parser to find the handle: e.g., LR(k) and LL(k) are such families, and, in a different sense, also the HOP(k) and the bounded-context grammars [14] already mentioned, which use also a look-back. To be precise, LR(k) and LL(k) parsers are driven by a finite-state machine and supplement the state information by looking at the next k look-ahead characters. On the other hand, in our HOP(k) model no finite-state information is carried from left to right (since it would be incompatible with local parsability) and the parser uses a sliding window of length $(k-1)/2$ characters.

Since HOP extends the OP language family, which in turn includes the input-driven (or visibly pushdown) language [10] family, it is interesting to compare the HOP family with the recent extension of VP languages, recognized by *tinput-driven pushdown automata* (TDPDA) [29], which enjoy similar closure properties. The families HOP and TDPDA are incomparable. On one hand, the language $\{a^n b a^n \mid n \geq 1\}$ is in TDPDA $-$ HOP. On the other hand, TDPDA automata only recognize real-time languages, and thus fail on the non real-time language $\{a^m b^n c^n d^m \mid n, m \geq 1\} \cup \{a^m b^+ e d^m \mid m \geq 1\}$ which is in HOP(3). Such ability to cope with non-real-time languages comes from the possibility to locally re-calculate syntax tags between terminals after each reduction, a flexibility not available for tinput-driven automata. Moreover, the tinput-driven parser is not suitable for local parsing, because it must operate from left to right, starting from the first input character.

We recall that OP grammars have been applied in early grammar inference studies [4, 5], which exploited the lattice-theoretical properties of the subfamily of so-called *free* OP languages. We did not introduce in this article the lattices associated to HOP(k) families, but it suffices to say that max-grammars and max-languages (Definition 4.9) are the top elements of such lattices. We mention two language classes loosely related to the HOP(k) hierarchy, which have been proposed by recent grammar inference research, which strives to discover expressive grammar types having good learnability properties. Within the so-called distributional approach, several authors have introduced various grammar types based on a common idea: that the syntax class of a word v is determined by the left and right contexts of occurrence, the context lengths being finite integers k and ℓ. Two examples are: the (k, ℓ) *substitutable* CF languages [30] characterized by the implication $x_1 v y_1 u z_1$, $x_1 v y_2 u z_1$, $x_2 v y_1 u z_2 \in L$ implies $x_2 v y_2 u z_2 \in L$ where $|v| = k$ and $|u| = \ell$; and the related hierarchies of languages studied in [31]. A closer comparison of HOP and language classes motivated by grammar inference would be interesting.

5.2. Future developments

Since HOP is a new language model not all of its properties have been examined. In particular, it remains to be seen whether other known properties of OP languages (such as the closure under concatenation and star) continue to hold for HOP.

Another peculiar property is the invariance of the OP family with respect to the *context-free non-counting property* [21, 22]. This means that an OP grammar has such property if, and only if, all equivalent OP grammars have the same property.

We finish by discussing the potential for applications. First, the enhanced generative capacity of higher degree HOP grammars together with the ease of extended context-free grammar rules may in principle simplify the task of writing syntactic definitions, with respect to OP grammars. But, of course, this statement needs to be supported by future practical experiments.

Then HOP(k) parsers should be experimented to measure how efficient they are, both for serial and parallel implementations. In principle the tool PAPAGENO for generating efficient parallel OP parsers [11] can be naturally extended to HOP(k) grammars, but some performance degradation has to be expected for larger values of k.

In a different area, it would be interesting to consider HOP languages for model-checking. For that, the model has to be extended to ω-languages and logically characterized, as recently done for OP languages in [12].

Last, for grammar inference: we observe that it would be possible to define a partial order based on language inclusion, within each subfamily of HOP(k) closed under Boolean operation, i.e., containing structurally compatible languages. Such a partially ordered set of grammars and languages, having the max-grammar as top element, is already known for the OP case, and its lattice-theoretical properties have been exploited for inferring grammars using just positive information sequences [5]. The availability of the k-ordered hierarchy may then enrich the learnable grammar space.

Acknowledgment. We thank Dino Mandrioli for his comments and suggestions.

[1] S. Crespi Reghizzi, M. Pradella, Higher-order operator precedence languages, in: E. Csuhaj-Varjú, P. Dömösi, G. Vaszil (Eds.), Proc. AFL 2017, Vol. 252 of EPTCS, 2017, pp. 86–100.
URL `http://arxiv.org/abs/1708.06226`

[2] R. W. Floyd, Syntactic Analysis and Operator Precedence, JACM 10 (3) (1963) 316–333.

[3] D. Grune, C. J. Jacobs, Parsing techniques: a practical guide, Springer, New York, 2008.

[4] S. Crespi Reghizzi, M. A. Melkanoff, L. Lichten, The Use of Grammatical Inference for Designing Programming Languages, Commun. ACM 16 (2) (1973) 83–90.

[5] D. Angluin, C. H. Smith, Inductive inference: Theory and methods, ACM Comput. Surv. 15 (3) (1983) 237–269. `doi:10.1145/356914.356918`.

[6] S. Crespi Reghizzi, D. Mandrioli, D. F. Martin, Algebraic Properties of Operator Precedence Languages, Information and Control 37 (2) (1978) 115–133.

[7] K. Mehlhorn, Pebbling mountain ranges and its application of DCFL-recognition, in: Automata, languages and programming (ICALP-80), Vol. 85 of LNCS, 1980, pp. 422–435.

[8] B. von Braunmühl, R. Verbeek, Input-driven languages are recognized in log n space, in: Proc. of the Symp. on Fundamentals of Computation Theory, LNCS 158, Springer, 1983, pp. 40–51.

[9] R. Alur, P. Madhusudan, Adding nesting structure to words, JACM 56 (3).

[10] S. Crespi Reghizzi, D. Mandrioli, Operator Precedence and the Visibly Pushdown Property, JCSS 78 (6) (2012) 1837–1867.

[11] A. Barenghi, S. Crespi Reghizzi, D. Mandrioli, F. Panella, M. Pradella, Parallel parsing made practical, Sci. Comput. Program. 112 (2015) 195–226. doi:10.1016/j.scico.2015.09.002.

[12] V. Lonati, D. Mandrioli, F. Panella, M. Pradella, Operator precedence languages: Their automata-theoretic and logic characterization, SIAM J. Comput. 44 (4) (2015) 1026–1088. doi:10.1137/140978818.

[13] D. Mandrioli, M. Pradella, Generalizing input-driven languages: Theoretical and practical benefits, Computer Science Review 27 (2018) 61–87. doi:10.1016/j.cosrev.2017.12.001.

[14] R. W. Floyd, Bounded context syntactic analysis, Commun. ACM 7 (2) (1964) 62–67.

[15] R. McNaughton, S. Papert, Counter-free Automata, MIT Press, Cambridge, USA, 1971.

[16] M. A. Harrison, Introduction to Formal Language Theory, Addison Wesley, 1978.

[17] S. Crespi Reghizzi, L. Breveglieri, A. Morzenti, Formal Languages and Compilation, Second Edition, Texts in Computer Science, Springer, 2013.

[18] D. Knuth, A characterization of parenthesis languages, Information and Control 11 (1967) 269–289.

[19] M. J. Fischer, Some properties of precedence languages, in: STOC '69: Proc. first annual ACM Symp. on Theory of Computing, ACM, New York, NY, USA, 1969, pp. 181–190. doi:http://doi.acm.org/10.1145/800169.805432.

[20] V. Lonati, D. Mandrioli, F. Panella, M. Pradella, Operator precedence languages: Their automata-theoretic and logic characterization, SIAM J. Comput. 44 (4) (2015) 1026–1088. doi:10.1137/140978818.
URL https://doi.org/10.1137/140978818

[21] S. Crespi Reghizzi, G. Guida, D. Mandrioli, Noncounting Context-Free Languages, JACM 25 (1978) 571–580.

[22] S. Crespi Reghizzi, G. Guida, D. Mandrioli, Operator Precedence Grammars and the Noncounting Property, SIAM J. Computing 10 (1981) 174—191.

[23] P. Caron, Families of locally testable languages, Theor. Comput. Sci. 242 (1-2) (2000) 361–376. doi:10.1016/S0304-3975(98)00332-6.

[24] R. McNaughton, Parenthesis Grammars, JACM 14 (3) (1967) 490–500.

[25] Y. Bar-Hillel, M. A. Perles, E. Shamir, On formal properties of simple phrase structure grammars, Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung 14 (1961) 143–172.

[26] A. Borsotti, L. Breveglieri, S. Crespi Reghizzi, A. Morzenti, Fast deterministic parsers for transition networks, Acta Informaticadoi:10.1007/s00236-017-0308-3.
URL https://doi.org/10.1007/s00236-017-0308-3

[27] N. Wirth, H. Weber, EULER: a generalization of ALGOL and it formal definition: Part 1, Communications of the ACM 9 (1) (1966) 13–25.

[28] S. Crespi Reghizzi, V. Lonati, D. Mandrioli, M. Pradella, Toward a theory of input-driven locally parsable languages, Theor. Comput. Sci. 658 (2017) 105–121. doi:10.1016/j.tcs.2016.05.003.

[29] M. Kutrib, A. Malcher, M. Wendlandt, Tinput-driven pushdown, counter, and stack automata, Fundam. Inform. 155 (1-2) (2017) 59–88. doi:10.3233/FI-2017-1576.

[30] R. Yoshinaka, Identification in the limit of k, l-substitutable context-free languages, in: A. Clark, F. Coste, L. Miclet (Eds.), Grammatical Inference: Algorithms and Applications, 9th International Colloquium, ICGI 2008, Vol. 5278 of LNCS, Springer, 2008, pp. 266–279. doi:10.1007/978-3-540-88009-7_21.

[31] F. M. Luque, G. G. I. López, PAC-learning unambiguous *k*, *l*-NTS $^{<=}$ languages, in: Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings, 2010, pp. 122–134.

Appendix 1: a Pascal-like toy language

Quite often the official grammars of technical languages, also when already in operator form, have OP conflicts. In most cases it is possible to eliminate conflicts by manual grammar transformations, which however may damage grammar terseness and adequacy of syntax to semantic. In such cases, if the original grammar meets the HOP condition, it can be used without change. We show an example of such situation for a toy language including some typical constructs of a programming language.

$$
\begin{array}{lll}
\langle\text{program}\rangle & \to & \langle\text{type declaration}\rangle;\langle\text{program}\rangle \mid \langle\text{statement}\rangle;\langle\text{program}\rangle \mid \\
 & & \langle\text{labeled statement}\rangle;\langle\text{program}\rangle \mid \\
 & & \langle\text{type declaration}\rangle; \mid \langle\text{statement}\rangle; \mid \langle\text{labeled statement}\rangle; \\
\langle\text{type declaration}\rangle & \to & id : id \\
\langle\text{labeled statement}\rangle & \to & int : \langle\text{statement}\rangle \\
\langle\text{statement}\rangle & \to & id := \langle\text{expression}\rangle \mid if\ \langle\text{expression}\rangle\ then\ \langle\text{statement}\rangle \mid \\
 & & while\ \langle\text{expression}\rangle\ do\ \langle\text{statement}\rangle \\
\langle\text{expression}\rangle & \to & id + id \mid id \mid (\langle\text{expression}\rangle)
\end{array}
$$

This grammar is not HOP(3) because of this conflict: $:\ \odot\ id,\ :\ [\ id$. It would be possible but annoying to transform the grammar to an equivalent (conflict-free) OP grammar.

On the other hand, the original grammar meets the HOP(5), as shown by the tagged 5-word set:

$$
\left\{
\begin{array}{l}
while \odot do \odot if,\ :=]\ end\ ]\ \#,\ \# \odot \# [\ id,\ \# [\ end\ [\ if,\ if\ [\ id\ \odot +,\ :\ [\ id \odot :=, \\
id \odot :=]\ end,\ int \odot :\ [\ if,\ :]\ end \odot end,\ \# [\ if \odot then,\ \# [\ end\ [\ int,\ while \odot do \odot while, \\
)\ ]\ then\ ]\ end,\ :\ [\ while \odot do,\ id\ \odot + \odot id,\ (\odot)\ ]\ do,\ id\ ]\ do\ ]\ end,\ (\odot)\ ]\ end, \\
end \odot end \odot end,\ end\ ]\ \# \odot \#,\ while\ [\ (\odot),\ if \odot then\ ]\ end,\ int \odot :\ [\ id,\ while \odot do \odot id, \\
end \odot end\ ]\ \#,\ + \odot id\ ]\ then,\ \# \odot \# [\ end,\ :=\ [\ id\ ]\ end,\ do\ ]\ end\ ]\ \#,\ \# [\ if\ [\ id, \\
\# \odot \# [\ while,\ :=\ [\ id\ \odot +,\ end\ [\ id \odot :=,\ \# \odot \# [\ if,\ id \odot : \odot id,\ do \odot while \odot do, \\
then \odot if \odot then,\ + \odot id\ ]\ end,\ id\ ]\ end \odot end,\ \# [\ end\ [\ while,\ :=]\ end \odot end, \\
then \odot while \odot do,\ int \odot :]\ end,\ \# [\ end\ ]\ \#,\ : \odot id\ ]\ end,\ do\ ]\ end \odot end,\ then\ ]\ end \odot end, \\
int \odot :\ [\ while,\ :=\ [\ (\odot),\ id \odot :=\ [\ id,\ )\ ]\ do\ ]\ end,\ (\odot)\ ]\ then,\ while\ [\ id\ ]\ do, \\
\# [\ while\ [\ id,\ end\ [\ if \odot then,\ then \odot id \odot :=,\ :\ [\ if \odot then,\ end\ [\ id \odot :,\ :]\ end\ ]\ \#, \\
\# [\ id \odot :=,\ \# [\ while\ [\ (,\ \# [\ while \odot do,\ while\ [\ id\ \odot +,\ \# [\ id \odot :,\ end\ [\ int \odot :, \\
+ \odot id\ ]\ do,\ if \odot then \odot while,\ if\ [\ (\odot),\ \# \odot \# [\ int,\ do \odot if \odot then,\ if \odot then \odot id, \\
\# [\ int \odot :,\ end\ [\ while \odot do,\ id \odot :=\ [\ (,\ do \odot id \odot :=,\ \# [\ end \odot end,\ if \odot then \odot if, \\
id\ ]\ then\ ]\ end,\ \# \odot \# \odot \#,\ \# [\ if\ [\ (,\ if\ [\ id\ ]\ then,\ while \odot do\ ]\ end,\ \# [\ end\ [\ id, \\
id\ ]\ end\ ]\ \#,\ )\ ]\ end\ ]\ \#,\ then\ ]\ end\ ]\ \#
\end{array}
\right\}
$$

Notice that just a few positions require value $k = 5$ to remove conflict. Therefore, a clever practical parser should use the locally minimal value of k which suffices to make handle positioning unique.

Appendix 2: Proof of Lemma 4.2

Theorem 5 of Knuth [18] addressed non-extended CF grammars, here we extend its proof to the ECF case, mostly adhering for convenience to his notation. Let $G_{(\,),1} = (V_{N_1}, \Sigma \cup \{(,)\}, P_1, S_1)$ and $G_{(\,),2} = (V_{N_2}, \Sigma \cup \{(,)\}, P_2, S_2)$ be ECF parenthesis grammars. We assume that V_{N_1} and V_{N_2} are disjoint sets. Let V be Σ plus the set of all pairs $[a, \mathcal{B}]$ where $A \in V_{N_1}$ and $\mathcal{B} \subseteq V_{N_2}$. For any pair of words θ_1, θ_2 over

$(\Sigma \cup V_{N_1} \cup V_{N_2} \cup V)$, let $\theta_1 \sim \theta_2$ mean θ_1 and θ_2 are of the same length and agree at all terminals, i.e., $\theta_1 = x_1 \dots x_n$, $\theta_2 = y_1 \dots y_n$ where $x_i = y_i$ whenever either x_i or y_i is in Σ.
Now define the set $\alpha \dot{-} \beta$ for words $\alpha \in V^*$, $\beta \in V_{N_2}^*$ as follows when $\alpha = x_1 \dots x_n$ and $\beta = y_1 \dots y_m$:

$$\alpha \dot{-} \beta = \begin{cases} \{x_1 \dots x_{k-1}\ [A, \mathcal{B} \cup \{y_k\}]\ x_{k+1} \dots x_n \mid 1 \le k \le n \wedge x_k = [A, \mathcal{B}]\}, & \text{if } \alpha \sim \beta; \\ \{\alpha\}, \text{ if } \alpha \nsim \beta. \end{cases} \tag{5}$$

This "difference" operation has the associative property and may be extended to sets as follows:

$$\begin{array}{l} \{\alpha_1, \dots, \alpha_m\} \dot{-} \beta = \bigcup_{1 \le j \le m} (\alpha_m \dot{-} \beta); \\ \{\alpha_1, \dots, \alpha_m\} \dot{-} \emptyset = \{\alpha_1, \dots, \alpha_m\}; \\ \{\alpha_1, \dots, \alpha_m\} \dot{-} \{\beta, \dots, \beta_n\} = \left(\{\alpha_1, \dots, \alpha_m\} \dot{-} \{\beta, \dots, \beta_{n-1}\}\right) \dot{-} \beta_n,\ n \ge 1. \end{array} \tag{6}$$

We construct grammar $G_{(\,)} = (\Sigma, V, P, S)$. Let the homomorphism $\tau : \Sigma \cup V_{N_1} \to V$ be :

$$\tau(a) = a, \text{ for } a \in \Sigma, \text{ and } \tau(A) = [A, \emptyset] \text{ for } A \in V_{N_1}$$

Then $S = \tau(S_1) \dot{-} S_2$ and

$$\begin{array}{l} P = \left\{[A, \mathcal{B}] \to R \mid R = \tau\left(\rho(A)\right) \dot{-} \rho(\mathcal{B})\right\} \text{ where} \\ \rho(A) = R_A \text{ such that } A \to R_A \in P_1 \\ \rho(\mathcal{B}) = \bigcup_{B \in \mathcal{B}} \left(R_B \text{ such that } B \to R_B \in P_2\right) \end{array} \tag{7}$$

Notice that the definition of ρ has been adjusted from the one in [18] to account for the fact that the right part of a rule $A \to \dots$ is a language R_A. This is the only change with respect to [18]. We show that $R = \tau\left(\rho(A)\right) \dot{-} \rho(\mathcal{B})$ is a regular language over $\Sigma \cup V$, with all words starting with "(" and ending with ")". Clearly $\rho(A)$ and $\rho(\mathcal{B})$ are parenthesized regular languages over $\Sigma \cup V$, and the "difference" operation is defined on sets of words. It remains to prove that the result of such operation is a parenthesized regular language over $\Sigma \cup V$. First, it is obvious from Eq. (5) that $\alpha \dot{-} \beta$ is parenthesized. Consider now the case $\alpha \sim \beta$ in Eq. (5), the other case being obvious. Each word in $\alpha \dot{-} \beta$ may differ from word α only in the k-th position, where symbol $x_k \in V$ is replaced by the symbol $[A, \mathcal{B} \cup \{y_k\}]$, here denoted z_k, which thus only depends on the k-th letter of words α and β. We define a nondeterministic finite transducer with two input tapes, containing $x_1 \dots x_k$ and $y_1 \dots y_k$, and an output tape. The machine can be in two states: the *todo* state and the *done* state, with the following transitions (the second component being the output):

$$\begin{array}{l} \delta(\text{todo}, x_j, y_j) = \{(\text{todo}, x_j), (\text{done}, z_j)\} \\ \delta(\text{done}, x_j, y_j) = \{(\text{done}, x_j)\} \end{array}$$

The initial state is *todo* and the final state is *done*. Clearly the translation is the set $(x_1 \dots x_k) \dot{-} (y_1 \dots y_k)$ and is a regular language. Therefore, we may assume that, when the domain of the "difference" is extended to a pair of sets (Eq. (6)), both sets are regular languages, and the application of a regularity preserving operation to each pair of words produces a regular set. It follows that the rules created at lines (7) are in ECF form.

Grammar $G_{(\,)}$ may contain useless nonterminals and rules, which are removed as usual. To show that $L(G_{(\,)}) = L(G_{(\,),1}) - L(G_{(\,),2})$ it suffices to reproduce the proof in [18], p. 285-6, with the understanding that all derivations are taken now for ECF grammars. $\square$