Journal of Computer and System Sciences 78 (2012) 1837–1867

Contents lists available at SciVerse ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Operator precedence and the visibly pushdown property ☆,☆☆

Stefano Crespi Reghizzi *, Dino Mandrioli

DEI – Politecnico di Milano, Piazza Leonardo da Vinci, 32, I-20133 Milano, Italy

ARTICLE INFO

Article history:
Received 14 July 2010
Received in revised form 26 January 2011
Accepted 14 November 2011
Available online 24 December 2011

Keywords:
Pushdown automata
Closure under language operations
Operator precedence grammars
Parenthesis languages
XML languages
Balanced languages
Height deterministic languages

ABSTRACT

Floyd's operator precedence grammars and languages (FG, FL) are a classical subclass of deterministic context-free (DCF) grammars and languages. We prove that several recently introduced language families motivated by the needs of model checking and of specifying XML-like languages are proper subsets of FL. The main cases considered include visibly pushdown languages (VPL) and balanced languages (BALAN), which are characterized by restricted precedence relations. FL have all the closure properties available for regular languages and generally viewed as necessary for application to model checking: reversal, prefixing and suffixing, concatenation, Kleene star, and boolean operations. All but the last results are new, and some require complex proofs, due to the necessary changes of syntax structure. Thus FL are the largest known subfamily of DCF having the same closure properties as VPL. FG, unlike VPL grammars, which are intended for abstract syntax modelling, are structurally adequate to specify real programming languages.

1. Introduction

In recent years a considerable flurry of research activity on formal languages has taken place, in connection with model checking of infinite-state systems and with the need to specify semi-structured data such as XML documents. The best known example are the *visibly pushdown automata and languages* (VPA, VPL) of Alur and Madhusudan [2,3], a subclass of real-time deterministic context-free (CF, DCF) languages, which has similar closure properties as the regular languages, namely closure under concatenation, Kleene star, and boolean operations. VPL constrain the terminal alphabet to be partitioned into three sets, corresponding to the symbols that open a parenthesis, close a parenthesis, and may occur inside parentheses; thus the syntactic role of a symbol is immutable in any part of a sentence.

We show that the same desirable properties of VPL are present in a classical, comfortable language family, Floyd's[1] operator precedence grammars [4] (FG) and languages (FL). Before we comment on this, it is worth while spending some words on the history of earlier efforts in similar directions.

At the root of this line of research, we find McNaughton's parenthesis grammars [5]. By considering instead of strings the stencil (or skeletal) trees encoded by parenthesized strings, some basic properties of regular languages that do not hold for CF languages are still valid: uniqueness of the minimal grammar, and boolean closure within the class of languages having the same rule stencils. Further mathematical developments of those ideas have been pursued in the setting of tree automata [6].

☆ Work partially supported by PRIN 2007TJNZRE-002, CNR-IEIIT, and Research Google.
☆☆ Parts of this work have been presented at WORDS 2009 (Salerno, Italy) and LATA 2010 (Crespi-Reghizzi and Mandrioli, 2010 [1]) conferences.
* Corresponding author.
E-mail addresses: stefano.crespireghizzi@polimi.it (S. Crespi Reghizzi), dino.mandrioli@polimi.it (D. Mandrioli).
[1] We propose to name them Floyd grammars to honor the memory of Robert Floyd and also to avoid confusion with other similarly named but quite different types of precedence grammars.

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2011.12.006

Shortly after, in connection with studies on grammar inference [7], we proved similar closure properties for Floyd's languages [8], which at that time were a leading grammar type for syntax-directed compilation, and are still used (see for instance [9] for a recent practical account). We also extended the notion of non-counting (or aperiodic) regular language of McNaughton and Papert [10] to the parentheses languages and to FL [11]. Surprisingly, however, the basic question of the closure or not of FL under concatenation and Kleene star has never been addressed until very recently [1].

Decades after [5], the widespread adoption of mark-up languages in semi-structured web documents, motivated research on the generalization of parenthesis languages, such as the *balanced grammars* and *languages* (BALAN) [12], which have been later shown to be contained within the VPL.

In connection with model checking and going beyond finite-state representations, other language families have been introduced such as [13], and, of course, VPL. Other related or derived examples, are the *synchronized languages* [14,15] and the *height-deterministic languages* [16]; some of them will be considered in this paper. It is fair to recognize that earlier presentations of similar ideas occurred in the context of parsing complexity studies, such as [17,18].

To sum up, VPL scored as the most comprehensive and robust language family of the series, yet they lack the expressiveness needed for defining practical languages: for instance, they cannot specify that in arithmetic expressions, multiplication takes precedence over addition.

We have found that this and other limitations, caused by the rigid 3-partition of the alphabet and by the realtime recognition constraint, can be dispensed with, yet preserving the nice mathematical properties. This claim is substantiated in this paper by the FL family, which from near oblivion returns to the forefront of technology motivated research on grammars.

An FG has an associated set of precedence relations, and we prove that VPL and BALAN languages are precisely characterized by simple structural restrictions on the precedence relations.

Concerning mathematical properties, adding to the already known boolean closure, we prove FL closure under reversal, suffix/prefix extraction, concatenation, and Kleene star. The last two properties are not obvious and require complex exhaustive arguments, in contrast with the fact that, for the main language families, the two properties are either trivially present or trivially absent: examples of the former case are CF and VPL, while DCF is an example of the latter. The complexity comes from the fact that, although the syntax tree is solely determined by the given precedence relations of the concatenated languages, the syntax tree of $x \cdot y$ can be sharply different from the adjoining of the trees of x and y. The difficulty increases for Kleene star, because the syntax tree of, say, $x \cdot x \cdot x$ cannot be obtained by combining the trees of either $x \cdot x$ and x, or x and $x \cdot x$, but it may have an entirely different structure. Such closure properties permit to extend the rules of Floyd grammars with regular expressions in their right-hand sides, yet preserving pushdown determinism. Thus, to the best of our knowledge, FL are the largest subclass of DCF that enjoy closure properties with respect to all basic algebraic operations.

A different argument in favor of FL is that, unlike other deterministic languages such as LR(k) and LL(k), parsing does not need to scan the text in a fixed direction, say, from left-to-right; this freedom is particularly relevant for developing parallel parsing algorithms suitable to current multiprocessor computers.

To sum-up, we believe that FL, far from being a relic, are an excellent candidate for the never ending search for trade-offs between formalism expressiveness and automatic analyzability.

The paper proceeds as follows: Section 2 lists the essential definitions of FG, and summarizes their basic properties. Section 3 compares FG generative power with VPL and with other related recent families (balanced, height-deterministic). Section 4 proves the closure of FL under all relevant operations and compares FL properties with those exhibited by the other families considered; it also extends FG with regular expressions. Section 5 concludes and suggests topics for further investigations. Appendix A lists all the acronyms used in the paper. To help the reader focusing on the essential contributions, in the body of the paper we rely on intuitive arguments and examples to support our statements, and we postpone to Appendix B more technical material (details of constructions and proofs).

2. Basic definitions of Floyd's grammars

For terms not defined here, we refer to any classical textbook on formal languages (e.g., [19] or [20]). The empty string is denoted ε, the terminal alphabet is Σ. For a string x and a letter a, $|x|_a$ denotes the number of occurrences of letter a in x, and, for a set $\Delta \subseteq \Sigma$, $|x|_\Delta$ is the corresponding extension. $first(x)$ and $last(x)$ denote the first and last letter of $x \neq \varepsilon$. The projection of a string $x \in \Sigma^*$ on Δ is denoted by $\pi_\Delta(x)$.

The operators union, concatenation, and Kleene star are called *regular*. A *regular expression* is a formula written using the regular operators, parentheses and letters from a specified alphabet.

A *context-free* (CF) grammar is a 4-tuple $G = (V_N, \Sigma, P, S)$, where V_N is the nonterminal alphabet, P the rule (or production) set, and S the axiom. The *total* alphabet is $V = V_N \cup \Sigma$. An *empty rule* has ε as the right-hand side (r.h.s.). A *renaming rule* has one nonterminal as r.h.s. A grammar is *reduced* if every rule can be used to generate some string in Σ^*. It is *invertible* if no two rules have identical r.h.s.

The following naming convention will be adopted, unless otherwise specified: lowercase Latin letters $a, b, \ldots$ denote terminal characters; uppercase Latin letters $A, B, \ldots$ denote nonterminal characters; letters $u, v, \ldots$ denote terminal strings; and Greek letters $\alpha, \ldots, \omega$ denote strings over $\Sigma \cup V_N$. The strings may be empty, unless stated otherwise.

Grammars:

$$G_1 = \{E \to E + T \mid T, \qquad T \to T \times a \mid a\}$$
$$G_2 = \{E \to E + T \mid T, \qquad T \to T \times F \mid F, \qquad F \to (E) \mid a\}$$

Precedence matrices:

$M_1 =$

	a	$+$	$\times$
a		$\gtrdot$	$\gtrdot$
$+$	$\lessdot$	$\gtrdot$	$\lessdot$
$\times$	$\doteq$		

$M_2 =$

	a	$+$	$\times$	$($	$)$
a		$\gtrdot$	$\gtrdot$		
$+$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\gtrdot$
$\times$	$\lessdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
$($	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$
$)$		$\gtrdot$	$\gtrdot$		$\gtrdot$

Fig. 1. Example of FG for arithmetic expressions without and with parentheses. Letter a denotes a variable or a constant.

The *stencil* of a rule $A \to u_0 A_1 u_1 \dots u_{k-1} A_k u_k$, $k \geqslant 0$, is

$$N \to u_0 N u_1 \dots u_{k-1} N u_k, \quad \text{where } N \text{ is not in } V_N.$$

A rule is in *operator form* if its r.h.s. does not contain adjacent nonterminals; an *operator grammar* (OG) contains only such rules. Any CF grammar admits an equivalent OG, which can be also assumed to be invertible [19,20].

For a CF grammar G over Σ, the associated *parenthesis grammar* [5] $\widetilde{G}$ is obtained from G by enclosing each r.h.s. of a rule within the parentheses '[' and ']' that are assumed not to be in Σ.

Two grammars G, G' are *equivalent* if they generate the same language, i.e., $L(G) = L(G')$. They are *structurally equivalent* if in addition the corresponding parenthesis grammars are equivalent, i.e., $L(\widetilde{G}) = L(\widetilde{G'})$.

For a grammar G a *sentential form* $\alpha \in V^*$ is a string such that $S \stackrel{*}{\Rightarrow} \alpha$. Let α be a sentential form such that $\alpha = \beta A$, $A \in V_N$; then A is a *suffix of the sentential form* (SSF). Similarly, we define the *prefix of a sentential form* (PSF).

The following definitions for operator precedence grammars [4], here renamed *Floyd grammars* (FG), are from [8].

For an OG G and a nonterminal A, the *left and right terminal sets* are

$$\mathcal{L}_G(A) = \{a \in \Sigma \mid A \stackrel{*}{\Rightarrow} Ba\alpha\}, \qquad \mathcal{R}_G(A) = \{a \in \Sigma \mid A \stackrel{*}{\Rightarrow} \alpha a B\} \tag{1}$$

where $B \in V_N \cup \{\varepsilon\}$. The two definitions are extended to a set W of nonterminals and to a string $\beta \in V^+$ via

$$\mathcal{L}_G(W) = \bigcup_{A \in W} \mathcal{L}_G(A) \quad \text{and} \quad \mathcal{L}_G(\beta) = \mathcal{L}_{G'}(D) \tag{2}$$

where D is a new nonterminal and G' is the same as G except for the addition of the rule $D \to \beta$. Notice that $\mathcal{L}_G(\epsilon) = \emptyset$. The definitions for $\mathcal{R}$ are similar. The grammar name G will be omitted unless necessary to prevent confusion.

R. Floyd took inspiration from the traditional notion of precedence between arithmetic operators in order to define a broad class of languages, such that the shape of the parse tree is solely determined by a binary relation between terminals that are consecutive, or become consecutive after a bottom-up reduction step.

For an OG G, let α, β range over $(V_N \cup \Sigma)^*$ and $a, b \in \Sigma$. Three binary operator precedence (OP) relations are defined:

$$\begin{aligned}
&\text{equal in precedence:} \quad a \doteq b \quad \Leftrightarrow \quad \exists A \to \alpha a B b \beta,\ B \in V_N \cup \{\varepsilon\},\\
&\text{takes precedence:} \quad a \gtrdot b \quad \Leftrightarrow \quad \exists A \to \alpha D b \beta,\ D \in V_N \text{ and } a \in \mathcal{R}_G(D),\\
&\text{yields precedence:} \quad a \lessdot b \quad \Leftrightarrow \quad \exists A \to \alpha a D \beta,\ D \in V_N \text{ and } b \in \mathcal{L}_G(D).
\end{aligned} \tag{3}$$

For an OG G, the *operator precedence matrix* (OPM) $M = OPM(G)$ is a $|\Sigma| \times |\Sigma|$ array that to each ordered pair (a, b) associates the set M_{ab} of OP relations holding between a and b. For two OPMs M_1 and M_2, we define set inclusion and union:

$$\begin{aligned}
&M_1 \subseteq M_2 \quad \text{if } \forall a, b\colon\ M_{1,ab} \subseteq M_{2,ab},\\
&M = M_1 \cup M_2 \quad \text{if } \forall a, b\colon\ M_{ab} = M_{1,ab} \cup M_{2,ab}.
\end{aligned} \tag{4}$$

Definition 1. G is an operator precedence or *Floyd grammar* (FG) if, and only if, $M = OPM(G)$ is a *conflict-free* matrix, i.e., $\forall a, b\colon\ |M_{ab}| \leqslant 1$. Two matrices are *compatible* if their union is conflict-free. A matrix is total if it contains no empty case.

To illustrate, we present in Fig. 1 two variants of a classical construct: the arithmetic expressions with prioritized operators, without parentheses (G_1), and with (G_2). In the following all precedence matrices are conflict-free.

Next we define two normal forms.

Languages:

$$L_1 = \{b^n c^n \mid n \geq 1\}, \quad L_2 = \{f^n d^n \mid n \geq 1\}, \quad L_3 = \{e^n (fb)^n \mid n \geq 1\}$$

Homogeneous normal form grammars:

$$F_1 = \{S_1 \to A, \quad A \to bAc \mid bc\}$$
$$F_2 = \{S_2 \to B, \quad B \to fBd \mid fd\}$$
$$F_3 = \{S_3 \to C, \quad C \to eCfb \mid efb\}$$

Precedence matrices:

$M_1 =$

	b	c	d	e	f
b	$\lessdot$	$\doteq$			
c		$\gtrdot$			
d					
e					
f					

$M_2 =$

	b	c	d	e	f
b					
c					
d			$\gtrdot$		
e					
f			$\doteq$		$\lessdot$

$M_3 =$

	b	c	d	e	f
b					$\gtrdot$
c					
d					
e				$\lessdot$	$\doteq$
f	$\doteq$				

Fig. 2. Example of homogeneous and right-bounded grammars. The three matrices are conflict-free and precedence compatible (in particular they are disjoint). Let $M = M_1 \cup M_2 \cup M_3$. Then the grammars belong to the family C_M of precedence-compatible grammars, as well as to $C_{M,4}$ and to $C_{M,\doteq}$.

Definition 2 *(Normal forms of FG).* An FG is in *Fischer normal form* [21] if it is invertible, the axiom S does not occur in the r.h.s. of any rule, no empty rule exists except possibly $S \to \varepsilon$, the other rules having S as left-hand side are renaming, and no other renaming rules exist.

An FG is in *homogeneous normal form* [8] if it is in Fischer normal form and, for any rule $A \to \alpha$ with $A \neq S$, $\mathcal{L}(\alpha) = \mathcal{L}(A)$ and $\mathcal{R}(\alpha) = \mathcal{R}(A)$.

Thus in a homogeneous grammar, for every nonterminal symbol other than S, all of its alternative rules have the same pairs of left and right terminal sets. The definition is illustrated in Fig. 2.

Statement 1. *(See [8].) For any FG G a structurally equivalent homogeneous FG H can be effectively constructed.*

Precedence compatible families. Next we define the family of FG having identical or compatible precedence relations, deferring to Section 4 the discussion of their boolean closure properties.

Definition 3 *(Precedence-compatible grammars).* For a precedence matrix M, the class [8] C_M of *precedence-compatible Floyd grammars* is $C_M = \{G \in FG \mid OPM(G) \subseteq M\}$.

The relation $\doteq$ of an FG is connected with an important parameter of the grammar, namely the length of the right-hand sides of the rules. Clearly, a rule $A \to A_1 a_1 \dots A_t a_t A_{t+1}$, where each A_i is a possibly missing nonterminal, is associated with relations $a_1 \doteq a_2 \doteq \cdots \doteq a_t$. If the $\doteq$ relation is circular, there is no finite bound on the length of the r.h.s. of a production. Otherwise the length is bounded by $2 \cdot c + 1$, where $c \geqslant 1$ is the length of the longest $\doteq$-chain. For both practical and mathematical reasons, when considering the class of FG associated to a given OPM, it is convenient to focus on grammars with bounded right-hand sides. This can be done in two ways.

Definition 4 *(Right-bounded grammars).* The class $C_{M,k}$ of FG with *right bound* $k \geqslant 1$ is defined as

$$C_{M,k} = \left\{G \in C_M \;\middle|\; \forall \text{ rule } A \to \alpha \text{ of } G\colon\ |\alpha| \leqslant k\right\}.$$

The class of *$\doteq$-acyclic* FG is defined as

$$C_{M,\doteq} = \left\{G \in C_M \;\middle|\; OPM(G) \text{ is } \doteq\text{-acyclic}\right\}.$$

A class of FG is *right-bounded* if it is k-right-bounded for some k.

The class of $\doteq$-acyclic FG is obviously right-bounded. Notice also that, for any matrix M, the set of the rule stencils of the grammars in $C_{M,k}$ (or in $C_{M,\doteq}$) is finite.

3. Language family relations

In this section we compare the generative power and the structural adequacy of FG versus other well-known grammar or automata families that aim at extending typical decidability and closure properties beyond the limits of regular languages. After the essential definitions of a relevant sample of such families, we discuss the relations of FL with them. For brevity, other classes are not defined here because they can be put in some relation with the above "basic" ones; they are nevertheless taken into consideration in Section 4.6. The sample includes the balanced grammars and several models of pushdown automata characterized by various restrictive hypotheses that lead to height-deterministic and visibly pushdown behaviors. For simplicity, the same name is sometimes given to a class of devices (grammars or automata) and to the corresponding language class; for reader convenience class abbreviations are listed in Appendix A.

Let us first recall the relation of FL to the classical language families.

3.1. FL versus regular and deterministic languages

FL is a rich subclass of the deterministic CF languages, which is expressive enough to be used for specifying real programming languages without twisting too much their syntax.

The fact that DCF languages strictly include FL is well known: a typical non-Floyd language is

$$L_1 = \{a^n b a^n \mid n \geqslant 0\}. \tag{5}$$

After D. Knuth showed that LR(k) grammars exactly match DCF languages and can be used to build efficient bottom-up parsers, the use of FG and other types of precedence grammars within syntax-directed compilers declined. A goal of this paper is to show that other recent technological motivations should renew the interest for FG, thanks to their formal properties.

It is also well known that regular languages are a special case of FL, as stated more precisely next.

Statement 2. *Let $R \subseteq \Sigma^*$ be a regular language. There exists a grammar for R in the family $C_{M,2}$ of precedence compatible grammars, where M is the precedence matrix such that $M_{ab} = \lessdot$ for all $a, b \in \Sigma$.*

This follows from the fact that every regular language can be generated by a *right-linear grammar*, whose rules have the form $A \to aB$, $a \in \Sigma, B \in V_N$, therefore the r.h.s. length is bounded by 2 and the only precedence relations are $\lessdot$. If the empty string is in R, the FG has rule $S \to \varepsilon$. A stronger statement holding for any precedence matrix will be proved in Section 4.5 as a corollary of the main closure theorems.

3.2. Pushdown machine

Next, we compare FL with more recent and less classical (DCF) language families. First, we restate the standard definition of pushdown machine in the following equivalent way.

Definition 5. A *pushdown automaton* (PDA) is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Q_F)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, and $Q_F \subseteq Q$ are the final states. Γ is the stack alphabet that contains the special bottom-of-stack symbol $\perp$. The transition function is

$$\delta : Q \times (\Sigma \cup \varepsilon) \times \Gamma \to \wp\big(Q \times \big(\Gamma \setminus \{\perp\}\big)^*\big),$$

where $\wp(X)$ denotes the set of all finite subsets of X.

To denote a move or transition, the notation $pX \xrightarrow{a} q\alpha$ is sometimes used instead of $(q, \alpha) \in \delta(p, a, X)$ or equivalently $\delta(p, a, X) \ni (q, \alpha)$. Performing this move, the machine starts in current state p with top-of-stack symbol X, reads the current letter a (unless a is the empty string), pops X and pushes string α, and finally enters state q.

A PDA is *realtime* (RPDA) if $pX \xrightarrow{a} q\alpha$ implies $a \neq \varepsilon$.

A PDA is *deterministic* (DPDA) if, for every $p \in Q$, $X \in \Gamma$ and $a \in \Sigma \cup \{\varepsilon\}$, we have

$$\big|\{q\alpha \mid pX \xrightarrow{a} q\alpha\}\big| \leqslant 1 \quad \text{and}$$

$$\text{if } pX \xrightarrow{a} q\alpha,\ a \neq \varepsilon, \text{ is defined,} \quad \text{then } pX \xrightarrow{\varepsilon} q'\alpha' \text{ is not defined.}$$

A *realtime deterministic* PDA is named a RDPDA.

The set $Q\Gamma^*$ is the set of *configurations* of a PDA, and the *initial* configuration is $q_0\perp$.

The *labelled transition system* generated by $\mathcal{A}$ is the edge-labeled directed graph

$$(V, E) = \Big(Q\perp\Gamma^*, \bigcup_{a \in \Sigma \cup \{\varepsilon\}} \xrightarrow{a}\Big),$$

where $\xrightarrow{a}$ is defined in the natural way from the notation defining the transition function. Given a string $w \in \Sigma^*$, we write $p\alpha \overset{w}{\Rightarrow} q\beta$ if there exists a finite w'-labelled path, $w' \in (\Sigma \cup \{\varepsilon\})^*$, from $p\alpha$ to $q\beta$, and w is the projection of w' onto Σ. Notice that the w'-labelled path may include transitions with empty label; when we need to talk of the empty label as a letter, we use the marked copy $\underline{\varepsilon}$ instead of ε.

A PDA $\mathcal{A}$ is *complete* if for every $w \in \Sigma^*$, it holds $q_0\perp \overset{w}{\Rightarrow} q\alpha$.

The language *recognized* by $\mathcal{A}$ is $L(\mathcal{A}) = \{w \in \Sigma^* \mid q_0\perp \overset{w}{\Rightarrow} p\alpha,\ \ p \in Q_F\}$.

Definition 6. A PDA $\mathcal{A}$ is *normalized* [16] if

1. $\mathcal{A}$ is complete;
2. for all $p \in Q$, all rules in δ of the form $pX \xrightarrow{a} q\alpha$ either satisfy $a \in \Sigma$, or all of them satisfy $a = \varepsilon$, but not both;
3. every rule in δ is of one of the forms:
   - $pX \xrightarrow{a} q\varepsilon$, abbreviated to $pX \xrightarrow{a} q$,
   - $pX \xrightarrow{a} qX$,
   - $pX \xrightarrow{a} qXY$, where $a \in \Sigma \cup \{\varepsilon\}$.

For a normalized PDA, a generic move $pX \xrightarrow{a} q\alpha$ is named as follows: *push* if $|\alpha| = 2$, *pop* if $|\alpha| = 0$, and *internal* if $|\alpha| = 1$. Normalization of a PDA preserves its property of being deterministic, realtime, and realtime deterministic.

3.3. Height-deterministic languages

This notion, introduced by [16] for a non-deterministic PDA, expresses the property that all the computations that are labelled by the same string construct a stack of the same length (or height).

Definition 7. Let $w \in (\Sigma \cup \{\underline{\varepsilon}\})^*$. The set $N(\mathcal{A}, w)$ of *stack heights* reached by $\mathcal{A}$ after reading w is $\{|\alpha| \mid q_0\perp \overset{w}{\Rightarrow} q\perp\alpha\}$. A *height-deterministic* PDA (HPDA) is a normalized machine such that $|N(\mathcal{A}, w)| = 1$ for every $w \in (\Sigma \cup \{\underline{\varepsilon}\})^*$.

The families of height-deterministic PDA, DPDA, and RDPDA are respectively denoted by HPDA, HDPDA, and HRDPDA; the acronyms of the corresponding language families end with "'L"' instead of "'A"'.

This definition actually reformulates in the framework of height-determinism the idea of synchronization of PDA due to Caucal [14].

We recall from [16] that any PDA can be converted into a normalized machine that is height-deterministic; if the original machine is deterministic, its normalized version is height-deterministic.

Definition 8. Two HPDA's $\mathcal{A}_1$ and $\mathcal{A}_2$ over the same alphabet Σ are in the equivalence relation named *H-synchronized* and denoted by $\mathcal{A}_1 \sim_H \mathcal{A}_2$, if $N(\mathcal{A}_1, w) = N(\mathcal{A}_2, w)$ for every $w \in (\Sigma \cup \{\underline{\varepsilon}\})^*$.

Then, $[\mathcal{A}]_{\sim_H}$ denotes the equivalence class containing the HPDA $\mathcal{A}$, and $\mathcal{A}_{HPDA}$ denotes the class of languages recognized by any HPDA H-synchronized with $\mathcal{A}$.

The comparison with language families characterized by height determinism is very brief. Clearly, for an FG the corresponding bottom-up parsing algorithms [9] implement a height-deterministic PDA, which however does not necessarily operate in real time.

Next we exhibit a non-realtime deterministic language that is in FG:

$$L_2 = \left\{a^m b^n c^n d^m \;\middle|\; m, n \geqslant 1\right\} \cup \left\{a^m b^+ e d^m \;\middle|\; m \geqslant 1\right\}. \tag{6}$$

On the other hand, language L_1 of Eq. (5) is obviously a HRDPDA language. Therefore we have:

Statement 3. *The families FL and HRDPDL are incomparable.*

3.4. Visibly pushdown automata

Before we move to the visibly pushdown machine class, we need to align the classical definition of pushdown machine (Definition 5) to the one of [2] (also in [3]).

Definition 9. A *visibly pushdown* (VP) alphabet is a 3-ple $\widehat{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_i \rangle$, where the three alphabets are disjoint and respectively named *calls*, *returns* and *internals*, and $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_i$.

A *VP automaton* (VPA) is a normalized realtime PDA $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Q_F)$, where $\widehat{\Sigma}$ is a VP alphabet, and the transition relation takes one of the cases, respectively shortened as shown:

pop transition: $\delta(p,a,X) \ni (q,\varepsilon)$, $a \in \Sigma_r$; short: $\delta(p,a,X) \ni q$;
pop on empty stack: $\delta(p,a,\perp) \ni (q,\perp)$, $a \in \Sigma_r$; short: $\delta(p,a,\perp) \ni q$;
internal transition: $\delta(p,a,X) \ni (q,X)$, $a \in \Sigma_i$; short: $\delta(p,a) \ni q$;
push transition: $\delta(p,a,X) \ni (q,XY)$, $a \in \Sigma_c$; short: $\delta(p,a) \ni (q,Y)$.

A language over $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_i$ is a visibly pushdown language (VPL) if it is recognized by a VPA having $\widehat{\Sigma}$ as VP alphabet.

A *balanced grammar*, introduced by [12] to describe XML languages and also as a generalization of the parenthesis grammars of [5] and [22], is a CF grammar extended with regular expressions, which generates well-formed strings over a set of parentheses.

Definition 10. In a *balanced grammar* the terminal alphabet is partitioned into $\Sigma = \Sigma_{par} \cup \Sigma_i$, where $\Sigma_{par} = \{a, \bar{a}, b, \bar{b}, \ldots\}$ is a set of pairs of *matching parentheses*, and the elements of Σ_i are named *internal*. Furthermore, every rule has the form $X \to a\alpha\bar{a}$, where X is a nonterminal, $a, \bar{a}$ is a matching pair in Σ_{par}, and α is a regular expression over $V_N \cup \Sigma_i$. The corresponding language family is denoted BALAN.

An example is the grammar

$$X \to aY^*\bar{a}, \qquad Y \to b\bar{b}.$$

3.5. Comparison with visibly pushdown languages and balanced languages

The main result of this section is that VPL are a well-characterized special case of Floyd languages. Since VPL are CF, previous papers (e.g. [3]) have also used grammars to define them, but such grammars are not OG or have precedence conflicts; instead, we present a construction producing a grammar with the required properties.

First, we give a construction from a VPA to an FG having a certain type of precedence matrix; second, we construct a VPA for any FG with such matrices. At last, we include also BALAN in the matrix-based characterization.

Preliminarily we need to analyze the structure of VPL sentences. We use letters c, r, and s respectively for calls, returns, and internal characters. A string in $\{c,r\}^*$ is *well parenthesized* if it reduces to ε via the cancellation rule $cr \to \varepsilon$.

Let ρ be the alphabetical mapping from $\Sigma_c \cup \Sigma_r \cup \Sigma_i$ to $\{c,r\}$ defined by $\rho(c_j) = c, \forall c_j \in \Sigma_c$, $\rho(r_j) = r$, $\forall r_j \in \Sigma_r$, and $\rho(s_j) = \varepsilon$, $\forall s_j \in \Sigma_i$. A string $x \in \Sigma^*$ is *well balanced* if $\rho(y)$ is well parenthesized; it is *well closed* if in addition $\mathit{first}(x) \in \Sigma_c$ and $\mathit{last}(x) \in \Sigma_r$.

Let $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Q_F)$ be a VPA, with $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_i$.

Lemma 4. *Any string $x \in L(\mathcal{A})$ can be factorized as $x = yc_0z$ or $x = y$, with $c_0 \in \Sigma_c$, such that*

1. *$y = u_1w_1u_2w_2\ldots u_kw_k$, $k \geqslant 1$, where $u_j \in (\Sigma_i \cup \Sigma_r)^*$, and $w_j \in \Sigma^*$ is a well-closed string, or $y = u_1w_1u_2w_2\ldots u_k$, $k \geqslant 1$.*
2. *$z = v_1c_1v_2c_2\ldots c_{r-1}v_r$, $r \geqslant 0$, where $c_j \in \Sigma_c$ and $v_j \in \Sigma^*$ is a well-balanced string.*

Proof. Let the transitions from state q to q' be labelled as follows: $(r,\perp)$ denotes a move of type $\delta(q,r,\perp) \ni q'$; (r,Z) denotes a move of type $\delta(q,r,Z) \ni q'$ with $Z \neq \perp$; $\frac{c}{Z}$ denotes a move of type $\delta(q,c) \ni (q',Z)$; s denotes a move of type $\delta(q,s) \ni q'$.

We examine the possible sequences of moves of a suitable VPA $\mathcal{A}$, which for convenience is non-deterministic (determinization is always possible [2]). We only discuss the case $x = yc_0z$, since the case $x = y$ is simpler.

Without loss of generality, we assume that the initial state q_0 is not reentered after the initial move. The computation starts with a series of moves in $\{(r,\perp) \mid s\}^*$, which scan the prefix u_1 and leave the stack empty.

Then the machine may do a series of moves to scan string w_1. The first move is of type $\frac{c}{Z_i}$. The move is possibly followed by a nested computation scanning a well-balanced string, and at last by a move of type (r,Z_i). The effect is to scan a well-closed string w_1. Clearly the nested computation may also include internal moves.

After scanning w_1 the stack is empty, and the computation may scan u_2, and so on, until w_k is scanned.

We denote by Q_q the set of states traversed during the scan of y.

At some point, when the stack is empty and the input symbol is in Σ_c, the machine non-deterministically changes behavior to scan c_0z: it makes use of a set of states named Q_p that is disjoint from $Q_q \cup \{q_0\}$. The machine performs a move $\frac{c_0}{Z_U}$, where Z_U denotes a symbol written on the stack, which will never be touched by a subsequent pop move. In other words, c_0 is non-deterministically assumed to be an unmatched call.

Then the z phase non-deterministically scans a well-balanced string v_1. Then, again non-deterministically, it may perform a move $\frac{c_1}{Z_U}$. Then it may scan another well-balanced string v_2, and so on, ending with a stack in $\perp Z_U{}^+$.

At any time, when the machine enters a final state, it may halt and recognize the scanned input. □

Clearly string y is the longest prefix such that the accepting computation ends with empty stack.

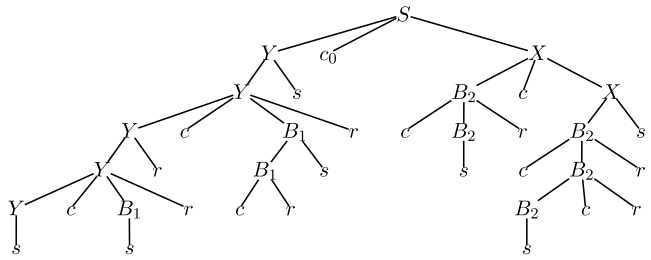

Fig. 3. Schema of a syntax tree generated by the precedence grammar constructed in Theorem 5.

Theorem 5. *For any visibly pushdown automaton $\mathcal{A}$ a Floyd grammar G such that $L(G) = L(\mathcal{A})$ can be effectively constructed.*

Proof. First we construct the grammar, then we prove that it is an FG, and lastly that it is equivalent to $\mathcal{A}$.

Grammar construction. We build our grammar G in such a way that, for a string x factorized as in Lemma 4, it generates x as the frontier of a syntax tree such as the one depicted in Fig. 3, where calls, returns, and internal letters are respectively denoted c, r, and s: for instance the leftmost s of Fig. 3 corresponds to substring u_1 of the lemma, csr corresponds to w_1, etc.

In the figure, the symbol Y denotes a nonterminal that generates a string such that the automaton, parsing it, starts and ends with empty stack.

Nonterminals of classes B_1, B_2 derive well-balanced (but not necessarily well-closed) strings.

Nonterminals of class X derive strings such that, starting with a non-empty stack of the form $\perp Z_U^+$, the stack never pops a Z_U and at last contains a string in $\perp Z_U^+$.

The nonterminal symbols of the grammar other than S are denoted by a pair of states $\langle q_i, q_j\rangle$ or $\langle p_i, p_j\rangle$, or by a triple $\langle q_i, Z, q_j\rangle$ or $\langle p_i, Z, p_j\rangle$, with $Z \in \Gamma$. Intuitively, a nonterminal of the generic form $\langle t_i \dots t_j\rangle$ generates a terminal string u if, and only if, there is a computation of the machine from the left state t_i to the right state t_j which reads the same string and never pops the initial stack. Furthermore, nonterminals $\langle q_i, q_j\rangle$ leave the stack unchanged; nonterminals $\langle p_i, p_j\rangle$ at most increase the number of Z_Us; and nonterminals $\langle q_i, Z, q_j\rangle$ or $\langle p_i, Z, p_j\rangle$ denote that the computation starts and ends with Z on the top and scans a well-balanced terminal string w.

Let us now derive G's rules from $\mathcal{A}$'s transitions.

As a first simple case, suppose that the input string x coincides with y. Thus its derivation starts with a rule of type $S \to Y$. In this case we simply build it as $S \to \langle q_0, q_f\rangle$ for every $q_f \in Q_F$. In fact the scanning of x by $\mathcal{A}$ starts in q_0 and ends in a final state with empty stack.

In the more general case when x ends with unmatched calls, the starting rule will be of type $S \to Yc_0X$. This means that $\mathcal{A}$ will scan substring y starting in q_0 and ending in some q_i with empty stack; then, reading c_0, $\mathcal{A}$ will push Z_U on the stack and move to some p_j; finally, it will scan z and end in some accepting state p_f, possibly pushing more Z_Us on the stack. Since the presence of such Z_U symbols is irrelevant for string acceptance, they do not appear in G, given that their presence is already marked by the use of letter p to denote the states; thus for any (p_j, Z_U) belonging to $\delta(q_i, c_0)$ we build the rule $S \to \langle q_0, q_i\rangle c_0 \langle p_j, p_f\rangle$, for every $p_f \in Q_F$.

Other cases of rules with S as l.h.s. are built in similar ways.

Among the various other cases, consider for instance a rule of type $Y \to cBr$: this means that $\mathcal{A}$ starts and ends scanning the string derived from Y with empty stack; however, after reading c it must push a symbol on the stack, say Z, which will be popped at the end of the scanning when reading r; notice also that, if a rule of this type is applied, this means that the input string x begins with c; thus, if (q_t, Z) belongs to $\delta(q_0, c)$ and q_h belongs to $\delta(q_k, r, Z)$, then we build the rule $\langle q_0, q_h\rangle \to c\langle q_t, Z, q_k\rangle r$, which, of course, will be applied only if $\mathcal{A}$ can go from q_t to q_h, starting and ending with Z on the top of the stack.

At this point the reader should be able to derive all other possible cases by herself. However, for the sake of completeness, Tables 3, 4, 5, and 6 in Appendix B.1 exhaustively list G's rules.

Appendix B.1 also contains a proof of the equivalence between the language generated by G and the one accepted by $\mathcal{A}$. Since it is based on a classical, though rather detailed, induction, it is not included here.

Now, let us show that G is an FG.

G is a Floyd grammar. By construction all the rules are in operator form. To verify that the operator precedence matrix M is conflict-free, it suffices to compute the relevant terminal sets and then the matrix entries using previous definitions. It should be enough to show one case.

For rule $\langle q_0, q_n\rangle \to \langle q_0, q_i\rangle c\langle q_j, Z, q_m\rangle r$ the set $\mathcal{R}_G(\langle q_0, q_i\rangle) \subseteq \Sigma_i \cup \Sigma_r$ produces the relations $s \gtrdot c$, $r \gtrdot c$.

The sets $\mathcal{L}_G(\langle q_j, Z, q_m\rangle) \subseteq \Sigma_i \cup \Sigma_c$, $\mathcal{R}_G(\langle q_j, Z, q_m\rangle) \subseteq \Sigma_i \cup \Sigma_r$ determine $c \lessdot c$, $c \lessdot s$ and $s \gtrdot r$, $r \gtrdot r$; the right part of the rule gives $c \doteq r$. Thus we obtain a conflict-free matrix $M \subseteq M_T$ where M_T is the total matrix in Fig. 4. □

	Σ_c	Σ_r	Σ_i
Σ_c	$\lessdot$	$\doteq$	$\lessdot$
Σ_r	$\gtrdot$	$\gtrdot$	$\gtrdot$
Σ_i	$\gtrdot$	$\gtrdot$	$\gtrdot$

Fig. 4. *Total VP precedence matrix* M_T.

$$\bot \text{ - } \bot$$

$$\bot \lessdot \text{ - } c_0 \gtrdot \text{ - } \bot$$

$$\bot \lessdot \text{ - } s \gtrdot c_0 \lessdot \text{ - - - - - - - } c \gtrdot \text{ - - - - - - - - - - - - - } \bot$$

$$\bot \lessdot \text{ - - - - - - - - - - - - } c \doteq \text{ - - - - - - - } r \gtrdot s \gtrdot c_0 \lessdot \text{ - - - - - - - } c \lessdot \text{ - - - - - - - - - - - - } s \gtrdot \bot$$

$$\bot \lessdot \text{ - - - - - - - - - } r \gtrdot c \doteq \text{ - - - - - - - } r \gtrdot s \gtrdot c_0 \lessdot \text{ - - - - - - - } c \lessdot c \doteq \text{ - - - - - - - } r \gtrdot s \gtrdot \bot$$

$$\bot \lessdot \text{ - - } c \doteq \text{ - - } r \gtrdot r \gtrdot c \lessdot \text{ - - - - - } s \gtrdot r \gtrdot s \gtrdot c_0 \lessdot c \doteq \text{ - - } r \gtrdot c \lessdot c \lessdot \text{ - - } c \doteq r \gtrdot r \gtrdot s \gtrdot \bot$$

$$\bot \lessdot s \gtrdot c \lessdot s \gtrdot r \gtrdot r \gtrdot c \lessdot c \doteq r \gtrdot s \gtrdot r \gtrdot s \gtrdot c_0 \lessdot c \lessdot s \gtrdot r \gtrdot c \lessdot c \lessdot s \gtrdot c \doteq r \gtrdot r \gtrdot s \gtrdot \bot$$

corresponding to

$$\bot \lessdot \underbrace{s}_{u_1} \gtrdot \underbrace{c \lessdot s \gtrdot r}_{w_1} \gtrdot \underbrace{r}_{u_2} \gtrdot \underbrace{c \lessdot c \doteq r \gtrdot s \gtrdot r}_{w_2} \gtrdot \underbrace{s}_{u_3} \gtrdot c_0 \lessdot \underbrace{c \lessdot s \gtrdot r}_{v_1} \gtrdot c \lessdot \underbrace{c \lessdot s \gtrdot c \doteq r \gtrdot r \gtrdot s}_{v_2} \gtrdot \bot$$

Fig. 5. Precedence relations between letters during the parsing of the string of Fig. 3. For the dummy string delimiter $\bot$, by hypothesis, the precedence relations with any other letter are as shown. After each pass the precedence relations are recomputed.

To exemplify, Fig. 5 reproduces the string of Fig. 3 throughout its bottom-up parsing, together with the precedence relations between letters that are consecutive or separated by a nonterminal. Any innermost pattern of the type $\lessdot \ldots \gtrdot$ delimits a string to be matched against the right part of a rule and to be reduced to the corresponding left part.

Strict inclusion. A natural question is whether every FG defines a VPL or not. The answer is in the following theorem.

Theorem 6. *The VPL family is strictly included in the FL family.*

Proof. Language

$$L_{123} = \left\{ b^n c^n \;\middle|\; n \geqslant 1 \right\} \cup \left\{ f^n d^n \;\middle|\; n \geqslant 1 \right\} \cup \left\{ e^n (fb)^n \;\middle|\; n \geqslant 1 \right\} \tag{7}$$

is an FL, a fact to be proved later (see Fig. 7), but it is not a VPL. In fact, strings of type $b^n c^n$ impose that b is a call and c a return. For similar reasons, f must be a call and d a return. Strings of type $e^n (fb)^n$ impose that at least one of b and f must be a return, a contradiction for a VP alphabet. □

3.5.1. FG with a partitioned precedence matrix

We prove that the OPM structure of Fig. 4, obtained in the proof of Theorem 5, is also a sufficient condition for an FG to generate a VPL, thus obtaining a complete characterization of VPA as a subclass of FG.

Definition 11. For an alphabet Σ, let M_T be an OPM such that there exists a partition of Σ into three subsets Σ_1, Σ_2 and Σ_3 satisfying the conditions:

$$\forall a \in \Sigma_1,\ \forall b \in \Sigma_1 \cup \Sigma_3\colon \quad M_T[a,b] = \lessdot \quad \text{and} \quad \forall a \in \Sigma_1,\ \forall b \in \Sigma_2\colon \quad M_T[a,b] = \doteq,$$

$$\forall a \in \Sigma_2,\ \forall b \in \Sigma\colon \quad M_T[a,b] = \gtrdot,$$

$$\forall a \in \Sigma_3,\ \forall b \in \Sigma\colon \quad M_T[a,b] = \gtrdot.$$

Then M_T is termed a *total VP matrix* representing the VP alphabet $\widehat{\Sigma} = \langle \Sigma_1, \Sigma_2, \Sigma_3 \rangle = \langle \Sigma_c, \Sigma_r, \Sigma_i \rangle$. Any OPM $M \subseteq M_T$ is termed a *VP matrix*.

For any grammar G, such that $OPM(G)$ is a VP matrix, any rule $A \to \alpha$ has $|\alpha|_\Sigma \leqslant 2$. The possible stencils of the r.h.s. of the rules are NcN, $NcNr$, Nr, Ns, and those obtained by erasing one or more N. On the other hand, the stencils rN, crN are forbidden because r does not yield precedence to any letter. It follows that, for any FG having a VP matrix, the length of any r.h.s. is $\leqslant 4$.

Theorem 7. *Let G be an FG such that $OPM(G)$ is a VP matrix. Then $L(G)$ is a VPL.*

Proof. First, observe that if $OPM(G)$ is a VP matrix, then, for every string $x \in L(G)$, the syntax tree induces the factorization

$$x = yc_0z \quad \text{or} \quad x = y, \quad y = u_1w_1u_2w_2 \dots u_kw_k, \quad z = v_1c_1v_2c_2 \dots c_{r-1}v_r$$

where all terms are as in Lemma 4, and its syntax tree has the structure shown in Fig. 3. It suffices to consider that the precedence relations of the VP matrix completely determine the skeleton of the syntax tree, as exemplified in Fig. 5.

Next, for a given $G = (V_N, \Sigma, P, S)$ satisfying the hypothesis, let us build a VPA $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Q_F)$ along the following lines. $\mathcal{A}$ recognizes by final state and for convenience is non-deterministic. We recall that the rule stencils are the ones previously listed. We set $Q = V_N \cup \{q_0, p, q_F\}$, where $q_0, p, q_F \notin V_N$. The pushdown alphabet is

$$\Gamma = \big((V_N \cup \{-\}) \times \Sigma_c \times (V_N \cup \{-\})\big) \cup \{\bot, Z_U\}.$$

Intuitively, $\mathcal{A}$ is built in such a way that it enters a state $B \in V_N$ after finishing the scanning of a substring syntactically rooted in B.

In state B, reading a symbol $c \in \Sigma_c$ (the only ones that yield precedence), $\mathcal{A}$ enters state p and pushes onto the stack a symbol, for which two cases occur. The symbol is Z_U, if the c is not to be matched by an r; it is $\langle B, c, C\rangle$, if the machine "looks for" a well-balanced string w such that $C \stackrel{*}{\Rightarrow} w$.

Simpler special cases also occur: for instance, in correspondence to a rule such as $A \to cr$, $\mathcal{A}$ "looks" directly for r and pushes on the stack the symbol $\langle -, c, -\rangle$ (or $\langle B, c, -\rangle$ in correspondence of a rule $A \to Bcr$).

In state p, reading a c, $\mathcal{A}$ remains in the state and pushes on the stack either the symbol Z_U if the c is not to be matched, or a symbol $\langle -, c, C\rangle$ if it "looks for" a string w such that C generates w.

Finally we describe the moves that read $r \in \Sigma_r$. If the stack is empty, the machine enters a state A associated to a nonterminal. If the top of stack is a symbol $\langle B, c, C\rangle$, the machine pops the stack and enters a state A corresponding to the l.h.s. of the rule terminating with letter r.

Here too some simpler special cases exist: for instance, symmetrically to a previous pushing case, in state p, with $\langle -, C, -\rangle$ at the top of the stack, $\mathcal{A}$ just pops and enters state A.

In Appendix B.2 Table 7 lists all the transitions of $\mathcal{A}$.

$\mathcal{A}$'s final state set is defined as $Q_F = \{A \mid S \stackrel{*}{\Rightarrow} \beta A,\ A \neq S\} \cup \{q_F\} \cup \{q_0 \text{ iff } S \to \varepsilon \in P\}$. Notice that a rule $A \to cB$ can be used only in a derivation such as $S \stackrel{*}{\Rightarrow} \alpha A \Rightarrow \alpha cB \stackrel{*}{\Rightarrow} x$, otherwise c would take precedence over some other letter. Thus, A and B are both in Q_F. Furthermore, relation $S \stackrel{*}{\Rightarrow} \alpha A$ is decidable thanks to well-known properties of CF grammars: thus the effectiveness of the construction is not hampered.

Also, state q_F is used just to reproduce the effect of rules of type $S \to \alpha a$, $a \in \Sigma$.

The proof of equivalence $L(\mathcal{A}) = L(G)$ somewhat mirrors the equivalence proof of Theorem 5. In Appendix B.2 we state the basic inductive lemmas supporting the equivalence. □

3.5.2. Balanced grammars and languages

We show that the family BALAN (Definition 10) corresponds to further restrictions on the precedence matrix and/or on the stencils of FG rules. Since BALAN is a strict subfamily of VPL [2], we may assume the language to be generated by an FG G such that $OPM(G)$ is a VP precedence matrix. The following constraints are then added.

1. No unmatched calls or unmatched returns.
   Balanced languages do not allow any c_i or r_i to be unmatched. Thus an FG such that no rule has stencils Nc_iN, Nc_i, c_iN, c_i, Nr_i ensure the balancing property.
2. All sentences are bracketed by a matching pair c_i, r_i.
   Therefore the stencil of a rule $S \to \dots$ or, if $S \Rightarrow A$, of $A \to \dots$, cannot be Ns.
3. Bijection of calls and returns.
   In BALAN a c_i cannot be matched by distinct returns r_j, r_k (and similarly an r_i cannot match distinct calls). Such bijection between calls and returns is simply obtained in an FG, by imposing that $|\Sigma_c| = |\Sigma_r|$ and the OPM submatrix identified by rows $c_1, c_2, \dots$ and columns $r_1, r_2, \dots$ contains $\doteq$ only on the diagonal.

Corollary 8. *The following statements are equivalent*:

- *L is a balanced language.*
- *$L = L(G)$ where G is FG with a VP precedence matrix satisfying conditions* 1, 2, *and* 3 *above.*

For convenience we collect in Fig. 6 the main containment relations between language families, which were already known [2,16,23] or are proved here.

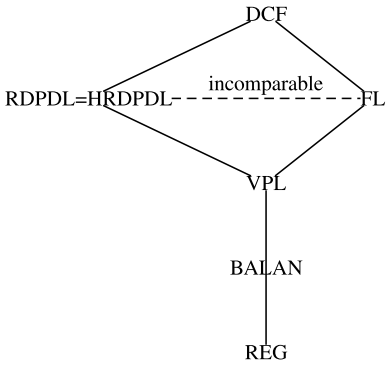

Fig. 6. Language family inclusions. All inclusions are strict.

3.6. Expressivity and structural adequacy

Let us now examine the practical impact of the comparison of FL with other families. VPL and BALAN are mainly targeted towards the formalization and analysis of so-called mark-up languages such as XML and HTML. A distinguishing feature of such languages is the presence of possibly nested, opening/closing (i.e., calling/returning) characters; this feature was initially captured by parenthesized CF languages and more recently generalized, mainly by BALAN and VPL, in various ways. However, the strict hierarchy exhibited by Fig. 6 and the counterexample (7) of Theorem 6 show that FL allow for a more general treatment of opening and closing scopes by means of multiple characters rather than by single ones: for instance keyword pairs such as /∗–∗/, *if–fi*, *begin–end*, etc. In some cases they would produce conflict in the alphabet partitioning imposed by BALAN and VPL. Usually, e.g. in the parsing of programming languages, this difficulty is avoided by a separate lexical analyzer, which translates keywords into single abstract characters (tokens or lexemes). This approach, however, though common in practice, requires a double pass with the use of different automata formalisms (finite state transducers and pushdown automata) and different alphabets.

A more important difference between VPL and FL abides in that VPL, like other classes of parenthesized languages, make the structure of the syntax tree explicit in the language sentence, whereas FL leave it implicit, though deterministically obtainable through a precedence parsing algorithm, as in their original motivation in the seminal paper by Floyd. In many practical cases, e.g., in programming and natural languages, the surface structure of the language is rather flexible and does not provide explicit tags (as it happens in markup languages such as HTML) that encode the deep structure.

Consider for instance the well-known case of arithmetic expressions of the example in Fig. 1 with two levels of operator precedence. This paradigm is also named a list of lists, each list level having distinct separator characters. For finite value k, k-level hierarchical lists are a very widespread regular language. A structurally (or semantically) adequate grammar is expected to generate the expressions with a structure reflecting their semantics, i.e., accounting for the precedence, say, of multiplication over sum. The FG in Fig. 1 have this property, unlike any VPA recognizing the same language. Adequacy of structure to semantics is impossible for a VPL, because the shape of the natural syntax tree imposes relations $\times \lessdot a$ and $\times \gtrdot +$, which violate the partitioned precedence matrix condition (Definition 11); in other words, terminal a marks both the opening and the closing of a new scope. The same language could be generated as a VPL by means, say, of the following grammar:

$$S \to a \mid S + a \mid S \times a \mid (S) \quad \text{with } \Sigma_i = \{a, X, +\},\ \Sigma_c = \{(\},\ \text{and } \Sigma_r = \{)\}.$$

However, the semantic structure of arithmetic operations would be lost and the only way to recover it in the sentence would be to make all scopes explicit through a pair of parentheses.

Structural adequacy is an essential prerequisite for syntax-directed translation, as used in compiler design and in program analysis (and of course in linguistic). We just mention a well-known example: machine instruction selection and register allocation (e.g., in [24]). This uses a tree pattern matching algorithm to cover by means of tiles the syntax tree of a given arithmetic expression. Then a dynamic programming algorithm optimally allocates registers for the expression tree. Clearly for such applications, the tree must reflect the correct precedence of the arithmetic operations.

Another limitation of VPL has to do with the fact that by definition they are recognized by realtime PDA, whereas FL are not restricted to be realtime languages.

In conclusion, VPL may be structurally adequate for expressing the paradigm of matching procedure invocations and returns, but are weak for other very common paradigms occurring in programming and natural languages.

On the other side, FG as the earliest grammar family for efficiently parsing programming languages, have been progressively superseded by more powerful grammars, such as *simple* and *weak precedence*, and LR(k) which cover the whole DCF spectrum. Unfortunately, the larger families loose the important closure properties which are instead enjoyed by VPL and FL, as we will see in Section 4. One reason for such loss of properties is that DCF languages (and other families such as height deterministic ones) impose a strict left-to-right analysis; on the contrary VPL and FL observe a “locality principle”

which allows to parse portions of the input sentence independently of its far context, a feature that is promising for parallel and incremental parsing. We will return to this discussion in the conclusions.

4. Closure properties of Floyd languages

In this section we investigate the closure properties of FL w.r.t. typical operations on formal languages. First, we resume the boolean closures originally proved in [8]. In doing so we also recall richer lattice properties which allow for a more thorough comparison of FL with other recent classes of H-synchronized languages. Then, we prove that FL are closed w.r.t. all other main operations: reversal, prefix and suffix, concatenation, and Kleene star.

4.1. Boolean operations

We need to present some concepts from [8] before we proceed to the boolean properties.

Definition 12 *(Free grammars and maxgrammars).* An FG G is *free* if it is homogeneous and, for all nonterminals $A, B \neq S$, $\mathcal{L}_G(A) = \mathcal{L}_G(B) \wedge \mathcal{R}_G(A) = \mathcal{R}_G(B)$ implies $A = B$. For an OPM M, $\mathcal{F}(M) = \{F \mid F \text{ is free } \wedge OPM(F) \subseteq M\}$ is the class of free precedence-compatible FG.

The following statements hold for right-bounded (Definition 4) classes of FG.

Statement 9. *For any OPM M and $k \geqslant 1$, a unique free FG $G_{M,k}$ exists such that $OPM(G_{M,k}) = M$ and for every $G \in C_{M,k}$, it is $L(G) \subseteq L(G_{M,k})$.*

In the quoted reference, grammar $G_{M,k}$ is termed a *maxgrammar*.

The inclusion relation between the rule sets of grammars $F_1, F_2 \in \mathcal{F}(M)$ induces a *partial order relation* $F_1 \leqslant F_2$ on free grammars, corresponding to the inclusion relation on their languages. In the quoted reference, a lattice is defined on top of this partial order, such that the maxgrammar is the top element.

Definition 13. Let $H = (V_N, \Sigma, P, S)$ be an FG in homogeneous normal form with M as OPM. The mapping

$$W : V_N \setminus \{S\} \to \wp(\Sigma) \times \wp(\Sigma)$$

is defined by

$$W(A) = \langle \mathcal{L}_H(A), \mathcal{R}_H(A) \rangle$$

and naturally extended to P (clearly $W(a) = a$, $\forall a \in \Sigma$).

The free grammar $F = W(H)$ is $F = (W(V_N) \cup \{S\}, \Sigma, W(P), S)$.

Statement 10. *Let $\mathcal{H}$ be the class of homogeneous FG. $\mathcal{H}$ is partitioned by W into mutually disjoint classes. For each class $W^{-1}(F)$ with $F \in \mathcal{F}(M)$, and for all $H \in W^{-1}(F)$, it is $L(H) \subseteq L(F)$.*

We denote by $\mathcal{F}(F) = \{G \in \mathcal{F} \mid G \leqslant F\}$ the class of free grammars that precede F in the partial order. For a free grammar F we define the class of homogeneous FG $\mathcal{H}(F) = \{H \in \mathcal{H} \mid W(H) \in \mathcal{F}(F)\}$. In other words, $\mathcal{H}(F)$ includes all homogeneous FG with the same rule stencils as F and therefore with an OPM compatible with $OPM(F)$.

The above concepts and properties are illustrated in Fig. 7.

In [8] an algebraic lattice of homogeneous FG and corresponding languages is defined and studied, leading to the next property.

Statement 11. *(See Corol. 5.7, Theorem 5.8 of [8].) Let F be a free FG. The class of FL defined by $\{L(H) \mid H \in W^{-1}(F)\}$ is closed under union, intersection, and complementation with respect to $L(F)$.*

In other words, the proposition applies to the class of FL such that: they are generated by right-bounded FG having precedence matrices which are included or equal to some precedence matrix M, and their rules are mapped on the rules of the free grammar F by the W mapping. The free grammar F has matrix M and generates the largest language within this class.

The lattice structure induced by free grammars resembles the H-synchronization of HDPDAs [16], described in Definition 8. In fact, it is easy to verify that a free maxgrammar H-synchronizes all grammars in its lattices; in particular a maxgrammar, whose OPM is complete, generates Σ^*; furthermore all grammars sharing the same OPM are such that the parsing of any string w by any pair of automata $\mathcal{A}_1$, $\mathcal{A}_2$ accepting their respective languages is such that $N(\mathcal{A}_1, w) = N(\mathcal{A}_2, w)$

1. *Three free grammars*:

$$F_1 = \{S_1 \to A, \quad A \to bAc \mid bc\}$$
$$F_2 = \{S_2 \to B, \quad B \to fBd \mid fd\}$$
$$F_3 = \{S_3 \to C, \quad C \to eCfb \mid efb\}$$

2. $\doteq$*-acyclic matrix* $M = M_1 \cup M_2 \cup M_3$:

$$M = \begin{array}{c|cccccc} & b & c & d & e & f \\ \hline b & \lessdot & \doteq & & & \gtrdot \\ c & & \gtrdot & & & \\ d & & & \gtrdot & & \\ e & & & & \lessdot & \doteq \\ f & \doteq & & \doteq & & \lessdot \end{array}$$

3. *Grammar/language families* $C_{M,4}$ *and* $C_{M,\doteq}$ *include*:

$$L_{123} = L_1 \cup L_2 \cup L_3 = L(F_{123})$$

where F_{123} has the rules $S_{123} \to A \mid B \mid C$ instead of $S_1 \to A$, etc., and is free.

4. *The family of homogeneous* FG $W^{-1}(F_3)$ *includes*:

$$H = \{S \to X_2, \quad X_2 \to eX_1fb \mid efb, \quad X_1 \to eX_2fb\}$$

Note: H is homogeneous but not free since $\mathcal{L}_H(X_2) = \mathcal{L}_H(X_1)$ and $\mathcal{R}_H(X_2) = \mathcal{R}_H(X_1)$.

5. *Language inclusion relations*:

$$L(H) \subseteq L(F_3)$$

Class $W^{-1}(F_3)$ is a boolean algebra.

Fig. 7. Precedence-compatible homogeneous grammars from the example in Fig. 2.

(some simple technicalities are needed to build a normalized complete automaton $\mathcal{A}$ accepting $L(G)$, G being an FG). However, the class of H-synchronized languages also includes languages that are not in the lattice $W^{-1}(F)$ and moreover may not even be in FL (e.g. the language a^nba^n, $n \geqslant 1$). Thus the closure of FL under boolean operations cannot be deduced from the closure of H-synchronized languages under the same operations.

4.2. Reversal, prefix, and suffix operations

Regular languages are closed under concatenation, boolean operations, reversal, and under the extraction of all prefixes and suffixes; on the other hand, some of these properties are missing in the cases of CF and DCF languages. CF is not closed under complement and intersection; DCF is not closed under concatenation, union, intersection, reversal and suffix extraction. Furthering the study of FL beyond boolean operations, we prove their closure under the remaining ones.

Although language reversal does not preserve determinism in CF languages, it is immediate to see that FL are closed under reversal, because of the left–right symmetry in their definition.

Statement 12. *FL is closed with respect to reversal* (*or mirror reflection*).

This follows from the fact that, if $a \lessdot b$ for an FG G, then it holds $b \gtrdot a$ for the grammar G^R obtained by specularly reversing the r.h.s. of the rules of G; and similarly for $a \gtrdot b$. The $a \doteq b$ relation is turned into $b \doteq a$ by rule reversal. It follows G^R is an FG, with $OPM(G^R)$ "reversed" with respect to $OPM(G)$.

Next, we show that the set of prefixes (or suffixes) of an FL is an FL with the same precedence matrix.

For a language L the *prefix language* is $L_P = \{y \mid \text{for some } z \in \Sigma^*, \ yz \in L\}$ also denoted as $pref(L)$. The *suffix language* $L_S = suf(L)$ is similarly defined.

First notice that, for a CF grammar G, it is straightforward to construct a grammar G' of $pref(L(G))$. But, if G is an FG, we also need to prove that the construction preserves conflict-freeness of $OPM(G)$.

Theorem 13. *FL is closed under prefix and under suffix operations.*

Proof. The proof, given just for prefixes since the other case is symmetric, is articulated in three steps: construction of grammar G' from the given grammar G, proof that G' has the same OPM as G, and proof that $L(G') = pref(L(G))$.

1. Construction of G'.
Let $G = (V_N, \Sigma, P, S)$ be an FG in Fischer normal form. The new grammar is $G' = (V'_N, \Sigma, P', S')$ where V'_N is the disjoint union of two sets denoted V^C and V^S, defined as follows:

$$V^C = \{A^C \mid A \in V_N \setminus \{S\}\}, \tag{8}$$
$$V^S = \{A^S \mid A \in V_N \setminus \{S\}\}. \tag{9}$$

Intuitively, the nonterminals A^S are all and only the suffixes of sentential forms (SSF) of G'.
To construct the rule set, we introduce two transformations α^C and α^S

$$\alpha^C, \alpha^S : (V_N \cup \Sigma)^* \to (V'_N \cup \Sigma)^*$$

on strings α of the form

$$\alpha = x_0 A_1 x_1 \dots A_n x_n, \quad x_i \in \Sigma^*.$$

If $x_n \neq \varepsilon$ then

$$\begin{cases} \alpha^C = x_0 A_1^C x_1 \dots A_n^C x_n, \\ \alpha^S = \text{undefined}. \end{cases}$$

If $x_n = \varepsilon$ then

$$\begin{cases} \alpha^C = x_0 A_1^C x_1 \dots A_n^C, \\ \alpha^S = x_0 A_1^C x_1 \dots A_{n-1}^C x_{n-1} A_n^S. \end{cases}$$

We also define the transformation $PREF(\alpha)$ as

$$PREF(\alpha) = \{\beta_i^S \mid \beta_i \neq \varepsilon \text{ is prefix of } \alpha\} \cup \{\beta_i^C \mid \beta_i \neq \varepsilon \text{ is prefix of } \alpha \wedge \beta_i \in V^*\Sigma\}.$$

Then, the new rule set P' contains

$$S' \to A^S, \quad \text{for every rule } S \to A \in P, \tag{10}$$
$$A^C \to \alpha^C, \quad \text{for every rule } A \to \alpha \in P, \tag{11}$$
$$A^S \to PREF(\alpha), \quad \text{for every rule } A \to \alpha \in P. \tag{12}$$

Notice that the construction can produce in G' renaming rules (besides those with the axiom as left-hand side) and also repeated r.h.s. A further step could be applied to transform G' in Fisher normal form – and possibly in homogeneous normal form – if needed in view of further elaboration. Also, some optimizations could be applied to the construction above, but we preferred to keep its description as simple as possible.

2. Proof that $OPM(G') = OPM(G)$ (obviously $OPM(G') \supseteq OPM(G)$).
It is straightforward to verify from Definition 1 the following identities of terminal sets, for every nonterminal $A \in V_N$:

$$\mathcal{R}_{G'}(A^C) = \mathcal{R}_G(A), \qquad \mathcal{L}_{G'}(A^C) = \mathcal{L}_G(A), \qquad \mathcal{L}_{G'}(A^S) = \mathcal{L}_G(A).$$

Therefore the only set that may be affected is $\mathcal{R}_{G'}(A^S)$.
For simplicity, since the nonterminal sets of the two grammars are disjoint, we drop the grammar names from the following notation.
Let us consider the case that, for some $a \in \Sigma$ and $A \in V_N$, it is $a \in \mathcal{R}(A^S) \setminus \mathcal{R}(A)$, and show that it does not alter the OPM. Thus there is some derivation

$$S' \overset{*}{\Rightarrow} \zeta A^S \overset{*}{\Rightarrow} \zeta \beta a X, \quad X \in V^S \cup \{\varepsilon\}$$

but no derivation

$$S' \overset{*}{\Rightarrow} \zeta A^C b \eta \overset{*}{\Rightarrow} \zeta \beta a X b \eta.$$

Since A^S can occur only as the rightmost symbol of a sentential form, it cannot be $a \gtrdot b$ for any $b \neq \perp$.
Moreover, the transformation $PREF(\alpha)$ applied in the construction of P' clearly cannot produce any new $\doteq$ relation.

3. Proof that $L(G') = pref(L(G))$.
We need first to observe the straightforward double implications:

$$\forall A^C: \quad S' \overset{*}{\Rightarrow} \alpha A^C \beta \quad \text{iff} \quad \beta \neq \varepsilon, \tag{13}$$
$$\forall A^S: \quad S' \overset{*}{\Rightarrow} \alpha A^S \beta \quad \text{iff} \quad \beta = \varepsilon. \tag{14}$$

(a) Let us prove by induction on the length of the derivation that, if $S \overset{*}{\Rightarrow} \alpha$, then $S' \overset{*}{\Rightarrow} PREF(\alpha)$. Notice that the particular case $\alpha \in \Sigma^*$ is included.
The induction basis is obvious by construction.
Induction step.
Assume that the thesis holds for all derivations $S \overset{k}{\Rightarrow} \alpha$. Consider now for G a generic derivation $S \overset{k+1}{\Rightarrow} \beta$ and a prefix of β, say, $\zeta\overline{\gamma}$ such that

$$S \overset{k}{\Rightarrow} \zeta A\eta \Rightarrow \zeta\overline{\gamma}\widetilde{\gamma}\eta = \beta.$$

By the induction hypothesis $S' \overset{k}{\Rightarrow} \zeta^C A^S$.
On the other hand, rule $A^S \to \overline{\gamma}$ is in P' and the induction step is completed.

(b) Conversely, it is immediate to verify that G' cannot generate any sentential form that is not a prefix of a sentential form generated by G. In fact G's sentential forms can only be cut in a derivation step that rewrites a nonterminal of type A^S, which in turn can be derived only at the right end of any sentential form. □

Corollary 14. *VPLs are closed under prefix and suffix.*

Proof. The prefix and suffix construction for FG does not alter the original OPM. Thus, if the original OPM is a partitioned precedence matrix, so is the OPM of the grammar generating the prefixes/suffixes; therefore, the generated language is still a VPL according to Theorem 7. □

In our view this is a more convincing argument than the original one [3] which "derives" closure under suffixes as a consequence of the closure under prefixes and reversal. However, these two closures together do not necessarily guarantee closure under suffixes: suffixes are not the prefixes of the reversal. As a counterexample consider the class of languages that are left-to-right deterministic and right-to-left deterministic: it is closed by definition under reversal and, as all DCF languages, it is also closed under prefix.[2] However, for the following language

$$L = \left\{ea^n c^m b^{n+m} e\right\} \cup \left\{da^n c^* b^n d\right\}, \quad m, n \geqslant 1,$$

the set of suffixes L_s is not deterministic. In fact a string such as $a^n c^k b^m e$ with $k > m \geqslant n$ does not belong to L_s, whereas string $a^n c^k b^m d$ with $k > m \geqslant n$ is in L_s; thus, string $a^n c^k b^m$, which is a prefix of both strings, cannot be parsed deterministically since in the former case k should be counted whereas in the latter one it should not (to allow for verifying $m \geqslant n$).

4.3. Concatenation

Although FG is the oldest deterministic specialization of CF, the fundamental but non-trivial questions concerning their closure under concatenation and Kleene star have never been addressed, to the best of our knowledge. This theoretical gap is perhaps due to the facts that DCF languages are not closed under these operations, and that the constructions used for other grammar or PDA families do not work for FG, because they destroy the operator grammar form or introduce precedence conflicts. The closure proofs to be presented, though rather involved, are constructive and practical. The grammars produced for the concatenation (or the Kleene star) structurally differ from the grammars of the two languages to be combined in rather surprising ways: the syntax tree of the prefix string may "invade" the other syntax tree, or conversely, and such trespasses may occur several times.

A simple case is illustrated by $L \cdot L$ where $L = a^+ \cup b^+ \cup ab$ with precedences $a \lessdot a, b \gtrdot b, a \doteq b$. Then for $y_1 = aaa$ the structure is $[a[a[a]]]$, for $y_2 = bb$ the structure is $[[b]b]$, but the structure of $y_1 \cdot y_2$ is $[a[a[ab]b]]$, which is not a composition of the two.

Let the grammars be $G_1 = (V_{N_1}, \Sigma, P_1, S_1)$ and $G_2 = (V_{N_2}, \Sigma, P_2, S_2)$, and the nonterminals be conveniently named $V_{N_1} = \{S_1, A, A_1, \ldots\}$ and $V_{N_2} = \{S_2, B, B_1, \ldots\}$, in order to have distinct names. To simplify the proofs we operate on FG in homogeneous normal form.

For two sets $\Delta_1, \Delta_2 \subseteq \Sigma$ and a precedence sign, say $\lessdot$, the notation $\Delta_1 \lessdot \Delta_2$ abbreviates $\forall a \in \Delta_1$, $\forall b \in \Delta_2$, $a \lessdot b$. Moreover, we extend precedence relations from $\Sigma \times \Sigma$ to pairs of strings $\alpha, \beta \in (\Sigma \cup V_N)^+$ such that $\alpha\beta \notin (\Sigma \cup V_N)^* \cdot V_N \cdot V_N \cdot (\Sigma \cup V_N)^*$ by positing $\alpha \lessdot \beta \Leftrightarrow \mathcal{R}(\alpha) \lessdot \mathcal{L}(\beta)$, and similarly for $\gtrdot$; for $\doteq$ the condition is $last(\pi_\Sigma\, \alpha) \doteq first(\pi_\Sigma\, \beta)$.

When writing derivations and left/right terminal sets, we usually drop the grammar name when no confusion is possible.

Theorem 15. *Let G_1, G_2 be FG such that $OPM(G_1)$ is compatible with $OPM(G_2)$. Then an FG G can be effectively constructed such that*

$$L(G) = L(G_1) \cdot L(G_2) \quad \textit{and} \quad OPM(G) \supseteq OPM(G_1) \cup OPM(G_2).$$

[2] Contrary to the statement in Fig. 10 of [3], DCF languages are not closed under suffix.

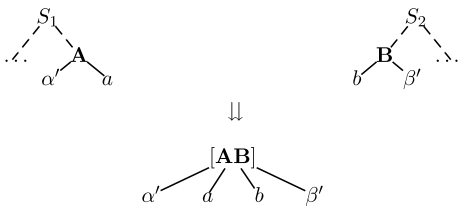

Fig. 8. Cross-border rule constructed when the facing letters are equal in precedence.

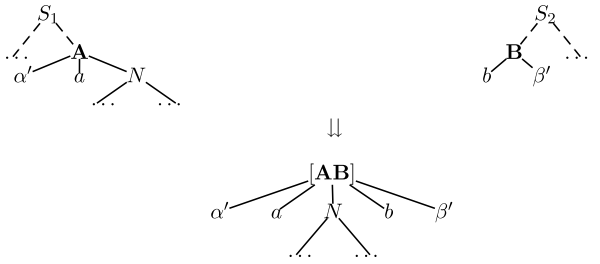

Fig. 9. Another case of cross-border rule when $\mathcal{R}(N) \gtrdot b$.

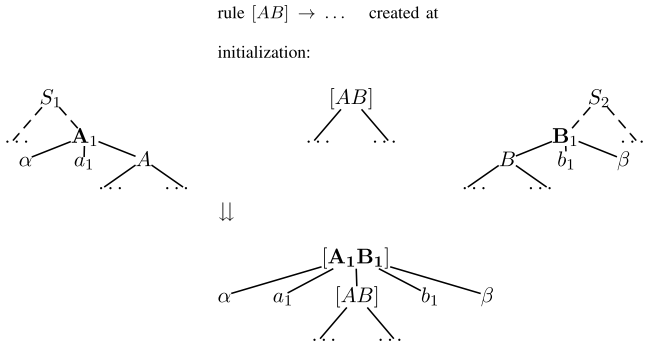

Fig. 10. Rule created by extending the cross-border thread in case $a_1 \doteq b_1$.

Proof. The full proof is reported in Appendix B.3 together with an example of the construction. Here we give a few hints on the construction of G and support the intuition by means of figures and examples. For simplicity, we assume that $M = OPM(G_1) \cup OPM(G_2)$ is a total matrix. This does not affect generality, because if, at any step, the following algorithm checks the precedence relation between two letters a and b, and $M_{ab} = \emptyset$, we can *arbitrarily* assign to M_{ab} a value, thus obtaining a matrix compatible with M.

The core of the algorithm builds a "thread" of rules that joins the parse trees of x_1 and x_2, $x_1 \in L_1$, $x_2 \in L_2$. The thread is recursively built in accordance with the precedence relations that connect the letters occurring at the right of the parse tree of x_1 and at the left of the parse tree of x_2, respectively. Since the parsing driven by operator precedence relations is bottom-up, the initialization of the construction is based on the possible "facing" of the rightmost letter of x_1 and the leftmost one of x_2. If $x_1 = y_1 \cdot a$, $x_2 = b \cdot y_2$ and $a \doteq b$, then we build a rule of the type $[AB] \to \dots ab \dots$, $[AB]$ being a new nonterminal (see Fig. 8). If instead the rightmost part of x_1 can be parsed without affecting x_2 up to a derivation $N \overset{*}{\Rightarrow} y_1$ because $\mathcal{R}(N) \gtrdot b$, then, when the parsing of x_1 leads to a rule such as $A \to \alpha' a N$ with $a \doteq b$, the jointure of the two syntax trees begins at that point by means of a rule such as $[AB] \to \alpha' a N b \beta'$ (see Fig. 9) so that the original precedence relations of G_1 and G_2 are unaffected.

Similar constructions apply if instead $a \lessdot b$.

After this initialization the construction of the "joint parsing thread" iteratively proceeds by following the natural bottom up parsing. For instance, suppose, as shown in Fig. 10, that a nonterminal of the type $[AB]$ has been built; this means that A is a SSF, B is a PSF and $[AB]$ "joins" two derivations $A \overset{*}{\Rightarrow} y_1$, at the end of a parse tree for some string x_1 of L_1 and $B \overset{*}{\Rightarrow} y_2$ at the start of a string x_2 of L_2; thus, if G_1 contains a rule $A_1 \to \alpha \cdot a_1 \cdot A$ (A_1 being a SSF) and symmetrically $B_1 \to B \cdot b_1 \cdot \beta$, with $a_1 \doteq b_1$, then the new rule $[A_1 B_1] \to \alpha \cdot a_1 \cdot [AB] \cdot b_1 \cdot \beta$ is created.

The case $a_1 \lessdot b_1$ is illustrated in Fig. 11: the left part shows a rule created at initialization and two component trees, while the right part depicts the rule created by the iterative step.

The symmetrical case $a_1 \gtrdot b_1$ is omitted. □

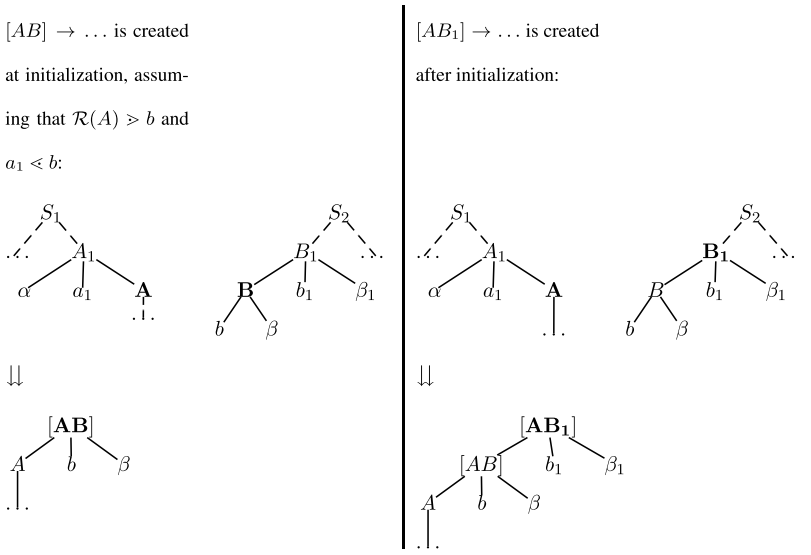

Fig. 11. Rule created by growing the cross-border thread in case $a_1 \lessdot b_1$.

4.4. Kleene star

For many language families the closure under Kleene star accompanies, though not by necessity, the closure under union and concatenation. This happens, say in a CF language L, because the syntax tree of a string $x = y_1 y_2 \ldots y_i \in L^i$ with $y_j \in L$, is simply obtained by linking, in a left- or right-linear structure, the syntax trees of the components $y_1, y_2, \ldots, y_i$. In the case of FG a similar composition in general is not possible, because the syntax tree of x may have a sharply different structure, as already observed for the concatenation $L \cdot L$.

A case is illustrated by the third power of a language $L \supset a^+ \cup b^+ \cup c^+$, assuming the precedences (induced by further sentences not considered) to be: $a \lessdot a$, $a \doteq b$, $b \gtrdot b$, $b \doteq c$, $c \lessdot c$. Then the structure of a string such as $y_1 \cdot y_2 \cdot y_3 = a^3 \cdot b^2 \cdot c^2 \in L^3$ is neither the composition of the structures of $y_1 \cdot y_2$ and y_3, nor of y_1 and $y_2 \cdot y_3$.

Before we enter the main topic, it is useful to return to the $\doteq$-acyclicity condition of Definition 4. Consider the language $L = \{aa\}$ with the circular precedence relation $M = \{a \doteq a\}$, and a string a^{2p}, $p \geqslant 0$, in the Kleene closure of L. The FG of L^* with OPM M would then need to contain an unbounded rule set $\{S \to a^{2p},\ p \geqslant 0\}$, which is not permitted by the standard definition of CF grammar. For this reason we make the hypothesis that the precedence matrix is $\doteq$-acyclic.

Theorem 16. *Let $G = (V_N, \Sigma, P, S)$ be an FG such that OPM(G) is $\doteq$-acyclic. Then an FG $\widehat{G} = (\widehat{V}_N, \Sigma, \widehat{P}, \widehat{S})$ with OPM$(\widehat{G}) \supseteq$ OPM(G) can be effectively built such that $L(\widehat{G}) = (L(G))^*$.*

As in Theorem 15 we assume without loss of generality the precedence matrix to be total. Not surprisingly the construction of $\widehat{G}$ is based on the construction in Theorem 15 for $L \cdot L$, but the required extensions involve some technical difficulties.

We need only to consider the irreflexive closure L^+, since for L^* it suffices to add the rule $S \to \varepsilon$. We always assume the form of grammar G to be homogeneous.

Again, the complete proof of the theorem is postponed to Appendix B.4 and we give here just an intuitive description of the construction.

$\widehat{G}$ is built as the last of a series of grammars that begins with $G_1 = G$ and continues with the grammar G_2 that generates $L_2 = L \cdot L \cup L$, computed according to the concatenation algorithm outlined in Section 4.3 and the union algorithm mentioned in Statement 11. Then G_3 is built by iterating the application of the concatenation algorithm to L_2 and L itself. Notice, however, that this new application produces new provisional nonterminals of the type $[[AB]C]$. Obviously this process cannot be iterated indefinitely since it would produce a grammar with infinite nonterminals and rules. Thus, nonterminals of the type $[[AB]C]$ are "collapsed" into $[AC]$. Intuitively, this operation is justified by the observation that the rule of an "intermediate" nonterminal of the type $[[AB]C]$ means that, in G, $A \stackrel{*}{\Rightarrow} x_1$, a suffix of some string $x \in L$, $B \stackrel{*}{\Rightarrow} y$ belonging to L, and $C \stackrel{*}{\Rightarrow} z_1$, a prefix of some $z \in L$. By this way, the number of possible new nonterminals is bounded and the construction of $\widehat{G}$ terminates when no new nonterminals and rules are generated. Fig. 12 gives an idea of how the sequence G_1, G_2, G_3 is built.

Notice also that the details of the construction involve the production of so-called *compound* nonterminals of the type $\{[AB], [CD], E\}$, i.e., collection of "boundary nonterminals". This is due to the need to iteratively apply a normalization

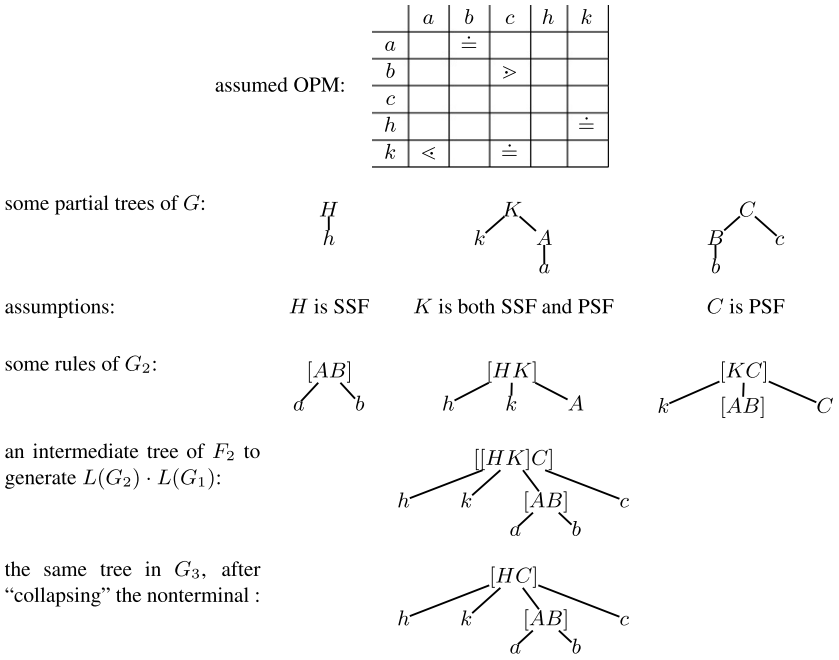

Fig. 12. Construction of grammar for $L \cdot L \cdot L$. From top to bottom: assumed precedence relations, some rules of G, some rules of G_2, an "intermediate" rule to generate $L(G_2) \cdot L(G_1)$, the same rule after "collapsing" the nonterminal.

procedure that eliminates repeated right-hand sides. For instance, suppose that during the process the following rules are built:

$$[AB] \to \alpha \mid \beta, \qquad [CD] \to \alpha \mid \gamma, \qquad E \to \alpha \mid \delta$$

where $A, B, C, D, E \in V_N$, and we recall that the nonterminals of the form of a pair $[AB]$ are created by the concatenation algorithm. Then, elimination of repeated r.h.s. produces a normalized homogeneous grammar containing the rules:

$$\big\{[AB], [CD], E\big\} \to \alpha, \qquad \big\{[AB]\big\} \to \beta, \qquad \big\{[CD]\big\} \to \gamma, \qquad \{E\} \to \delta.$$

We observe that it is possible to prove that the closure of FL under the boolean operations, concatenation and Kleene star, implies that certain subfamilies of FL are closed under the same set of operations: the cases of regular languages and of visibly pushdown languages over the same partitioned alphabet are obvious.

4.5. Floyd grammars with regular expressions

The preceding closure properties make it possible and interesting to consider regular expressions over FL defined by precedence compatible grammars. We briefly elaborate on two aspects. First, we enrich Statement 2, then we extend FG to allow for regular expressions and augmented regular expressions in r.h.s., yet preserving context-freeness and conflict-freeness (i.e., determinism).

Regular languages with prescribed precedences. For regular languages, we have already observed that their standard Chomsky grammar of type 3, say right-linear, is a very special FG containing only $\lessdot$ relations. Let $R \subseteq \Sigma^*$ be a regular language and let M be any precedence matrix. A more interesting question is whether it is possible to find an FG that generates R, with M as OPM. The answer is positive under rather general hypotheses.

Corollary 17. *Let M be any total (i.e., no element is null) $\doteq$-acyclic matrix over Σ. Then the family of regular languages over Σ is (strictly) included in the family of languages generated by grammars in the family $C_{M,\doteq}$ ($\doteq$-acyclic, precedence compatible FG).*

Proof. Let R be defined by a regular expression. To construct an FG with the given matrix M, we analyze the regular expression starting from the atomic subexpressions. Anytime two subexpressions are combined by union or concatenation respectively, the constructions of Statement 11 and Theorem 15 respectively produce a corresponding grammar, compatible with M. Similarly, anytime a subexpression is under star, the construction of Theorem 16 produces the corresponding grammar. □

Table 1
Closure properties of language families

	Boolean (all)	Concatenation	Kleene star	Reversal	Prefix	Suffix	Reference
Regular	YES	YES	YES	YES	YES	YES	
Deterministic CF (DCF)	NO	NO	NO	NO	YES	NO	[27]
Balanced (BALAN)	YES	NO	NO	YES	NO	NO	[12]
Visibly pushdown (VPL)	YES	YES	YES	YES	YES	YES	[3]
H-sync. Height-deterministic (HDPDL)	YES	NO	NO	NO	YES	NO	[16]
Realtime HDPDL (HRDPDL)	YES	NO	NO	NO	YES	NO	[16]
Floyd grammar (FL)	YES	YES	YES	YES	YES	YES	here

This result gives a procedure, based on the previous algorithms, for constructing from a regular expression, a Floyd grammar with the specified precedences. In particular, when the assigned precedences correspond to the left-linear (or right-linear) structure, the procedure returns a left-linear (or right-linear) grammar. When the precedences are those of a visibly pushdown language, the procedure returns a grammar with the specified partition of the alphabet into call, return and internal symbols.

We are not aware of any comparable method for constructing a deterministic context-free grammar that imposes a prescribed syntactic structures to the sentences specified by a regular expression.

Floyd grammars extended with (augmented) regular expressions. CF grammars that contain the regular operators in the rule r.h.s. are usually called *extended Backus normal form* (EBNF) grammars and are frequently used in language reference manuals because of their readability and concision. For FG the EBNF form has never been considered because the needed closure properties were not known; now we informally describe this new possibility. It is safe to introduce EBNF rules into Floyd grammars, since we know that they do not cause precedence conflicts. Although from Theorem 16 the closure w.r.t. Kleene star requires the additional hypothesis of an $\doteq$-acyclic OPM, the lack of conflicts in the OPM allows for a natural extension of parsing algorithms to this – unusual – case too.

Actually there is more than that: in the rule r.h.s. we can use, besides the regular operators, intersection and set difference (in particular complement with respect to the maxgrammar language). Consider an FG that uses in the r.h.s. such *augmented* regular expressions over precedence-compatible languages. Then, by previous closure results, the language generated is a precedence-compatible FL.

In contrast, the introduction of complement and intersection into CF grammar rules produces more powerful classes of grammars (see for instance [25] where they are named *boolean grammars*), which generate languages that may not be context-free.

Lastly, we compare the behavior of EBNF grammars with respect to determinism, in the cases of FG and of LR(k) and LL(k) grammars. When regular expressions are applied to deterministic CF languages, the resulting language is in general non-deterministic. In parsing theory (see, e.g., [26]) special cases and sufficient conditions are known and used to check that the regular composition of LR(k) or LL(k) languages preserves the corresponding condition for determinism. For FG this is not necessary since any regular expression of precedence-consistent languages is precedence-consistent, therefore it is deterministic.

4.6. Comparison of closure properties

We summarize in Table 1 the closure properties of several families of languages that have been proposed in the context of studies on generalized parenthesis languages and on languages suited to infinite-state model checking. Regular and deterministic CF languages are the basic references of the comparison. The table does not specify the hypotheses for closures to hold, but of course the domain of the operations ought to be qualified, as we have discussed at length in earlier sections: thus, VPL are closed under union just for consistently partitioned alphabets, and FL just for precedence-compatible grammars, etc.

Also, the statements concerning reversal need polishing: the reversal of a VPL is not in the same class as the original language, with respect to the alphabet partition (and similarly for the reversal of an FL with respect to precedence-compatibility classes).

The last column gives the main references; for the last row it suffices to observe that the language

$$L = \left\{ a^n b^n c a^m b^{2m} \;\middle|\; n \geqslant 0,\ m \geqslant 0 \right\}$$

is in HRDPDL, but its self-concatenation $L \cdot L$ includes strings such as $a^n b^n c a^m b^m c$, which cannot be deterministically parsed.

Lastly, we do not have to discuss closure properties for alphabetical homomorphisms because they are obvious: if a morphism preserves conflict-freeness (or the VPA alphabet partition) then the transformed language is in FL (or in VPL).

5. Conclusions and further developments

We have argued that the old operator precedence grammars, which we call Floyd grammars to honor their inventor, may play an important, unifying role amongst the language families recently introduced by several researchers with the

motivation of extending typical closure properties of regular languages to wider classes of languages. FL are the largest – known to us – family of deterministic languages that enjoy closure w.r.t. all basic algebraic operations. Furthermore their parsing algorithm is simpler than for more modern and widespread families such as LR(k); in particular, it is non-directional, in the sense that it is not necessary to perform the analysis from left to right. This property can be exploited in several ways as we point out later. Thus we believe that FL currently offer the best compromise between expressiveness and automatic analyzability of a language formalism.

Our results can be, and partially already have been, enhanced along several directions. First, other important properties of VPL can be transferred to, or compared with, FL. In particular we realized that, somewhat surprisingly, FG lacked their "automata counterpart": although the deterministic shift-reduce parsing algorithms used for LR(k) grammars can obviously be applied to FG, they refer to more general classes of DPDA which hinders a full exploitation of the nice properties of FG. We recently filled up this theoretical hole and defined a new class of pushdown automata [28], named Floyd automata (FA), that perfectly match FG, in the same way as VPA match VPL. FA nicely lie between VPA and DPDA. The new class of automata allows, for instance, to extend their application to languages of infinite words by exploiting quite naturally Buchi or Muller approaches.

It is also interesting to characterize FL in terms of logic formulas, as it was done for VPL; this feature is of particular relevance to applications in the field of model checking.

Looking for other more comprehensive language families that share all closure properties enjoyed by regular languages, VPL, and FL, could also produce further progress.

In a different direction, it is interesting to transfer to VPL a rather surprising invariance property of FG. We recall the definition of *non-counting context-free grammar* (NC) [29], which extends the notion of NC regular language of McNaughton and Papert [10]. A CF grammar G is NC if the parenthesized language $L(\widetilde{G})$ satisfies the following condition: $\exists n > 0$: $\forall x, v, w, \underline{v}, y \in \Sigma^*$, where w and $vw\underline{v}$ are well parenthesized, and $\forall m \geqslant 0$, $xv^n w\underline{v}^n y \in \widetilde{L}$ if, and only if, $xv^{n+m} w\underline{v}^{n+m} y \in \widetilde{L}$. In general, two equivalent CF grammars may differ with respect to the NC property, but if an FG is NC, then all equivalent FG are NC [11]. Consider now, for a VPL $L \subseteq \Sigma^*$, two equivalent VPA recognizers, which may differ with respect to the 3-partition of the letters. The two corresponding FG (as from Theorem 5) may differ in precedence relations, but they are either both NC, or both counting.

FL also promise to enrich the application fields of VPL beyond the original sectors of mark-up languages and model checking. In fact VPL, BALAN, and other related classes belong to the family of "explicitly parenthesized languages" (they are "visibly pushdown"); thus, the parsing of their strings is trivial since their syntax trees are already embedded in the parenthesization; this property, however, makes them unsuitable to deal with programming and *a fortiori* natural languages where the syntactic structure of the strings is implicit and must be obtained by more or less sophisticated parsing algorithms. On the contrary, FL have been invented just with the goal of efficient, deterministic parsing, yet they retain the property of explicitly parenthesized languages – which is not shared by general deterministic languages: that the position of the "handle" (i.e., the r.h.s. of a rule) can be uniquely identified in a text independently from its external context. This makes FL better suited than general DPDL for parallel and/or incremental parsing, where the possibility of (re)starting the analysis at any point of the input string is fundamental. This approach too is the object of further ongoing research, in terms, e.g., of parallel FA.

Finally, a typical technical problem that often arises when applying verification techniques, is the combinatorial explosion that leads to theoretically intractable complexity. FL are no exception in this respect since the core of most verification techniques is the "determinization" of some formalism. For instance, transforming a non-deterministic VPA into a deterministic one may cause the number of states to grow from n to 2^{n^2}; similarly and not surprisingly, the decision procedures for FG are based on the elimination of repeated right-hand sides, which requires a nonterminal alphabet that is the power set of the original one [5]. Perhaps not surprisingly, our procedure to derive an FG from a VPA builds a nonterminal alphabet that is the set of pairs of VPA states. On the other hand, despite this theoretical hurdle, decades of intense researches in the field of model checking have produced plenty of heuristic and abstraction techniques that permit to handle systems with billions of potential states. Thus, it seems worth to investigate whether similar or new techniques may produce comparable results within VPL, FL and related families.

Acknowledgments

We thank Christof Loeding and Wolfgang Thomas for references to early related research, Jean Berstel and Didier Caucal for comments on previous versions of this paper, and Violetta Lonati and Matteo Pradella for insightful discussions. We are grateful to the anonymous referees for their careful revisions.

Appendix A

For the reader convenience the acronyms (dropping the "s" when writing plurals) of the numerous devices and language families are collected in Table 2:

Table 2
Acronyms.

BALAN	balanced grammar/language
CF, PDA	context-free, pushdown automaton
DCF or DPDA	deterministic CF, deterministic pushdown automaton
FG, FL, FA	Floyd grammar/language/automaton
HDPDA, HDPDL	height-deterministic deterministic pushdown automaton/language
HPDA, HPDL	height-deterministic pushdown automaton/language
HRDPDA, HRDPD	height-deterministic realtime deterministic pushdown automaton/language
NC	non-counting context-free grammar
OG	operator grammar
OPM	operator precedence matrix
REG	regular language
RDPDA, RDPDL	realtime deterministic pushdown automaton/language
RPDA, RPDL	realtime pushdown automaton/language
VPA, VPL	visibly pushdown automaton/language

Table 3
Rules of the axiom.

Case	Transitions	Rules
$S \to Yc_0X$	$\delta(q_i, c_0) \ni (p_j, Z_U)$	$S \to \langle q_0, q_i\rangle c_0 \langle p_j, p_f\rangle$, $\forall p_f \in Q_F$
$S \to Y$		$S \to \langle q_0, q_f\rangle$, $\forall q_f \in Q_F$
$S \to Yc_0$	$\delta(q_i, c_0) \ni (p_f, Z_U)$, $p_f \in Q_F$	$S \to \langle q_0, q_i\rangle c_0$
$S \to c_0X$	$\delta(q_0, c_0) \ni (p_j, Z_U)$	$S \to c_0 \langle p_j, p_f\rangle$, $\forall p_f \in Q_F$
$S \to c_0$	$\delta(q_0, c_0) \ni (p_f, Z_U)$, $p_f \in Q_F$	$S \to c_0$

Table 4
Rules of nonterminals of class Y (deriving the maximal prefix ending with empty stack).

Case	Transitions	Rules
$Y \to s$	$\delta(q_0, s) \ni q_i$	$\langle q_0, q_i\rangle \to s$
$Y \to r$	$\delta(q_0, r, \bot) \ni q_i$	$\langle q_0, q_i\rangle \to r$
$Y \to Ys$	$\delta(q_i, s) \ni q_j$	$\langle q_0, q_j\rangle \to \langle q_0, q_i\rangle s$
$Y \to Yr$	$\delta(q_i, r, \bot) \ni q_j$	$\langle q_0, q_j\rangle \to \langle q_0, q_i\rangle r$
$Y \to cBr$	$\delta(q_0, c) \ni (q_t, Z)$, $\delta(q_k, r, Z) \ni q_h$	$\langle q_0, q_h\rangle \to c\langle q_t, Z, q_k\rangle r$
$Y \to cr$	$\delta(q_0, c) \ni (q_t, Z)$, $\delta(q_t, r, Z) \ni q_h$	$\langle q_0, q_h\rangle \to cr$
$Y \to YcBr$	$\delta(q_i, c) \ni (q_j, Z)$, $\delta(q_m, r, Z) \ni q_n$	$\langle q_0, q_n\rangle \to \langle q_0, q_i\rangle c\langle q_j, Z, q_m\rangle r$
$Y \to Ycr$	$\delta(q_i, c) \ni (q_j, Z)$, $\delta(q_j, r, Z) \ni q_n$	$\langle q_0, q_n\rangle \to \langle q_0, q_i\rangle cr$

Table 5
Rules for nonterminals of classes B_1 and B_2, generating well-balanced strings. (The case B_2 just differs from B_1 in that the state set is Q_p instead of Q_q.)

Case	Transitions	Rules
$B_1 \to B_1cB_1r$	$\delta(q_i, c) \ni (q_j, Z)$, $\delta(q_m, r, Z) \ni q_n$	$\langle q, q_n\rangle \to \langle q, q_i\rangle c\langle q_j, Z, q_m\rangle r$, $\forall q \in Q_q$
$B_1 \to B_1cr$	$\delta(q_i, c) \ni (q_j, Z)$, $\delta(q_j, r, Z) \ni q_n$	$\langle q, q_n\rangle \to \langle q, q_i\rangle cr$, $\forall q \in Q_q$
$B_1 \to cB_1r$	$\delta(q_i, c) \ni (q_j, Z)$, $\delta(q_m, r, Z) \ni q_n$	$\langle q_i, q_n\rangle \to c\langle q_j, Z, q_n\rangle r$
$B_1 \to cr$	$\delta(q_i, c) \ni (q_j, Z)$, $\delta(q_j, r, Z) \ni q_n$	$\langle q_i, q_n\rangle \to cr$
$B_1 \to B_1cB_1r$	$\delta(q_i, c) \ni (q_j, Z)$, $\delta(q_m, r, Z) \ni q_n$	$\langle q_i, W, q_n\rangle \to \langle q, q_i\rangle c\langle q_j, Z, q_m\rangle r$, $\forall q \in Q_q$, $W \in \Gamma$
$B_1 \to cB_1r$	$\delta(q_i, c) \ni (q_j, Z)$, $\delta(q_m, r, Z) \ni q_n$	$\langle q, W, q_n\rangle \to c\langle q_j, Z, q_m\rangle r$, $\forall q \in Q_q$, $W \in \Gamma$
$B_1 \to B_1cr$	$\delta(q_i, c) \ni (q_j, Z)$, $\delta(q_j, r, Z) \ni q_n$	$\langle q, W, q_n\rangle \to \langle q, q_i\rangle cr$, $\forall q \in Q_q$, $W \in \Gamma$
$B_1 \to B_1s$	$\delta(q_h, s) \ni q_m$	$\langle q, W, q_m\rangle \to \langle q, q_h\rangle s$, $\forall q \in Q_q$, $W \in \Gamma$
$B_1 \to s$	$\delta(q_j, s) \ni q_m$	$\langle q_j, W, q_m\rangle \to s$, $\forall W \in \Gamma$

Appendix B

B.1. Complements to the proof of VPL inclusion within FL, i.e., Theorem 5

B.1.1. Grammar construction

Tables 3, 4, 5, and 6 provide a complete definition of G's rules on the basis of the terminology of Section 3.5. The rules are keyed to the factorization of Lemma 4. The scheme of a sample syntax tree produced by the grammar, for a string factorized as in Lemma 4, was shown in Fig. 3. Also, recall that the state set Q is partitioned into the disjoint sets $\{q_0\} \cup Q_q \cup Q_p$.

In the tables, calls, returns and internals are respectively denoted c, r and s; Z, W are stack symbols different from $\bot$. Notice that the constructed grammar may be not reduced (i.e. some nonterminal may be unreachable from the axiom or it may not derive any terminal string). In that case the useless nonterminals and rules can be removed by well-known

Table 6
Rules for nonterminals of class X (deriving the suffix containing unmatched calls).

Case	Transitions	Rules
$X \to cX$	$\delta(p_i, c) \ni (p_j, Z_U)$	$\langle p_i, p_f\rangle \to c\langle p_j, p_f\rangle$, $\forall p_f \in Q_F$
$X \to c$	$\delta(p_i, c) \ni (p_f, Z_U)$, $p_f \in Q_F$	$\langle p_i, p_f\rangle \to c$
$X \to BcX$	$\delta(p_j, c) \ni (p_h, Z_U)$	$\langle p, p_f\rangle \to \langle p, p_j\rangle c\langle p_h, p_f\rangle$, $\forall p_f \in Q_F$, $p \in Q_p$
$X \to BC$	$\delta(p_j, c) \ni (p_f, Z_U)$, $p_f \in Q_F$	$\langle p, p_f\rangle \to \langle p, p_j\rangle c$
$X \to BcBr$	$\delta(p_i, c) \ni (p_j, Z)$, $\delta(p_m, r, Z) \ni p_n$	$\langle p, p_f\rangle \to \langle p, p_i\rangle c\langle p_j, Z, p_m\rangle r$, $\forall p \in Q_p$, $p_f \in Q_F$
$X \to cBr$	$\delta(p_i, c) \ni (p_j, Z)$, $\delta(p_m, r, Z) \ni p_n$	$\langle p_i, p_f\rangle \to c\langle p_j, Z, p_n\rangle r$, $p_f \in Q_F$
$X \to Bcr$	$\delta(p_i, c) \ni (p_j, Z)$, $\delta(p_j, r, Z) \ni p_n$	$\langle p, p_f\rangle \to \langle p, p_i\rangle cr$, $\forall p \in Q_p$, $\forall p_f \in Q_F$
$X \to cr$	$\delta(p_i, c) \ni (p_j, Z)$, $\delta(p_j, r, Z) \ni p_n$	$\langle p_i, p_f\rangle \to cr$, $\forall p \in Q_p$, $p_f \in Q_F$
$X \to Bs$	$\delta(p_j, s) \ni p_f$, $p_f \in Q_F$	$\langle p, p_f\rangle \to \langle p, p_j\rangle s$, $\forall p \in Q_p$
$X \to s$	$\delta(p_j, s) \ni p_f$, $p_f \in Q_F$	$\langle p, p_f\rangle \to s$

algorithms (e.g. in [27]). Similarly, if repeated r.h.s. are generated, a transformation to re-normalize the grammar has to be applied.

B.1.2. G is a Floyd grammar

By construction all the rules are in operator form. To verify that the operator precedence matrix M is conflict-free, it suffices to compute the relevant terminal sets and the matrix entries using the previous definitions. It should be enough to show one case.

For the rule $\langle q_0, q_n\rangle \to \langle q_0, q_i\rangle c\langle q_j, Z, q_m\rangle r$ the set $\mathcal{R}_G(\langle q_0, q_i\rangle) \subseteq \Sigma_i \cup \Sigma_r$ produces the relations $s \gtrdot c$, $r \gtrdot c$.

The sets $\mathcal{L}_G(\langle q_j, Z, q_m\rangle) \subseteq \Sigma_i \cup \Sigma_c$, $\mathcal{R}_G(\langle q_j, Z, q_m\rangle) \subseteq \Sigma_i \cup \Sigma_r$ determine $c \lessdot c$, $c \lessdot s$ and $s \gtrdot r$, $r \gtrdot r$; the right part of the rule gives $c \doteq r$. Thus we obtain a conflict-free matrix $M \subseteq M_T$ where M_T is the total matrix that was shown in Fig. 4.

B.1.3. Proof of the equivalence $L(G) = L(\mathcal{A})$

It is obtained by a fairly natural induction showing the double implication between computations and derivations. It is structured into several "macro-steps" mirroring the factorization introduced in Lemma 4. We develop in detail only a sample of the various cases, since the others are similar.

1. $(q_i, \bot) \overset{*}{\underset{x}{\mapsto}} (q_j, \bot) \Leftrightarrow \langle q_i, q_j\rangle \overset{*}{\Rightarrow} x$, $x \in (\Sigma_r \cup \Sigma_i)^*$.
2. $(q_i, \sigma) \overset{*}{\underset{x}{\mapsto}} (q_j, \sigma) \Leftrightarrow \langle q_i, q_j\rangle \overset{*}{\Rightarrow} x$, $x \in \Sigma^*$ and well balanced.
3. $(p_i, \sigma) \overset{*}{\underset{x}{\mapsto}} (p_j, \sigma) \Leftrightarrow \langle p_i, p_j\rangle \overset{*}{\Rightarrow} x$, $x \in \Sigma^*$ and well balanced.
4. $(p_i, \bot Z_U^k) \overset{*}{\underset{c^n}{\mapsto}} (p_j, \bot Z_U^{k+n}) \Leftrightarrow \langle p_i, p_j\rangle \overset{*}{\Rightarrow} c^n$.
5. $\forall \gamma \in \Gamma^*, Z$, $(p_i, \bot\gamma Z) \overset{*}{\underset{w}{\mapsto}} (p_j, \bot\gamma Z)$ (without ever popping Z) $\Leftrightarrow \langle p_i, Z, p_j\rangle \overset{*}{\Rightarrow} w$, where w is a well-balanced string.

   Induction base:
   (a) $\delta(p_i, c) \ni (p_k, Z) \wedge \delta(p_k, r, Z) \ni p_j \Leftrightarrow \exists W\colon\ \langle p_i, W, p_j\rangle \to cr$.
   (b) $\delta(p_i, s) \ni p_j \Leftrightarrow \exists W\colon\ \langle p_i, W, p_j\rangle \to s$.

   From the inductive hypotheses:
   (a) $(p_i, \bot\gamma W) \overset{*}{\underset{x}{\mapsto}} (p_h, \bot\gamma W) \Leftrightarrow \langle p_i, p_h\rangle \overset{*}{\Rightarrow} x$, $x \in \Sigma^*$,
   (b) $(p_h, \bot\gamma W) \underset{c}{\mapsto} (p_t, \bot\gamma W Z)$,
   (c) $(p_t, \bot\gamma W Z) \overset{*}{\underset{w_1}{\mapsto}} (p_r, \bot\gamma W Z) \Leftrightarrow \langle p_t, Z, p_r\rangle \overset{*}{\Rightarrow} w_1$,
   (d) $(p_r, \bot\gamma W Z) \underset{r}{\mapsto} (p_j, \bot\gamma W)$,

we derive:

$$(p_i, \bot\gamma W) \overset{*}{\underset{w}{\mapsto}} (p_j, \bot\gamma W) \quad \Leftrightarrow \quad \langle p_i, W, p_j\rangle \overset{*}{\Rightarrow} w, \quad w = xcw_1r. \tag{15}$$

Special cases, such as $x = \varepsilon$ and many others, can be similarly treated.

N.B. Each inductive proof of the various assertions may exploit other assertions in the inductive steps. For instance the inductive hypothesis (a) above is based on assertion 3.

B.2. Complements to the proof that an FG with VP partitioned matrix generates a VPL, i.e., Theorem 7

The complete definition of $\mathcal{A}$'s transition function is given in Table 7. Notice that the derivations $S \overset{*}{\Rightarrow} A\alpha$ needed in section 1 of the table can be effectively computed.

Table 7
Transition relation δ of the VPA $\mathcal{A}$ accepting $L(G)$.

	Rules	δ
1	$A \to s$	$\delta(q_0, s) \ni A$
	$A \to r$, such that $S \overset{*}{\Rightarrow} A\alpha$	$\delta(q_0, r, \bot) \ni A$
2	$A \to s$	$\delta(p, s) \ni A$
	$A \to Bs$	$\delta(B, s) \ni A$
	$A \to Br$	$\delta(B, r, \bot) \ni A$
3	$A \to cB$	$\delta(p, c) \ni (p, Z_U)$
	$S \to BcC$	$\delta(B, c) \ni (p, Z_U)$
4	$S \to BcCr$	$\delta(B, c) \ni (p, \langle B, c, C\rangle)$
		$\delta(C, r, \langle B, c, C\rangle) \ni q_F$
	$S \to s$	$\delta(q_0, s) \ni q_F$
	$S \to c$	$\delta(q_0, c) \ni (q_F, Z_U)$
	$S \to r$	$\delta(q_0, r, \bot) \ni q_F$
	$A \to BcCr$	$\delta(B, c) \ni (p, \langle B, c, C\rangle)$
		$\delta(p, r, \langle B, c, C\rangle) \ni A$
	$A \to Bcr$	$\delta(B, c) \ni (p, \langle B, c, -\rangle)$
		$\delta(p, r, \langle B, c, -\rangle) \ni A$
5	$A \to cBr$	$\delta(p, c) \ni (p, \langle -, c, B\rangle)$
		$\delta(B, r, \langle -, c, B\rangle) \ni A$
	$A \to cr$	$\delta(p, c) \ni (p, \langle -, c, -\rangle)$
		$\delta(p, r, \langle -, c, -\rangle) \ni A$

The proof of the equivalence $L(\mathcal{A}) = L(G)$ somewhat mirrors the equivalence proof of Theorem 5. For instance, from section 2 of Table 7 the following lemma immediately descends:

$$A \overset{*}{\Rightarrow} w, \quad w \in (\Sigma_i \cup \Sigma_r)^* \quad \Leftrightarrow \quad \exists \sigma \in \left(\Gamma \setminus \{\bot\}\right)^*,\ t \in Q \quad \text{such that} \quad (t, \bot\sigma) \overset{*}{\underset{w}{\mapsto}} (A, \bot\sigma).$$

Similarly, the lemma

$$A \overset{*}{\Rightarrow} w, \quad w \text{ well balanced} \quad \Leftrightarrow \quad \exists \sigma \in \left(\Gamma \setminus \{\bot\}\right)^*,\ t \in Q \quad \text{such that} \quad (t, \bot\sigma) \overset{*}{\underset{w}{\mapsto}} (A, \bot\sigma)$$

can be proved by a natural induction, taking as the basis the cases $A \to cr$ and $A \to s$, and then exploiting for the induction steps sections 2, 4, and 5 of Table 7. Further details of the proof are omitted as fairly obvious.

B.3. Proof of concatenation closure of FG, i.e., Theorem 15

We assume that G_1 and G_2 are in homogeneous normal form. First we present the algorithm that constructs a homogeneous FG $G = (V_N, \Sigma, P, S)$ such that $OPM(G) \supseteq OPM(G_1) \cup OPM(G_2)$. Then, by means of a series of lemmas, we prove that $L(G) = L(G_1) \cdot L(G_2)$.

B.3.1. Algorithm building grammar G of concatenation

For simplicity, we assume that $M = OPM(G_1) \cup OPM(G_2)$ is a total matrix. This does not affect generality, because if, at any step, the following algorithm checks the precedence relation between two letters a and b, and $M_{ab} = \emptyset$, we can *arbitrarily* assign to M_{ab} a value, thus obtaining a matrix compatible with M.

Two types of nonterminals occur in G: the nonterminals other than the axioms of G_1 and G_2, and new nonterminals named by certain pairs $[AB]$ with $A \in V_{N_1}$ and $B \in V_{N_2}$.

The algorithm may produce non-invertible grammars and may create useless rules. Thus, a final step will transform the grammar into the desired normal form at the end.

Initialization. Initialize G with $V_N = V_{N_1} \cup V_{N_2} \setminus \{S_1, S_2\}$ and P with the rules of P_1 and P_2 that do not contain S_1 or S_2. Then add to G the rules created using the following two initial cases.

1. *Initial case* $\doteq$. For every $A \to \alpha \in P_1$, $B \to \beta \in P_2$ such that A is a SSF and B is a PSF and $\alpha \doteq \beta$, and one of the following mutually exclusive conditions holds:
   - $\alpha = \alpha' a$, $\beta = b\beta'$, schematized in Fig. 8,
   - $\alpha = \alpha' aN$, $\beta = b\beta'$, $\mathcal{R}(N) \gtrdot b$, $N \in V_{N_1}$, schematized in Fig. 9,
   - $\alpha = \alpha' a$, $\beta = Nb\beta'$, $a \lessdot \mathcal{L}(N)$, $N \in V_{N_2}$ (the symmetrical of the preceding case),

   add to V_N the nonterminal $[AB]$ and to P the rule $[AB] \to \alpha\beta$. $\mathcal{L}_G([AB]) = \mathcal{L}_{G_1}(A)$ and $\mathcal{R}_G([AB]) = \mathcal{R}_{G_2}(B)$, therefore no new precedence relations are added to M by the new rules, because, thanks to the homogeneous form of G_1 and G_2, $\mathcal{L}(\beta) = \mathcal{L}(B)$ and $\mathcal{R}(\alpha) = \mathcal{R}(A)$.

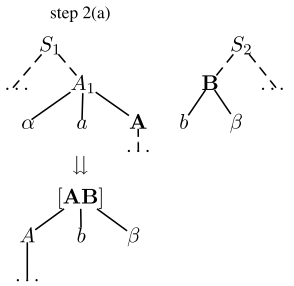

Fig. 13. Rules created by Case 2(a).

2. *Initial cases* $\lessdot$ and $\gtrdot$.
   (a) *Case* $\lessdot$. For every derivation

$$S_1 \overset{*}{\Rightarrow} \xi A_1 \Rightarrow \xi\alpha a A \quad \text{with } \xi \neq \varepsilon, \quad \text{and} \quad S_2 \overset{*}{\Rightarrow} B\zeta \Rightarrow b\beta\zeta$$

   such that $\mathcal{R}(A) \gtrdot b$ and $a \lessdot b$, add to V_N the nonterminal $[AB]$ and to P the rule $[AB] \to Ab\beta$. This case is schematized in Fig. 13.
   Remark: from the homogeneous form, for any rules $A \to \alpha_1 \mid \alpha_2$, it is $\mathcal{R}(\alpha_1) = \mathcal{R}(\alpha_2) = \mathcal{R}(A)$ and precedence relations are unaffected.
   (b) *Case* $\gtrdot$. This and the previous case are symmetrical. For every derivations

$$S_1 \overset{*}{\Rightarrow} \zeta A \Rightarrow \zeta\alpha a \quad \text{and} \quad S_2 \overset{*}{\Rightarrow} B_1\xi \Rightarrow Bb\beta\xi \quad \text{with } \xi \neq \varepsilon$$

   such that $a \lessdot \mathcal{L}(B)$ and $a \gtrdot b$, add to V_N the nonterminal $[AB]$ and to P the rule $[AB] \to \alpha a B$.
   Remark: the precedence relations are unaffected.
   (c) For every pair of rules

$$S_1 \to A_1 \quad \text{and} \quad B \to b\beta$$

   such that B is a PSF and $\mathcal{R}(A_1) \gtrdot b$, add to G the rule $[A_1 B] \to A_1 b\beta$.
   Notice that this case can be seen as a subcase of (a), where the derivation is $S_1 \Rightarrow A_1$ and therefore the hidden $\perp$ plays the role of a.
   Remark: the precedence relations are unaffected.
   (d) Symmetrically, for every pair of rules

$$A \to \alpha a, \qquad S_2 \to B_1$$

   such that A is a SSF and $a \lessdot \mathcal{L}(B_1)$, add to G the rule $[AB_1] \to \alpha a B_1$.
   Remark: the precedence relations are unaffected.

After initialization. The following steps 3, 4, 5 are applied over and over, until no more rules are added to the grammar. In steps 3, 4, 5, the nonterminals A_1, A are SSF and the nonterminals B_1, B are PSF. In all cases the precedence relations are unaffected.

(3) *Case* $\doteq$. For every $A_1 \to \alpha a A \in P_1$ and $B_1 \to Bb\beta \in P_2$ (clearly it is $A_1 \neq S_1$ and $B_1 \neq S_2$ since the grammar is homogeneous), such that $a \doteq b$ and $[AB] \in V_N$, add to P the rule $[A_1 B_1] \to \alpha a [AB] b\beta$.
(4) *Case* $\gtrdot$. For every $A_1 \to \alpha a A \in P_1$ such that A_1 is a SSF, and for every $B \in V_{N_2}, b \in \Sigma$, such that $a \gtrdot b$ and $S_2 \overset{*}{\Rightarrow} B_1\zeta \Rightarrow Bb\beta\zeta$ and $[AB] \in V_N$, add the rule $[A_1 B] \to \alpha a [AB]$ to P.
(5) *Case* $\lessdot$. For every $B_1 \to Bb\beta \in P_2$ such that B is a PSF, for every $A \in V_{N_1}, a \in \Sigma$, such that $a \lessdot b$ and $S_1 \overset{*}{\Rightarrow} \xi A_1 \Rightarrow \xi\alpha a A$ and $[AB] \in V_N$, add the rule $[AB_1] \to [AB] b\beta$ to P.

See Fig. 14 for illustration.
At last, to create the rules for the axiom, apply the next step:

(6) *Case axiom.* Apply the following steps to every pair of rules $S_1 \to A_1$ and $S_2 \to B_1$.
   (a) If $[A_1 B_1] \in V_N$, add to P the rule $S \to [A_1 B_1]$.
   (b) If $A_1 \to \alpha a N$ with $N \in V_{N_1} \cup \{\varepsilon\}$ is in P_1 and $B_1 \to b\beta$ is in P_2, such that $a \gtrdot b$, then add to P the rules $[A_1 B_1] \to A_1 b\beta$ and $S \to [A_1 B_1]$.
   (c) If $A_1 \to \alpha a \in P_1$ and $B_1 \to Nb\beta \in P_2$ with $N \in V_{N_2} \cup \{\varepsilon\}$, such that $a \lessdot b$, then add to P the rules $[A_1 B_1] \to \alpha a B_1 \in P$ and $S \to [A_1 B_1]$.

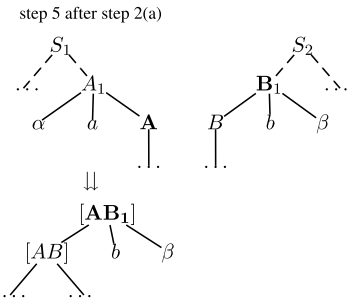

Fig. 14. Rules created by Case 5.

(d) If $A_1 \to \alpha a A$ is in P_1, $S_2 \to B$ is in P_2, and $[AB] \in V_N$, then add the rule $[A_1 S_2] \to \alpha a [AB]$; furthermore, if $S_1 \to A_1$ is in P_1, add the rule $S \to [A_1 S_2]$.
(e) If $A_1 \to \alpha a A$ is in P_1 and $[AS_2] \in V_N$, then add the rule $[A_1 S_2] \to \alpha a [AS_2]$.
(f) symmetrical to (d).
(g) symmetrical to (e).
Remark: the rules thus added to P do not modify the precedence matrix M.

It is obvious that the construction terminates, because there are finitely many possible new nonterminals of the form $[AB]$ and finitely many possible right-hand sides for the rules.

Finally, since each step of the construction does not induce new precedence relations, the grammar G is an FG with a matrix M compatible with M_1 and M_2.

From Statement 1, a structurally equivalent grammar in homogeneous normal form can be obtained. In particular, the grammar must be reduced by eliminating useless rules and repeated right-hand sides.

B.3.2. Proof that $L(G) = L(G_1) \cdot L(G_2)$

The inclusion $L(G) \subseteq L(G_1) \cdot L(G_2)$ follows easily from the construction, so we just have to prove the inclusion $L(G) \supseteq L(G_1) \cdot L(G_2)$. For that we need the next four lemmas.

Lemma 18. *It states two symmetrical properties*:

- *If there are derivations*

$$S_1 \Rightarrow A_1 \stackrel{*}{\Rightarrow} x, \qquad S_2 \stackrel{*}{\Rightarrow} y = b \cdot z, \quad b \in \Sigma$$

and $\mathcal{R}(A_1) \gtrdot b$, then G has a derivation

$$S \Rightarrow [A_1 B] \stackrel{*}{\Rightarrow} [A_1 B_k]\zeta \Rightarrow A_1 b \zeta \stackrel{*}{\Rightarrow} xy.$$

- *If there are derivations*

$$S_1 \stackrel{*}{\Rightarrow} x = z \cdot a, \quad a \in \Sigma, \qquad S_2 \Rightarrow B_1 \stackrel{*}{\Rightarrow} y$$

and $a \lessdot \mathcal{L}(B_1)$, then G has a derivation

$$S \Rightarrow [AB_1] \stackrel{*}{\Rightarrow} \xi [A_k B_1] \Rightarrow \xi a B_1 \stackrel{*}{\Rightarrow} xy.$$

Proof. For the first item: if $S_2 \Rightarrow B_1 \Rightarrow b\beta \stackrel{*}{\Rightarrow} y$ then by construction 2(c) and 6(a) $\exists [A_1 B_1] \to A_1 b\beta$ and $S \to [A_1 B_1]$. If $S_2 \stackrel{*}{\Rightarrow} B_k \xi \Rightarrow b\beta\xi \stackrel{*}{\Rightarrow} y$, then from 2.(a) $\exists [A_1 B_k] \to A_1 b\beta$ and for every $B_i \to B_j \beta_j$ in the derivation $S_2 \stackrel{*}{\Rightarrow} B_i \xi \Rightarrow B_j \beta_j \xi \stackrel{*}{\Rightarrow} y$ there exists by step 5 a rule $[A_1 B_i] \to [A_1 B_j]\beta_j$. Thus $S \stackrel{*}{\Rightarrow} x \cdot y$. For the second item the proof is symmetrical. □

Lemma 19. *It states two symmetrical properties*:

- *If there are derivations $S_1 \stackrel{*}{\Rightarrow} x_1 A_1 \Rightarrow x_1 \alpha a A \stackrel{*}{\Rightarrow} xw$, $A \stackrel{*}{\Rightarrow} w$, and $S_2 \stackrel{*}{\Rightarrow} By \Rightarrow b\beta y \stackrel{*}{\Rightarrow} bz \cdot y$ such that $\mathcal{R}(A) \gtrdot b$, $a \lessdot b$, then there exists a nonterminal $[AB]$ such that $[AB] \Rightarrow Ab\beta \stackrel{*}{\Rightarrow} w \cdot bz$.* (*Proof*: *by case* (2)(a).)
- *Symmetrically, if there are derivations $S_1 \stackrel{*}{\Rightarrow} xA \Rightarrow x\alpha a \stackrel{*}{\Rightarrow} x \cdot za$ with $a \in \Sigma$ and $S_2 \stackrel{*}{\Rightarrow} B_1 y_1 \Rightarrow Bb\beta y_1 \stackrel{*}{\Rightarrow} wy$ with $b \in \Sigma$, $B \stackrel{*}{\Rightarrow} w$ such that $a \lessdot \mathcal{L}(B)$, $a \gtrdot b$, then there exists a nonterminal $[AB]$ such that $[AB] \Rightarrow \alpha a B \stackrel{*}{\Rightarrow} zaw$.* (*Proof*: *by case* (2)(b).)

Similar to Lemma 19, the next lemma takes care of the condition $a \doteq b$.

Lemma 20. *It states two symmetrical properties*:

- *If there are derivations*

$$S_1 \overset{*}{\Rightarrow} x_1 A \Rightarrow x_1 \alpha a A_1 \overset{*}{\Rightarrow} x_1 v w$$

with $A_1 \overset{*}{\Rightarrow} w$, *and*

$$S_2 \overset{*}{\Rightarrow} By \Rightarrow b\beta y \overset{*}{\Rightarrow} bzy$$

such that $a \doteq b$ *and* $\mathcal{R}(A_1) \gtrdot b$, *then there exists a nonterminal* $[A\ B]$ *and a rule* $[A\ B] \to \alpha a A_1 b \beta$. (*Proof*: *by case* (1).)
- *If there are derivations*

$$S_1 \overset{*}{\Rightarrow} xA \Rightarrow x\alpha a \overset{*}{\Rightarrow} xva, \qquad S_2 \overset{*}{\Rightarrow} By_1 \Rightarrow B_1 b\beta y_1 \overset{*}{\Rightarrow} wzy_1$$

with $B_1 \overset{*}{\Rightarrow} w$, *such that* $a = b$ *and* $a \lessdot \mathcal{L}(B_1)$, *then there exists a nonterminal* $[A\ B]$ *and a rule* $[AB] \to \alpha a B_1 \beta$.

Finally, the following lemma synthesizes all previous results and leads to the thesis.

Lemma 21. *If there are derivations*

$$S_1 \overset{*}{\Rightarrow} xA, \quad A \overset{*}{\Rightarrow} w, \qquad S_2 \overset{*}{\Rightarrow} By, \quad B \overset{*}{\Rightarrow} z$$

then G has the derivation

$$S \overset{*}{\Rightarrow} x[AB]y \overset{*}{\Rightarrow} xw \cdot zy.$$

Proof. The proof is by induction. The initialization is covered by Lemmas 18, 19, 20, and we explain the induction step. Assume by inductive hypothesis that

$$S_1 \overset{*}{\Rightarrow} xA_1 \Rightarrow x\alpha aA \overset{*}{\Rightarrow} x\alpha a x_1 \quad \text{and} \quad S_2 \overset{*}{\Rightarrow} B_1 y \Rightarrow Bb\beta y \overset{*}{\Rightarrow} y_1 b\beta y$$

and G has the derivation $[AB] \overset{*}{\Rightarrow} x_1 y_1$. Three cases may occur.

1. $a \doteq b$. Then, the constructions at case (1) (initial case $\doteq$) and at case (3) of the algorithm ensure that G contains the rule $[A_1 B_1] \to \alpha a [AB] b \beta$.
2. $a \gtrdot b$. Similarly, by construction case (4).
3. $a \lessdot b$. Similarly, by construction case (5). □

The next example illustrates the construction.

Example 1. Consider the following FG

$$G_1 = \{S_1 \to A_0,\ A_0 \to aA_1,\ A_1 \to aA_1 b \mid ab\},$$
$$G_2 = \{S_2 \to B,\ B \to bc \mid Bbc\}.$$

The left/right terminal sets and the precedence matrix $OPM(G_1) \cup OPM(G_2)$ are:

	$\mathcal{R}$	$\mathcal{L}$
A_0	a, b	useless
A_1	b	useless
B	useless	b

$$M_{1,2} = \begin{array}{c|ccc} & a & b & c \\ \hline a & \lessdot & \doteq & \\ b & & \gtrdot & \doteq \\ c & & \gtrdot & \end{array}$$

We list the results of the steps.

$$V_N = \{A_0, A_1, B\}; \qquad P = \{A_0 \to aA_1,\ A_1 \to aA_1 b \mid ab,\ B \to bc \mid Bbc\}.$$

For initial case (1), $\doteq$:
A_0, A_1 are SSF, B is PSF. From the derivations $A_0 \Rightarrow aA_1$, $B \Rightarrow bc$ and from the relation $\mathcal{R}(A_1) = b \gtrdot b$, create the rule

$$[A_0 B] \to aA_1 bc.$$

Then no rule is created by the following steps:
(2)(a). initial case $\lessdot$; (2)(b). initial case $\gtrdot$; (2)(c).; (2)(d).; (3), case $\doteq$; (4), case $\gtrdot$.

For case (5), $\lessdot$, from $S_1 \to A_0$, $B \to Bbc$, from the relation $\perp \lessdot b$, and the existence of nonterminal $[A_0B]$, create the rule

$$[A_0B] \to [A_0B]bc.$$

Finally, for case (6):
From $S_1 \to A_0$, $S_2 \to B$

6(a). Since $[A_0B]$ exists, create the rule

$$S \to [A_0B].$$

The cases 6(b). and 6(c). are unproductive, and collecting all the rules, we obtain the grammar:

$$S \to [A_0B], \qquad [A_0B] \to aA_1bc, \qquad [A_0B] \to [A_0B]bc,$$
$$A_0 \to aA_1, \qquad A_1 \to aA_1b \mid ab, \qquad B \to bc \mid Bbc.$$

Grammar normalization deletes rules $A_0 \to aA_1$, $B \to bc \mid Bbc$. No repeated r.h.s. are present and the homogeneous condition is already met.

B.4. Proof of the closure of FL under Kleene star, i.e., Theorem 16

First we present the algorithm to build $\widehat{G}$, then we prove the equivalence $L(\widehat{G}) = L^*$.

B.4.1. Algorithm constructing the grammar of Kleene closure

The algorithm constructs a finite series of grammars $G = G_1, G_2, \ldots, G_I, \ldots$ with $L = L(G_1) \subset L(G_2) \subset \ldots$, eventually converging to $L^+ = L(\widehat{G})$.

Initialization. Let $G_1 = G = (V_N, \Sigma, P, S)$. Using the construction for concatenation, we build the grammar denoted by F_2 such that $L(F_2) = L^2(G_1)$.

Then we build the grammar G_2 generating the language $L(G_2) = L^2(G_1) \cup L(G_1)$, and cast it into homogeneous normal form. It is important that when applying normalization, the nonterminal names of G_2 are chosen so to preserve information on the original nonterminals.

Disciplined normalization. More precisely, in G_2, as well as in all subsequent grammars, the nonterminal alphabet is $V_{N,2} \subseteq \wp(V_N \times V_N \cup V_N)$. A nonterminal such as $\{[AB], [CD], E\}$ is termed *compound*, while a *simple* nonterminal has the form, say, $\{E\}$ or $\{[AB]\}$.

The intended meaning of such nonterminal names is clarified by the next example. Suppose that before normalization the grammar contained the rules:

$$[AB] \to \alpha \mid \beta, \qquad [CD] \to \alpha \mid \gamma, \qquad E \to \alpha \mid \delta$$

where $A, B, C, D, E \in V_N$, and we recall that the nonterminals of the form of a pair $[AB]$ are created by the concatenation algorithm. The normalized, invertible grammar then contains the rules:

$$\big\{[AB], [CD], E\big\} \to \alpha, \qquad \big\{[AB]\big\} \to \beta, \qquad \big\{[CD]\big\} \to \gamma, \qquad \{E\} \to \delta. \tag{16}$$

In addition, in order to preserve language equivalence, for any occurrence of a nonterminal N in a r.h.s., the alternatives must be added that have instead of N any compound nonterminal N' such that $N \subset N'$. Continuing the example, if rule $X \to \eta[AB]\vartheta$ is present, the normalized grammar contains the rules $\{X\} \to \eta\{[AB]\}\vartheta$ and $\{X\} \to \eta\{[AB], [CD], E\}\vartheta$.

We observe that, since G does not contain rules with identical r.h.s., a compound nonterminal $N \in V_{N,2}$ may not contain more than one singleton, i.e., $|N \cap V_N| \leqslant 1$.

The next lemmas are straightforward consequences of disciplined normalization.

Lemma 22. *The set of strings generated by grammar G_2 starting from a compound nonterminal N is*

$$L_{G_2}(N) = \bigcap_{A,B \in V_N \wedge [AB] \in N \wedge C \in N} \big\{L_G(A) \cdot L_G(B), L_G(C)\big\}.$$

We observe that the language generated by G_2 starting from a simple nonterminal, say $\{A\}$, may differ from the language generated by A in G:

$$L_{G_2}\big(\{A\}\big) \subseteq L_G(A)$$

if A occurs also in some compound nonterminal. In other words the above change in the nonterminal and rule sets of G_2 partitions the set of strings derivable from an original nonterminal according to the context where they can occur: if, in some place, the same string can be derived, say, both by $[AB]$ and by C, in the new version it will be derived by $\{[AB], C\}$. We will see that this separation or partition is necessary to distinguish between sentences that are part of a concatenation of two adjacent strings, say, x and y, of L in L^* and those that are generated by the original grammar G.

The next proposition relates the derivations in G and in G_2.

Lemma 23. *Let $N \in V_{N,2}$ and $[AB] \in N$. Then in G_2 nonterminal A is a SSF, nonterminal B is a PSF, and all strings derivable by $A \cdot B$ in G are derivable in G_2 by some nonterminal N containing $[AB]$.*

Iteration. Initially we set the iteration index I to 2.

1. Each iterative step computes the grammar generating language $L(G_I) \cdot L(G) \cup L(G_I)$. The grammar generating $L(G_I) \cdot L(G)$ is obtained by applying the algorithm for concatenation, with the following variant. Whenever the old algorithm would combine a nonterminal $N \in V_{N,I}$ and a nonterminal $H \in V_N$ into the pair $[NH]$ with associated rule $\{[NH]\} \to \alpha$, the new algorithm writes instead the rule $Q \to \alpha$ where

$$Q \text{ is } \big\{[AH], [CH] \,\big|\, \text{for some } B \in V_N, \text{ it is } [AB] \in N, \text{ and } C \in V_N \text{ is in } N\big\}. \tag{17}$$

   Clearly the mapping from the old to the new nonterminal names can be many-to-one.
   To illustrate, the combination of $N = \{[AB], [CD], E\}$ and of $N' = \{[AD], [CD], E\}$ with H gives rise to the same compound nonterminal $\{[AH], [CH], [EH]\}$. In such cases, we say that the nonterminals B, D have been *dropped by the renaming mapping*. On the other hand, the pair $[EH]$ has been created without dropping a nonterminal.
   Notice that this construction can introduce new recursive rules or derivations.
2. Normalize the resulting grammar into homogeneous normal form, adopting the same disciplined naming scheme (16) for nonterminals, as in the initialization step.
3. If G_I differs from G_{I-1} go to step 1. Otherwise set $\widehat{G} = G_I$ and halt.
   The algorithm clearly terminates since the nonterminal alphabet $V_{N,I}$ is a subset of $\wp(V_N \times V_N \cup V_N)$ and the length of the rule right-hand sides is bounded by the hypothesis that the OPM is $\doteq$-acyclic.

The first consequence of the above procedure is the next lemma.

Lemma 24. *Let $N \in V_{N,I}$ and $[AB] \in N$. Then for grammar G the following holds*:

- *A is a SSF and B is a PSF. Moreover either one of the following cases occurs*:
  - *nonterminal $[AB]$ was created as left part of a rule of type (17) without dropping a nonterminal*;
  - *nonterminal $[AB]$ was created in (17) dropping a nonterminal X, i.e., $[AB]$ comes from $[[AX]B]$ with $[AX] \in N' \in V_{N,I-1}$ and X a PSF.*

Thus, whereas in the original concatenation construction $[AB]$ denotes a nonterminal that generates two adjacent substrings that are, respectively, the suffix and the prefix of strings belonging to L, after the iteration steps an $[AB]$ belonging to a nonterminal N may also mean that $[AB]$ generates such substrings that encompass one or more whole sentences of L.

Notice also that the iterative step of the algorithm may produce derivations such as $N \overset{*}{\Rightarrow} \alpha N \beta$ with $[AB] \in N$, which, in the absence of the steps that collapse several $[[\ldots[AX]Y\ldots]B]$ into a single $[AB]$ would not be recursive. By this way strings belonging to L^*, and not only to a finite concatenation of L with itself, may be generated. On the other hand the separation between the various N_i guarantees that N_i derives only those strings that would be generated by everyone of its elements (Lemma 22); thus, wherever N_i is generated in a derivation of $\widehat{G}$, the strings it produces are compatible with their context since N_i is produced in sentential forms where at least one of its elements would have been generated by some intermediate grammar produced by the above iterative algorithm.

Finally, notice that a generic element $[AB]$, whether it appears in one or more N_i, may be the result of the collapsing of different previous elements. For instance, the following rules could be generated: $\{[AB]\} \to \alpha\gamma\beta$ as the result of collapsing $[[AX]B]$ into $[AB]$ from rules such as

$$A \to \alpha, \qquad X \to \gamma, \qquad B \to \beta, \quad \text{with } \alpha \doteq \gamma \doteq \beta,\ L(X) \subseteq L$$

and $\{[AB]\} \to \zeta.\delta.\eta$ as the result of collapsing $[[AY]B]$ into $[AB]$ from rules such as

$$A \to \zeta, \qquad Y \to \delta, \qquad B \to \eta, \quad \text{with } \zeta \doteq \delta \doteq \eta,\ L(Y) \subseteq L.$$

This again means that, if the singleton $\{[AB]\}$ is generated in some context, then it may generate strings that consist of an appropriate SSF derived from A and a PSF derived from B, that encompass strings of L derivable from X and Y respectively.

As a consequence we can assert the following lemma.

Lemma 25. *At every iteration step I, $L(G_I)$ is contained in L^*.*

To complete the proof we now state the converse lemma.

Lemma 26. *L^+ is contained in $L(\widehat{G})$.*

Proof. Consider a string $x = x_1 \cdot x_2 \dots x_n$, $x_i \in L$. We want to show that it is possible to parse it according to $\widehat{G}$. Without loss of generality, we assume that $M = OPM(G)$ is complete so that a bottom-up deterministic parsing of any string in Σ^* is always possible according to a maxgrammar (v.s. Statement 9) with $OPM = M$.

1) The first (possibly absent) parsing pass is performed on x by applying all reductions completely "contained within single x_i". More precisely, let $x_{i,p}$ and $x_{i,s}$ respectively denote a prefix and a suffix of x_i. Then it is $x_{i,p} \cdot \delta x_{i,s} \stackrel{*}{\Rightarrow} x_i$. This pass can be performed by using rules originally in G, up to some possible renaming of nonterminals due to the normalization.
2) The second pass may involve rules generated during the construction of G_2 (again, up to some renaming of nonterminals and splitting of their set of right-hand sides) in order to apply reductions in the "boundaries" between x_i and x_{i+1}, i.e., $\delta \stackrel{*}{\Rightarrow} x_{i,s} \cdot x_{i+1,p}$. This pass produces a string $\alpha = \alpha_2 \cdot \alpha_3 \dots \alpha_n$ such that $\alpha_i \stackrel{*}{\Rightarrow} x_{i-1,s} \cdot x_{i,p}$, and no further reductions of the previous types 1) and 2) are possible. This means that at least in one case $\alpha_i \doteq \alpha_{i+1}$ and, within α_i, the pair $\lessdot, \gtrdot$ never occurs. Notice that, as a particular case, the parsing could be completed after this phase if, e.g.,

$$S \stackrel{*}{\Rightarrow} Sx_n \stackrel{*}{\Rightarrow} Sx_{n-1} \cdot x_n \dots \stackrel{*}{\Rightarrow} x_1 \cdot x_2 \dots x_n.$$

3) If the above case does not occur, let us first consider the case
   3.1) $\dots \lessdot \alpha_{i,s} \doteq \alpha_{i+1,p} \gtrdot \dots$, where $\alpha_{i,s}$ is a suffix of α_i and $\alpha_{i+1,p}$ a prefix of α_{i+1}, i.e., only two consecutive α's are involved in the first reduction after phases 1 and 2.
   Also, $\alpha_{i,s}$ derives a suffix y_{i-1} of x_{i-1}, concatenated with a prefix z_i of x_i and $\alpha_{i+1,p}$ derives a suffix y_i of x_i concatenated with a prefix z_{i+1} of x_{i+1}.
   Since $x_{i-1} \cdot x_i \in L^2$, its parsing – by using the rules of G_2 – must produce some

$$\eta \alpha_{i,s} \zeta \stackrel{*}{\Rightarrow} z_{i-1} \cdot y_{i-1} \cdot z_i \cdot y_i,$$

   i.e., $\zeta \stackrel{*}{\Rightarrow} y_i$; notice that no further reduction $N \to \alpha_{i,s}\zeta_p$, ζ_p being a *proper* prefix of ζ, would be possible, since, otherwise, the same reduction could be applied even during the second pass above without involving $\alpha_{i+1,p}$.
   Consider now the parsing of $x_{i-1} \cdot x_i \cdot x_{i+1}$ according to G_3. On the one hand we have reductions (building bottom-up the derivations) $\eta \alpha_{i,s} \stackrel{*}{\Rightarrow} z_{i-1} \cdot y_{i-1} \cdot z_i$; on the other, $\alpha_{i+1,p} \vartheta \stackrel{*}{\Rightarrow} y_i \cdot z_{i+1} \cdot y_{i+1}$. Both reductions up to this point have been applied by using rules in G_2 (as usual, up to the renaming of some nonterminals).
   Notice also that $\alpha_{i+1,p}$ is of the type $\xi N_i \gamma$, with $N_i = \{[AB], \dots H, \dots\}$ and $\zeta = \xi A$ for some A belonging to some $[AB]$ belonging to N_i. In other words, N_i denotes the "boundary" where the parsing of x_i and that of x_{i+1} merge according to G_2.
   (a) If both ξ and γ are $\neq \varepsilon$, this implies $\xi \doteq \gamma$ and the existence in G of a rule $C \to B\gamma$ with B the second symbol of $[AB]$ (A being the same as in ξA). Thus, on the basis of the construction of G_3 the rule $[[N_h]C] \to \alpha_{i,s} \cdot \alpha_{i+1,p}$ subsequently transformed into $N_k \to \alpha_{i,s} \cdot \alpha_{i+1,p}$ is in G_3 and the corresponding reduction can be applied. It produces a string

$$\dots \chi N_k \phi \cdot \alpha_{i+2} \dots \Rightarrow \chi \alpha_{i,s} \alpha_{i+1,p} \phi . \alpha_{i+2} \stackrel{*}{\Rightarrow} x_{i-1,s} \cdot x_{i,p} \cdot x_{i,s} \cdot x_{i+1,p} . x_{i+1,s} \cdot x_{i+2,p}.$$

   For instance, with reference to Fig. 12, the parsing of a string *hkabc* within a context $\alpha - \beta$ such that $\alpha \lessdot h$ and $c \gtrdot \beta$, would build the derivation

$$\alpha[HC]\beta \Rightarrow \alpha \lessdot h \doteq k[AB] \doteq c \gtrdot \beta \Rightarrow \alpha \lessdot h \doteq k \lessdot a \doteq b \gtrdot \doteq c \gtrdot \beta.$$

   (b) If instead γ is ε (the case $\xi = \varepsilon$ is symmetric), this means that $B\vartheta \stackrel{*}{\Rightarrow} z_{i+1} \cdot y_{i+1}$ in G and $\xi \gtrdot \vartheta$. Thus a rule $N_h \to \alpha_{i,s} \xi A$ is in G_2 with $[HA_1]$ belonging to N_h and $A_1 \to \alpha A$ being a rule of G for some A_1, H. Therefore, again, a rule $[[N_h]B] \to \alpha_{i,s} \cdot \alpha_{i+1,p}$, later transformed into $N_k \to \alpha_{i,s} \cdot \alpha_{i+1,p}$, with $[HB] \in N_k$ has been built in G_3.
   At this point the parsing can proceed in the same way by using suitable rules generated during the various iterations of the construction algorithm.

3.2) It may also happen that, during the parsing, a reduction involving more than two consecutive α's, of the type $\ldots \to \lessdot \alpha_i \doteq \alpha_{i+1} \doteq \cdots \doteq \alpha_{i+k} \gtrdot$ (with no $\lessdot, \gtrdot$ in between) becomes necessary, with

$$\ldots \to \lessdot \alpha_i \doteq \alpha_{i+1} \doteq \cdots \doteq \alpha_{i+k} \gtrdot \stackrel{*}{\Rightarrow} x_j \cdot x_{j+1} \ldots x_{j+h}$$

where $j \geqslant i$, $h \geqslant k$.
For the sake of simplicity assume that such a reduction occurs right after the second phase (the reasoning applies identically to other cases): this means that $\alpha_i \stackrel{*}{\Rightarrow} x_{i-1,s} \cdot x_{i,p}$, etc.
Thus the construction of G_3 has built the rule $N_k \to \alpha_{i,s} \cdot \alpha_{i+1,p}$ defined as above (notice: $\alpha_{i+1,p}$ coincides with α_{i+1} since α_{i+1} does not contain any $\lessdot, \gtrdot$; not the same for $\alpha_{i,s}$).
At a subsequent iteration $I > 3$, grammar G_I contains the further rule $N_s \to \alpha_{i,s} \cdot \alpha_{i+1,p} \cdot \alpha_{i+2,p}$ (again $\alpha_{i+2,p}$ coincides with α_{i+2}) and so on, until producing $N_t \to \lessdot \alpha_i \doteq \alpha_{i+1} \doteq \cdots \doteq \alpha_{i+k} \gtrdot$. Since at every iteration new rules are generated and since k is bounded by the hypothesis that there is no circularity in the $\doteq$ relation, the rule $N_t \to \lessdot \alpha_i \doteq \alpha_{i+1} \doteq \cdots \doteq \alpha_{i+k} \gtrdot$ has certainly been generated by the construction of $\widehat{G}$. Thus, the parsing of x can proceed until the full derivation has been built bottom up.

We finish with an example to illustrate the construction of the grammar for Kleene star.

Example 2. Consider the following FG

$$G = \{S \to A \mid B,\ A \to aA \mid a,\ B \to b\}$$

with $OPM(G) = \{a \lessdot a\}$. For the undetermined relations, we choose $a \doteq b$, $b \gtrdot a$, and $b \gtrdot b$. We list the results of the first steps of the construction. Grammar F_2, after disciplined normalization, is:

$$\begin{aligned}
&S \to \{[AA], A\} \mid \{[AB]\} \mid \{[AS]\} \mid \{[BA]\} \mid \{[BB]\},\\
&\{[AA], A\} \to a \mid a\{[AA], A\},\\
&\{[AB]\} \to ab,\\
&\{[AS]\} \to a\{[AB]\} \mid a\{[AS]\},\\
&\{[BA]\} \to Ba\{[AA], A\} \mid Ba,\\
&B \to b,\\
&\{[BB]\} \to Bb.
\end{aligned}$$

Then the grammar G_2 generating $L(F_2) \cup L(G)$ is obtained by adding the rule $S \to B$. For brevity, we develop just in part the next iteration producing grammar F_3 generating $L(G_2) \cdot L(G)$. We focus on the concatenation of the languages generated from nonterminals $\{[AB]\}$ and A. Since $b \gtrdot a$, the original concatenation algorithm would produce the rules

$$\big[\{[AB]\}A\big] \to \{[AB]\}aA \mid \{[AB]\}a$$

and dropping B (see item 1. of the iterative step), we obtain

$$\{[AA]\} \to \{[AB]\}aA \mid \{[AB]\}a.$$

In this way the new nonterminal $\{[AA]\}$ generates strings of the form a^+ba^+.

References

[1] S. Crespi-Reghizzi, D. Mandrioli, Operator precedence and the visibly pushdown property, in: A.H. Dediu, H. Fernau, C. Martín-Vide (Eds.), LATA, in: Lecture Notes in Comput. Sci., vol. 6031, Springer, ISBN 978-3-642-13088-5, 2010, pp. 214–226.
[2] R. Alur, P. Madhusudan, Visibly pushdown languages, in: L. Babai (Ed.), STOC, ACM, ISBN 1-58113-852-0, 2004, pp. 202–211.
[3] R. Alur, P. Madhusudan, Adding nesting structure to words, J. ACM (ISSN 0004-5411) 56 (2009) 16:1–16:43.
[4] R. Floyd, Syntactic analysis and operator precedence, J. ACM 10 (3) (1963) 316–333.
[5] R. McNaughton, Parenthesis grammars, J. ACM 14 (3) (1967) 490–500.
[6] J. Thatcher, Characterizing derivation trees of context-free grammars through a generalization of finite automata theory, J. Comput. System Sci. 1 (1967) 317–322.
[7] S. Crespi-Reghizzi, M.A. Melkanoff, L. Lichten, The use of grammatical inference for designing programming languages, Commun. ACM 16 (2) (1973) 83–90.
[8] S. Crespi-Reghizzi, D. Mandrioli, D.F. Martin, Algebraic properties of operator precedence languages, Inf. Control 37 (2) (1978) 115–133.
[9] D. Grune, C.J. Jacobs, Parsing Techniques: A Practical Guide, Springer, New York, 2008.
[10] R. McNaughton, S. Papert, Counter-Free Automata, MIT Press, Cambridge, USA, 1971.
[11] S. Crespi-Reghizzi, G. Guida, D. Mandrioli, Operator precedence grammars and the noncounting property, SICOMP: SIAM J. Comput. 10 (1981) 174–191.
[12] J. Berstel, L. Boasson, Balanced grammars and their languages, in: W. Brauer, et al. (Eds.), Formal and Natural Computing, in: Lecture Notes in Comput. Sci., vol. 2300, Springer, ISBN 3-540-43190-X, 2002, pp. 3–25.

[13] D. Fisman, A. Pnueli, Beyond regular model checking, in: R. Hariharan, M. Mukund, V. Vinay (Eds.), FSTTCS, in: Lecture Notes in Comput. Sci., vol. 2245, Springer, ISBN 3-540-43002-4, 2001, pp. 156–170.
[14] D. Caucal, Synchronization of pushdown automata, in: O.H. Ibarra, Z. Dang (Eds.), Developments in Language Theory, in: Lecture Notes in Comput. Sci., vol. 4036, Springer, ISBN 3-540-35428-X, 2006, pp. 120–132.
[15] D. Caucal, S. Hassen, Synchronization of grammars, in: E.A. Hirsch, A.A. Razborov, A.L. Semenov, A. Slissenko (Eds.), CSR, in: Lecture Notes in Comput. Sci., vol. 5010, Springer, ISBN 978-3-540-79708-1, 2008, pp. 110–121.
[16] D. Nowotka, J. Srba, Height-deterministic pushdown automata, in: L. Kucera, A. Kucera (Eds.), MFCS, in: Lecture Notes in Comput. Sci., vol. 4708, Springer, ISBN 978-3-540-74455-9, 2007, pp. 125–134.
[17] K. Mehlhorn, Pebbling mountain ranges and its application to DCFL-recognition, in: ICALP: Annual International Colloquium on Automata, Languages and Programming, 1980, pp. 422–435.
[18] B. von Braunmühl, R. Verbeek, Input-driven languages are recognized in $\log n$ space, in: Fundamentals (or Foundations) of Computation Theory, in: Lecture Notes in Comput. Sci., vol. 158, 1983, pp. 40–51.
[19] M. Harrison, Introduction to Formal Language Theory, Addison–Wesley, Reading, MA, 1978.
[20] A. Salomaa, Formal Languages, Academic Press, New York, NY, 1973.
[21] M. 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.
[22] D. Knuth, A characterization of parenthesis languages, Inf. Control 11 (1967) 269–289.
[23] N. Limaye, M. Mahajan, A. Meyer, On the complexity of membership and counting in height-deterministic pushdown automata, in: E.A. Hirsch, et al. (Eds.), CSR, in: Lecture Notes in Comput. Sci., vol. 5010, Springer, ISBN 978-3-540-79708-1, 2008, pp. 240–251.
[24] A. Aho, M. Lam, R. Sethi, J. Ullman, Compilers: Principles, Techniques, and Tools, second ed., Pearson/Addison–Wesley, Boston, MA, USA, ISBN 0-321-48681-1, 2007.
[25] A. Okhotin, Boolean grammars, Inform. and Comput. 194 (1) (2004) 19–48.
[26] K. Hemerik, Towards a taxonomy for ECFG and RRPG parsing, in: A.H. Dediu, A.-M. Ionescu, C. Martín-Vide (Eds.), LATA, in: Lecture Notes in Comput. Sci., vol. 5457, Springer, ISBN 978-3-642-00981-5, 2009, pp. 410–421.
[27] J. Hopcroft, J. Ullman, Introduction to Automata and Formal Languages, Addison–Wesley, Reading, MA, 1979.
[28] V. Lonati, D. Mandrioli, M. Pradella, Precedence automata and languages, in: CSR 2011, St. Petersburg, 2011.
[29] S. Crespi-Reghizzi, G. Guida, D. Mandrioli, Noncounting context-free languages, J. ACM 25 (1978) 571–580.