# Operator precedence and the visibly pushdown property [⋆]

Stefano Crespi Reghizzi and Dino Mandrioli

Dipartimento di Elettronica e Informazione, Politecnico di Milano
P.za Leonardo da Vinci 32, I–20133 Milano
{stefano.crespireghizzi, dino.mandrioli}@polimi.it

**Abstract.** Operator precedence languages, designated as Floyd's Languages (FL) to honor their inventor, are a classical deterministic context-free family. FLs are known to be a boolean family, and have been recently shown to strictly include the Visibly Pushdown Languages (VPDL); the latter are FLs characterized by operator precedence relations determined by the alphabet partition. In this paper we give the non-obvious proves that FLs have the same closure properties that motivated the introduction of VPDLs, namely under reversal, concatenation and Kleene's star. Thus, rather surprisingly, the historical FL family turns out to be the largest known deterministic context-free family that includes the VPDL and has the same closure properties needed for applications to model checking and for defining mark-up languages such as HTML. As a corollary, an extended regular expression of precedence-compatible FLs is a FL and a deterministic parser for it can be algorithmically obtained.

## 1 Introduction

From the very beginning of formal language science, research has struggled with the wish and need to extend as far as possible the elegant and practical properties of regular languages to other language families that overcome the limitations of finite-state models in terms of expressivity and allow more accurate modelling of relevant phenomena. In particular, it is well known that closure properties under basic operations allow to automatically construct complex models from simple ones, and to decide important problems: e.g. model checking relies on closure w.r.t. boolean operations and on decidability of the emptiness problem. Since, among the classic formal language families, only regular languages enjoy closure w.r.t. the needed operations, the search for new subclasses – mostly of context-free (CF) languages – exhibiting the wished properties, is a long-standing research concern.

A major step has been made by McNaughton with parenthesis grammars [12], whose productions are characterized by enclosing any righthand side (r.h.s) within a pair of parentheses; the alphabet is the disjoint union of internal characters and the pair. By considering instead of strings the *stencil* (or skeletal) trees encoded by parenthesized strings, some typical 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 production stencils. Further mathematical developments of those ideas have been pursued in the setting of tree automata [16]. Short after McNaughton's results, we investigated similar closure properties of *Floyd's* operator precedence *Grammars* and *Languages*

[⋆] 10.02.2010. Partially supported by PRIN 2007TJNZRE-002, CNR-IEIIT and ESF AutoMathA.

[10] [1] (FG and FL), elegant precursors of LR($k$) grammars and Deterministic CF (DCF) languages, also exploited in early work on grammar inference [5]. The production set of an operator grammar determines three binary precedence relations (greater/less/equal) over the alphabet, that for a FG grammar are disjoint, and are presented in a matrix. The precedence matrix, even in the absence of the productions, fully determines the topology (or stencil) of the syntax tree, for any word that is generated by any FG having the same precedence matrix. The families of FGs that share the same precedence matrix and the corresponding languages are boolean algebras [8, 5].
We also extended the notion of *non-counting* regular languages of McNaughton and Papert [13] to parenthesis languages and to FLs [6].

Decades later, novel interest for parentheses-like languages arose from research on mark-up languages such as XML, and produced the family of *balanced grammars* and languages [3]. They generalize parenthesis grammars in two ways: several pairs of parentheses are allowed, and the r.h.s of grammar rules allow for regular expressions over nonterminal and internal symbols to occur between matching parentheses. The property of uniqueness of the minimal grammar is preserved, and the family has the closure property w.r.t. concatenation and Kleene star, which was missing in parenthesis languages. Clearly, balanced as well as parenthesis languages, are closed under reversal.

Model checking and static program analysis provide another motivation for such families of languages – those that extend the typical finite-state properties to infinite-state pushdown systems. The influential paper by Alur and Madhusudan [1] (also in [2]) defines the *visibly pushdown automata* and *languages* (VPDA, VPDL), a subclass of realtime pushdown automata and DCF. The input alphabet is partitioned into three sets named calls (or opening), returns (or closing), and internals. The decision of the type of move to perform (push, pop, or a stack neutral move) is purely driven by the membership of an input character in one of the three sets, a property that justifies the name “visibly pushdown”. VPDLs extend balanced grammars in ways that are important for modelling symbolic program execution. For each partitioned alphabet, the corresponding language family is closed under reversal and boolean operations, concatenation and Kleene star.

Guided by the intuition that precedence relations between terminals in a FG determine the action on the pushdown stack in a more flexible way than in a VPDA, we recently [7] proved that VPDLs are a proper subclass of FLs, characterized by a fixed partition of the precedence matrix, induced by the alphabetic partition into opening, closing and internal letters.

From the standpoint of their generative capacity and expressivity, FGs are more powerful than VPDAs in various practical ways. An example of structural adequacy possible with FG but not with VPDA is the semantic precedence of multiplication operators over additive ones (which inspired Floyd in the definition of operator precedence). An example of the higher generative capacity are the nested constructs opened and closed by means of a sequence of characters, e.g. `/* .... */`, rather than by two distinct single characters, one

[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.

for the opening and one for the closing of the scope. Overall FGs, though less general than LR(1) grammars, offer a comfortable notation for specifying syntaxes of the complexity of programming languages, and are still occasionally used in compilation [11]. Surprisingly enough, nothing was known on the closure of FL under concatenation and Kleene star. This paper provides a positive but not immediate answer, in contrast to the fact that for the main language families closure under the two operations is either trivially present or trivially absent: examples of the former case are CF and VPDLs with a fixed alphabetic partitioning, while DCF is an example of the latter. In this perspective FGs represent an interesting singularity: they are closed but in a far from obvious way. Precisely, although the parse tree of a string $x \cdot y$ is solely determined by the given precedence relations of the grammars generating the two factors, the tree of $x \cdot y$ may be sharply different from the pasting together 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 composing the trees of either $x \cdot x$ and $x$ or $x$ and $x \cdot x$, but may have an entirely different structure.

Thus, rather surprisingly, a classical, half-forgotten language family turns out to enjoy all the desirable properties that motivated the recent invention of VPDLs! To the best of our knowledge FG currently qualifies as the largest DCF family closed under boolean operations, reversal, concatenation and Kleene star.

The paper proceeds as follows: Section 2 lists the essential definitions of FG, and a summary of known results; Section 3 proves closure under concatenation; Section 4 proves closure under Kleene star and shows an application of the closure properties to regular expressions. Section 5 concludes. Appendix 1 shows introductory examples. Appendices 2 and 3 contain the full proofs of the main theorems.

## 2 Basic definitions and properties

We list the essential definitions of Floyd grammars. For the basic definitions of CF grammars and languages, we refer to any textbook, such as [15]. The empty string is denoted $\varepsilon$, the terminal alphabet is $\Sigma$. For a string $x$ and a letter $a$, $|x|_a$ denotes the number of occurrences of letter $a$, and the same notation $|x|_\Delta$ applies also to a set $\Delta \subseteq \Sigma$; $first(x)$ and $last(x)$ denote the first and last letter of $x \neq \varepsilon$. The projection of a string $x \in \Sigma^*$ on $\Delta$ is denoted $\pi_\Delta(x)$.

A *Context-Free* CF grammar is a 4-tuple $G = (V_N, \Sigma, P, S)$, where $V_N$ is the nonterminal alphabet, $P$ is the production set, $S$ is the axiom, and $V = V_N \cup \Sigma$. An *empty rule* has $\varepsilon$ as right hand side (r.h.s.). A *renaming rule* has one nonterminal as r.h.s. A grammar is *reduced* if every production can be used to generate some terminal string. A grammar is *invertible* if no two productions have identical r.h.s.

The following naming convention will be adopted, unless otherwise specified: lowercase Latin letters $a, b, c$ denote terminal characters; letters $u, v, x, y, w, z$ denote terminal strings; Latin capital letters $A, B, C$ denote nonterminal symbols, and Greek letters $\alpha, \ldots, \omega$ denote strings over $V$. The strings may be empty, unless stated otherwise.

For a production $A \to u_0 A_1 u_1 A_2 \ldots u_{k-1} A_k$, $k \geq 0$, the *stencil* is the production $N \to u_0 N u_1 N \ldots u_{k-1} N$, where $N$ is not in $V_N$.

A production is in *operator form* if its r.h.s. has no adjacent nonterminals, and an *operator grammar* (OG) contains just such productions. Any CF grammar admits an equivalent OG, which can be also assumed to be invertible [15].
For a CF grammar $G$ over $\Sigma$, the associated *parenthesis grammar* [12] $\widetilde{G}$ has the rules obtained by enclosing each r.h.s. of a rule of $G$ within the parentheses '[' and ']' that are assumed not to be in $\Sigma$.
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 $G$ consider a *sentential form* $\alpha$ with $S \overset{*}{\Rightarrow} \alpha$ and $\alpha = \beta A$, $A \in V_N$. Then $A$ is a *Suffix of the Sentential Form* (SSF); similarly we define the *Prefix of a Sentential Form* (PSF).

The coming definitions for operator precedence grammars [10], here named *Floyd Grammars*, are from [8]. (See also [11] for a recent practical account.)
For a nonterminal $A$ of an OG $G$, the *left and right terminal sets* are

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

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

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

where $D$ is a new nonterminal and $G'$ is the same as $G$ except for the addition of the production $D \to \beta$. 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 $\alpha, \beta$ range over $(V_N \cup \Sigma)^*$ and $a, b \in \Sigma$. Three binary operator precedence (OP) relations are defined:

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

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 $b$. Between two OPMs $M_1$ and $M_2$, we define set inclusion and operations.

$$M_1 \subseteq M_2 \text{ if } \forall a, b : M_{1,ab} \subseteq M_{2,ab}, \quad M = M_1 \cup M_2 \text{ if } \forall a, b : M_{ab} = M_{1,ab} \cup M_{2,ab} \tag{4}$$

**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., $\forall a, b$, $|M_{ab}| \leq 1$. Two matrices are compatible if their union is conflict-free. A matrix is total if it contains no empty case.*

In the following all precedence matrices are conflict-free.

## 2.1 Known properties of Floyd grammars

We recall some relevant definitions and properties of FGs. To ease cross-reference, we follow the terminology of [8].

**Definition 2.** *Normal forms of FG*
*A FG is in Fischer normal form [9] if it is invertible, the axiom $S$ does not occur in the r.h.s. of any production, the only permitted renaming productions have $S$ as left hand side, and no $\varepsilon$-productions exist, except possibly $S \to \varepsilon$.*
*An FG is in homogeneous normal form [8, 5] if it is in Fischer normal form and, for any production $A \to \alpha$ 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, all of its alternative productions have the same pairs of left and right terminal sets.

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

For the reader unfamiliar with FGs, in Appendix 1 an example illustrates the formalism, and the boolean family induced by a given precedence matrix, to be next presented.

We consider FGs having identical or compatible precedence relations and we state their boolean properties.

**Definition 3.** *Precedence-compatible grammars*
*For a precedence matrix $M$, the class [8] $C_M$ of precedence-compatible FGs is*

$$C_M = \{G \mid OPM(G) \subseteq M\}\,.$$

The equal-precedence relations of a FG have to do with an important parameter of the grammar, namely the maximal length of the r.h.s. of the productions. Clearly, a production $A \to A_1 a_1 \ldots A_t a_t A_{t+1}$, where each $A_i$ is a possibly missing nonterminal, is associated with $a_1 \doteq a_2 \doteq \ldots \doteq a_t$. If the $\doteq$ relation is circular, the grammar can have productions of unbounded length. Otherwise the length of any r.h.s. is bounded by $(2.c) + 1$, where $c$ 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 restrict attention to grammars with bounded r.h.s. This can be done in two ways.

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

$$C_{M,k} = \{G \mid G \in C_M \wedge (\forall \text{ production } A \to \alpha \text{ of } G, |\alpha| \leq k)\}$$

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

$$C_{M,\doteq} = \{G \mid G \in C_M \mid \textit{ matrix } M \textit{ is } \doteq\textit{-acyclic}\}$$

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

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

The following closure properties are from [8] (Corol. 5.7 and Theor. 5.8).

**Statement 2** *For every precedence matrix $M$, the class of FLs*

$$\{L(G) \mid G \in C_{M,k}\}$$

*is a boolean algebra.*

In other words, the proposition applies to languages generated by right-bounded FGs having precedence matrices that are included in, or equal to some matrix $M$. Notice that the top element of the boolean lattice is the language of the FG that generates all possible syntax trees compatible with the precedence matrix; in particular, if $M$ is total, the top element is $\Sigma^*$.

We observe that the boolean closure properties of VPDL immediately follow from Statement 2 and from the fact that a VPDL is a FL characterized by a particular form of precedence matrix [7].

*Other simple properties*

**Statement 3** *The class of FG languages is closed with respect to reversal.*

This follows from the fact that, if $a \lessdot b$ for a FG grammar $G$, then it holds $b \gtrdot a$ for the grammar $G^R$ obtained by reversing the r.h.s. of the productions of $G$; and similarly for $a \gtrdot b$. The $a \doteq b$ relation is turned into $b \doteq a$ by production reversal. It follows that $G^R$ is a FG.

It is interesting to briefly discuss the cases of very simple precedence matrices. First consider a matrix $M$ containing only $\lessdot$, hence necessarily conflict free. Then the nonempty r.h.s.'s of the productions of any grammar in $C_M$ may only be of the types $aN$ or $a$. Therefore the grammar is *right-linear*. (Conversely for $\gtrdot$ and left-linearity.) Second, suppose $M$ does not contain $\doteq$. Then any production of any grammar in $C_M$ only admits one terminal character. Notice that *linear* CF grammars can be cast in that form, but not all linear CF languages are FG since they may be non-DCF.

To finish, we compare regular languages and FGs.

**Statement 4** *Let $R \subseteq \Sigma^*$ be a regular language. There exists a FG grammar for $R$ in the family $C_{M,2}$, where $M$ is the precedence matrix such that $M_{ab} = \lessdot$ for all $a, b \in \Sigma$.*

The statement follows from the fact that every regular language is generated by a right-linear grammar. If the empty string is in $R$, the FG has the axiomatic rule $S \to \varepsilon$.
A stronger statement holding for any precedence matrix will be proved in Subsection 4.1 as a corollary of the main theorems.

## 3 Closure under concatenation

Although FGs are the oldest deterministic specialization of CF grammars, 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 outlined, though necessarily 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 $\Big(a\big(a(a)\big)\Big)$, for $y_2 = bb$ the structure is $\big((b)b\big)$, but the structure of $y_1 \cdot y_2$ is $\Big(a\big(a(ab)b\big)\Big)$, which is not a composition of the two.

The following notational conventions apply also to Section 4. 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 $V_{N_1} = \{S_1, A_1, A_2, \ldots\}$ and $V_{N_2} = \{S_2, B_1, B_2, \ldots\}$, in order to have distinct names for the axioms and nonterminals of the two grammars.
To simplify the proofs we operate on FGs 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 $\alpha\beta \notin ((\Sigma \cup V_N)^* \cdot V_N V_N \cdot (\Sigma \cup V_N))^*$ by positing $\alpha \lessdot \beta \iff \mathcal{R}(\alpha) \lessdot \mathcal{L}(\beta)$, 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 1.** *Let $G_1, G_2$ be FGs such that $OPM(G_1)$ is compatible with $OPM(G_2)$. Then a FG grammar $G$ can be effectively constructed such that*

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

*Proof* 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 at every step, the algorithm checks the precedence relation between two letters $a$ and $b$, and if $M_{ab} = \emptyset$, it can *arbitrarily* assign a value to $M_{ab}$, thus obtaining a matrix compatible with $M$.

The core of the algorithm builds a “thread” of productions 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 production of the type $[AB] \to \dots ab \dots$, $[AB]$ being a new nonterminal (see Figure 1). If instead the rightmost part of $x_1$ can be parsed without affecting $x_2$ up to a

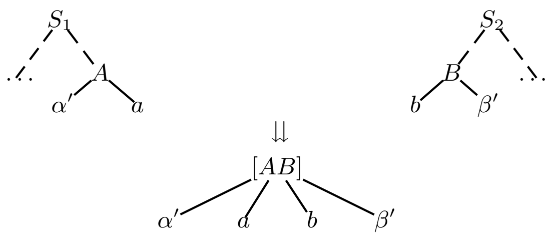


**Fig. 1.** Cross-border production constructed when the facing letters are equal in precedence.

derivation $N \overset{*}{\Rightarrow} y_1$ because $\mathcal{R}(N) \gtrdot b$, then, when the parsing of $x_1$ leads to a production such as $A \to \alpha \cdot a \cdot N$ with $a \doteq b$, the junction of the two syntax trees begins at that point by means of a production such as $[AB] \to \alpha \cdot a \cdot N \cdot b \cdot \beta$ (see Figure 2) so that the original precedence relations of $G_1$ and $G_2$ are unaffected.

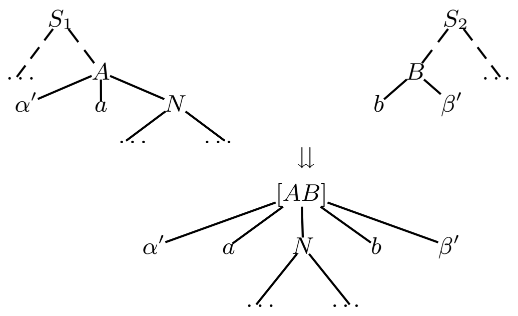
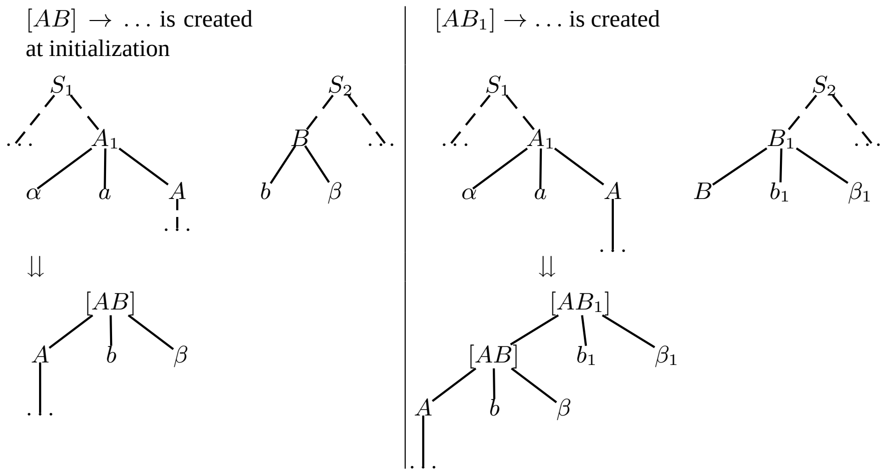


**Fig. 2.** Another case of cross-border production when $\mathcal{R}(N) \gtrdot b$.

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

After this initialization, the construction of the “joint parsing thread” follows the natural bottom-up parsing. For instance, suppose that a nonterminal of 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 $A_1 \to \alpha \cdot a \cdot A$ ($A_1$ being a SSF) and symmetrically $B_1 \to B \cdot b \cdot \beta$, with $a \doteq b$, then the new production $[A_1 B_1] \to \alpha \cdot a \cdot [AB] \cdot b \cdot \beta$ is created.
The cases $a \gtrdot b$ and $a \lessdot b$ are treated accordingly. The last situation is illustrated in Figure 3 (right). $\square$

The complete proof is in Appendix 2.



$[AB] \rightarrow \ldots$ is created at initialization
$[AB_1] \rightarrow \ldots$ is created
$S_1$
$A_1$
$\alpha$
$a$
$A$
$S_2$
$B$
$b$
$\beta$
$\Downarrow$
$[AB]$
$B_1$
$b_1$
$\beta_1$
$[AB_1]$

**Fig. 3.** Productions created by growing the cross-border thread.

## 4 Closure under Kleene star

In many language families the closure under Kleene star comes together with the closure under union and concatenation. Thus for a CF language $L$, 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 components $y_1, y_2, \ldots, y_i$. In the case of FG, a similar composition is in general 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 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 not the composition of the structures of either $y_1 \cdot y_2$ and $y_3$, or 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 language $L = \{aa\}$ with the circular precedence relation $M = \{a \doteq a\}$, and a string $a^{2p}, p \geq 0$, in the Kleene closure of $L$. The FG grammar of $L^*$ with OPM $M$ would then need to contain an unbounded production set $\{S \rightarrow a^{2p}, p \geq 0\}$, which is not permitted by the standard definition[2] of CF grammar. For this reason we make the hypothesis that the precedence matrix is $\doteq$-acyclic.

**Theorem 2.** *Let $G = (V_N, \Sigma, P, S)$ be a FG such that $OPM(G)$ is $\doteq$-acyclic. Then a 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 1 we assume without loss of generality the precedence matrix to be total, and in this case also $\doteq$-acyclic. Not surprisingly the construction of $\widehat{G}$ is based on the construction in Theorem 1 for $L.L$, but the required extensions involve some technical difficulties.

[2] We do not discuss the possibility of allowing regular expressions in the productions.

We need to consider only the irreflexive closure $L^+$, since for $L^*$ it suffices to add the production $S \to \varepsilon$. We assume the form of grammar $G$ to be homogeneous. Again, we give here just an intuitive description of the construction; the complete proof of the theorem is Appendix 3.

$\widehat{G}$ is built as the last element of a series of grammars that begins with $G_1 = G$ and continues with the grammar $G_2$ that generates $L \cdot L \cup L$, computed according to the concatenation algorithm outlined in Section 3 and the union algorithm implied by Statement 2. 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 nonterminals of the type $[[AB]C]$. Obviously this process cannot be iterated indefinitely since it would produce a grammar with infinite nonterminals and productions. Thus, nonterminals of type $[[AB]C]$ are "collapsed" into $[AC]$. Intuitively, this operation is justified by the observation that the production of an "intermediate" nonterminal of the type $[[AB]C]$ means that, in $G$, $A \overset{*}{\Rightarrow} x_1$ a suffix of some string $x \in L$, $B \overset{*}{\Rightarrow} y$ belonging to $L$, and $C \overset{*}{\Rightarrow} z_1$, a prefix of some $z \in L$. In this way, the number of possible new nonterminals is bounded and the construction of $\widehat{G}$ terminates when no new nonterminals and productions are generated. Figure 4 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 type $\{[AB], [CD], E\}$, i.e., collection of "boundary nonterminals". This is due to the need to iteratively apply a normalization procedure that eliminates repeated r.h.s. For instance, suppose that during the process the following productions are built:

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

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.'s produces a normalized homogeneous grammar containing the rules:

$$\{[AB], [CD], E\} \to \alpha, \quad \{[AB]\} \to \beta, \quad \{[CD]\} \to \gamma, \quad \{E\} \to \delta$$

□

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

## 4.1 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 a FG that generates $R$, with $M$ as OPM. The positive answer follows from Theorems 1 and 2 under fairly general hypotheses.

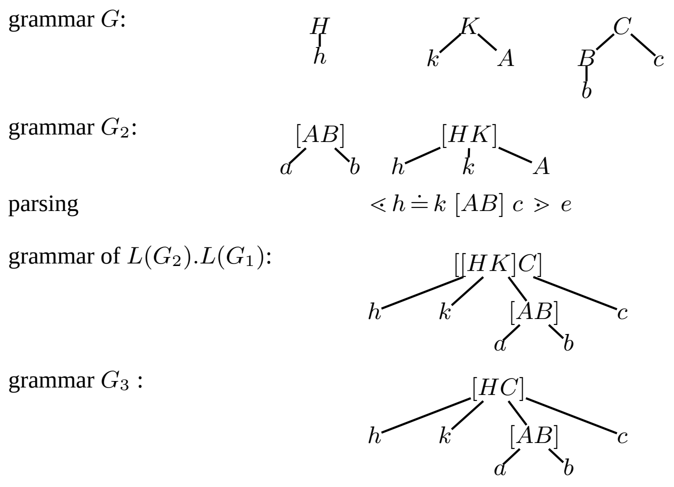


**Fig. 4.** Productions used for parsing a string in $L.L.L.$

**Corollary 1.** *Let $M$ be any total $\doteq$-acyclic precedence matrix over $\Sigma$. Then the family of regular languages over $\Sigma$ is (strictly) included in the family of languages generated by FGs ranging over $C_M$.*

*Proof.* Let $R$ be defined by a regular expression. In order to construct a FG grammar 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 [8] or of Theorem 1 respectively produce a grammar, compatible with $M$, for the union or concatenation. Similarly, anytime a subexpression is under star, the construction of Theorem 2 produces the corresponding grammar. $\square$

This result gives a procedure, based on the previous algorithms, for constructing from a regular expression, a FG 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 VPL [1, 7], the procedure returns a grammar with the specified partition of the alphabet into call, return and internal symbols.

Notice that the procedure works as well for ambiguous regular expressions. We are not aware of comparable methods for constructing a deterministic CF grammar, in order to parse in accordance with a prescribed syntactic structure, a language specified by a regular expression.

The same procedure, when applied to a regular expression, possibly augmented with intersection and complement, over precedence-compatible FG languages, permits to obtain an equivalent FG. From a practical standpoint, this approach would permit to construct a precedence parser for a grammar featuring “extended” regular expressions in the right-hand sides. This should of course be contrasted with the well-known non-closure of DCF under intersection and union.

## 5 Conclusion

We mention some questions raised by the present study.

Every class of Floyd languages over a given $\doteq$-acyclic precedence matrix includes (possibly after filling the precedence matrix to totality) the regular language family and is closed with respect to the basic operations: concatenation, Kleene star, reversal, and all boolean operations. The FG family appears at present to be the largest family exhibiting such properties; in particular it strictly includes [7] the visibly pushdown language (VPDL) family. On the other hand, several other language families fall in between the VPDL and the CF, such as the height-deterministic family of [14] or the synchronized pushdown languages of [4], but some of the basic closure properties are missing or the language family is nondeterministic. We wonder whether a significant family of deterministic CF languages, more general than FL, yet preserving the same closure properties, can be found.

In another direction, one could ask a similar question about the invariance property of FLs with respect to the non-counting (or aperiodicity) property [6].

It would be interesting to assess the suitability of Floyd languages for applications, especially to XML documents and to model checking, that have motivated other language models such as the balanced grammars and the visibly pushdown languages. We observe that the greater generative capacity of FGs should improve the realism of the intended models, by permitting the use of more flexible multi-level and recursively nested structures.

Finally, to apply these theoretical results in practice, e.g. to derive model checking algorithms therefrom, the computational complexity aspects should be investigated. Admittedly, the closure algorithms presented in this paper have a typical combinatorial nature: the worst case complexity in fact is dominated by the size of nonterminal alphabets which are constructed as power sets of the original ones. Notice also that the algorithms assume as starting point grammars in homogeneous normal form, which in turn require a nonterminal alphabet constructed on top of the power set of the terminal alphabet.

On the other hand, the risk of combinatorial explosion is rather intrinsic in these families of algorithms, starting from the seminal papers on model checking and VPDL. We hope that further research will produce suitable heuristics and techniques to manage such a complexity with the same success obtained by the long-standing research in model checking.

*Aknowledgment.* We thank Jean Berstel for lively discussions.

## References

1. R. Alur and P. Madhusudan. Visibly pushdown languages. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2004.
2. R. Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3), 2009.
3. J. Berstel and L. Boasson. Balanced grammars and their languages. In W. Brauer et al., editor, *Formal and Natural Computing*, volume 2300 of *LNCS*, pages 3–25. Springer, 2002.
4. D. Caucal. Synchronization of pushdown automata. In O. H. Ibarra and Z. Dang, editors, *Developments in Language Theory*, volume 4036 of *LNCS*, pages 120–132. Springer, 2006.
5. S. Crespi Reghizzi. *The mechanical acquisition of precedence grammars*. PhD thesis, University of California UCLA, School of Engineering, 1970.

6. S. Crespi-Reghizzi, G. Guida, and D. Mandrioli. Operator precedence grammars and the noncounting property. *SICOMP: SIAM Journ. on Computing*, 10:174—191, 1981.
7. S. Crespi-Reghizzi and D. Mandrioli. Algebraic properties of structured context-free languages: old approaches and novel developments. In *WORDS 2009 - 7th Int. Conf. on Words, preprints*. available as http://arXiv.org/abs/0907.2130, 2009.
8. S. Crespi-Reghizzi, D. Mandrioli, and D. F. Martin. Algebraic properties of operator precedence languages. *Information and Control*, 37(2):115–133, May 1978.
9. M. J. Fischer. Some properties of precedence languages. In *STOC '69: Proc. first annual ACM Symp. on Theory of Computing*, pages 181–190, New York, NY, USA, 1969. ACM.
10. R. W. Floyd. Syntactic analysis and operator precedence. *J. ACM*, 10(3):316–333, 1963.
11. D. Grune and C. J. Jacobs. *Parsing techniques: a practical guide*. Springer, New York, 2008.
12. R. McNaughton. Parenthesis grammars. *J. ACM*, 14(3):490–500, 1967.
13. R. McNaughton and S. Papert. *Counter-free Automata*. MIT Press, Cambridge, USA, 1971.
14. D. Nowotka and J. Srba. Height-deterministic pushdown automata. In L. Kucera and A. Kucera, editors, *Mathematical Foundations of Computer Science 2007, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, volume 4708 of *LNCS*, pages 125–134. Springer, 2007.
15. A. K. Salomaa. *Formal Languages*. Academic Press, 1973.
16. J. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *Journ. of Comp. and Syst.Sc.*, 1:317–322, 1967.

## Appendix 1. Illustration of Floyd grammars

*Example 1.* The languages

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

are generated by the homogeneous normal form grammars:

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

Their precedence matrices are:

$$M_1 = \begin{array}{c|c|c|c|c|c|} & b & c & d & e & f \\ \hline b & \lessdot & \doteq & & & \\ \hline c & & \gtrdot & & & \\ \hline d & & & & & \\ \hline e & & & & & \\ \hline f & & & & & \\ \hline \end{array} \quad M_2 = \begin{array}{c|c|c|c|c|c|} & b & c & d & e & f \\ \hline b & & & & & \\ \hline c & & & & & \\ \hline d & & & \gtrdot & & \\ \hline e & & & & & \\ \hline f & & & \doteq & & \lessdot \\ \hline \end{array} \quad M_3 = \begin{array}{c|c|c|c|c|c|} & b & c & d & e & f \\ \hline b & & & & & \gtrdot \\ \hline c & & & & & \\ \hline d & & & & & \\ \hline e & & & & \lessdot & \doteq \\ \hline f & \doteq & & & & \\ \hline \end{array}$$

The three matrices are conflict-free and precedence compatible (in particular they are disjoint). Consider the matrix $M = M_1 \cup M_2 \cup M_3$, which is $\doteq$-acyclic, and the grammar/language families $C_{M,4}$ and $C_{M,\doteq}$. The two families contains the three grammars, as well as the union language $L_{123} = L_1 \cup L_2 \cup L_3$, whose grammar $F_{123}$ has the productions $S_{123} \to A \mid B \mid C$ instead of $S_1 \to A$, etc.
Incidentally, in [7] it is proved that language $L_{123}$ is not a visibly pushdown language [1].
The four grammars are also free FG. Now consider the family of homogeneous FGs $W^{-1}(F_3)$, and a grammar $H$ included therein:

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

We observe that $H$ is a homogeneous but not a free grammar because $\mathcal{L}_H(X_2) = \mathcal{L}_H(X_1)$ and $\mathcal{R}_H(X_2) = \mathcal{R}_H(X_1)$. Then $H \leq F_3$ and $L(H) \subseteq L(F_3)$. The class $W^{-1}(F_3)$ is a boolean algebra.

## Appendix 2. Proof of concatenation closure

**Theorem 1.** Let $G_1, G_2$ be FGs such that $OPM(G_1)$ is compatible with $OPM(G_2)$. Then a FG grammar $G$ can be effectively constructed such that

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

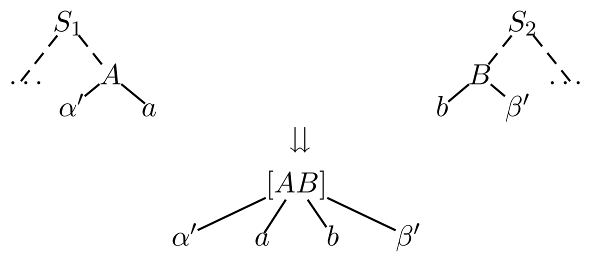
*Proof.* We assume that $G_1$ and $G_2$ are in homogeneous normal form. First we present the algorithm that constructs a homogeneous FG $G = (V_N, \Sigma, P, S)$ such that $OPM(G) \supseteq OPM(G_1) \cup OPM(G_2)$. Then, by means of a series of lemmas, we prove that $L(G) = L(G_1).L(G_2)$.

### Algorithm building grammar $G$ of concatenation

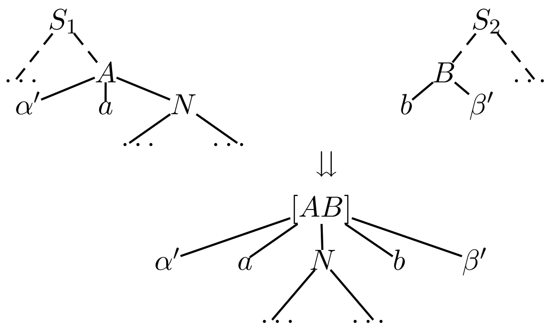
For simplicity, we assume that $M = OPM(G_1) \cup OPM(G_2)$ is a total matrix. This does not affect generality, because if, at any step, the following algorithm checks the precedence relation between two letters $a$ and $b$, and $M_{ab} = \emptyset$, we can *arbitrarily* assign to $M_{ab}$ a value, thus obtaining a matrix compatible with $M$.
Two types of nonterminals occur in $G$: the non-axiomatic nonterminals of $G_1$ and $G_2$ and new nonterminals named by certain pairs $[AB]$ such that $A \in V_{N_1}$ and $B \in V_{N_2}$.
The algorithm may produce non-invertible grammars and may create useless productions. 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 productions of $P_1$ and $P_2$ that do not contain $S_1$ or $S_2$. Then add to $G$ the productions 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 hold:
   - $\alpha = \alpha' a, \beta = b\beta'$, schematized by:



   - $\alpha = \alpha' a N, \beta = b\beta'$, $\mathcal{R}(N) \gtrdot b$, $N \in V_N$ schematized by:



– $\alpha = \alpha' a, \beta = N b \beta'$, $a \lessdot \mathcal{L}(N)$, $N \in V_N$ (the symmetric of the preceding case); add to $V_N$ the nonterminal $[AB]$ and to $P$ the production $[AB] \to \alpha\beta$. $\mathcal{L}_G([AB]) = \mathcal{L}_{G_1}(A)$ and $\mathcal{R}_G([AB]) = \mathcal{R}_{G_2}(B)$, therefore no new precedence relations are added to $M$ by the new productions, because, thanks to the homogenous form of $G_1$ and $G_2$, $\mathcal{L}(\beta) = \mathcal{L}(B)$ and $\mathcal{R}(\alpha) = \mathcal{R}(A)$.

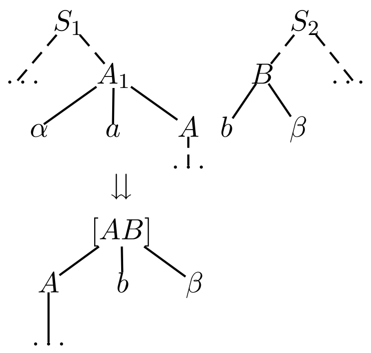
2. *Initial cases* $\lessdot$ and $\gtrdot$.

   (a) *Case* $\lessdot$. For every derivations

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

   such that $\mathcal{R}(A) \gtrdot b$ and $a \lessdot b$, add to $V_N$ the nonterminal $[AB]$ and to $P$ the production $[AB] \to Ab\beta$. This case is schematized below.

step 2(a)



   Remark: from the homogeneous form, for any productions $A \to \alpha_1 \mid \alpha_2$, 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

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

   such that $a \lessdot \mathcal{L}(B)$ and $a \gtrdot b$, add to $V_N$ the nonterminal $[AB]$ and to $P$ the production

   $$[AB] \to \alpha a B$$

Remark: the precedence relations are unaffected.

(c) For every productions

$$S_1 \to A_1, \quad B \to b\beta$$

such that $B$ is a PSF and $\mathcal{R}(A_1) \gtrdot b$, add to $G$ the production $[A_1B] \to A_1b\beta$.
Remark: the precedence relations are unaffected.

(d) Symmetrically, for every

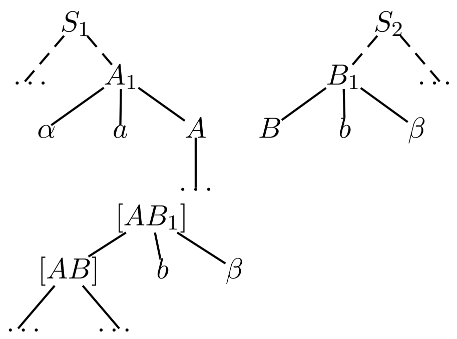
$$A \to \alpha a, \quad S_2 \to B_1$$

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

Then the following rules 3., 4., 5. are applied over and over, until no more productions are added to the grammar. In rules 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 production $[A_1B_1] \to \alpha a[AB]b\beta$.
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 $S_2 \overset{*}{\Rightarrow} B_1\zeta \Rightarrow Bb\beta$ and $[AB] \in V_N$, add the production $[A_1B] \to \alpha a[AB]$ to $P$.
The subcase $S_2 \to B$ is included in the case, because conventionally every $a \in \Sigma$ takes precedence over $\perp$.
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 $S_1 \overset{*}{\Rightarrow} \alpha a A$ and $[AB] \in V_N$, add the production $[AB_1] \to [AB]b\beta$ to $P$.
The next figure illustrates the productions created by Case 5.

step 5 after step 2(a)



The subcase $S_1 \to A$ is included in the case, because conventionally $\perp$ yields precedence to every $b \in \Sigma$.

At last, to create the productions for the axiom, apply the rule:

6. *Case axiom*. Apply the following rules to every pair of productions $S_1 \to A_1$ and $S_2 \to B_1$.
   (a) If $[A_1B_1] \in V_N$, add to $P$ the production $S \to [A_1B_1]$.
   (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 productions $[A_1B_1] \to A_1 b\beta$ and $S \to [A_1B_1]$.
   (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 productions $[A_1B_1] \to \alpha a B_1 \in P$ and $S \to [A_1B_1]$.
   Remark: the productions 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 righthand sides for the productions.
Finally, since each step of the construction does not induce new precedence relations, the grammar $G$ is a 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 productions and repeated righthand sides. The normalization will be refined later, for the construction of Kleene star.

**Proof that $L(G) = L(G_1).L(G_2)$** The inclusion $L(G) \subseteq L(G_1).L(G_2)$ follows easily from the construction, so we just have to prove the inclusion $L(G) \supseteq L(G_1).L(G_2)$. For that we need the next four lemmas. The next one states two symmetrical properties.

**Lemma 1.** – *If there are derivations*

$$S_1 \Rightarrow A_1 \overset{*}{\Rightarrow} x \qquad S_2 \overset{*}{\Rightarrow} y = b.z \quad b \in \Sigma$$

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

$$S \Rightarrow [A_1B] \overset{*}{\Rightarrow} [A_1B_k]\zeta \Rightarrow A_1 b\zeta \overset{*}{\Rightarrow} xy.$$

– *If there are derivations*

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

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

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

*Proof. For the first item: if $S_2 \Rightarrow B_1 \Rightarrow b\beta \overset{*}{\Rightarrow} y$ then by construction 2(c) and 6(a) $\exists[A_1B_1] \to A_1b\beta$ and $S \to [A_1B_1]$. If $S_2 \overset{*}{\Rightarrow} B_k\xi \Rightarrow b\beta\xi \overset{*}{\Rightarrow} y$, then from 2.(a) $\exists[A_1B_k] \to A_1b\beta$ and for every $B_i \to B_j\beta_j$ in the derivation $S_2 \overset{*}{\Rightarrow} B_i\xi \Rightarrow B_j\beta_j\xi \overset{*}{\Rightarrow} y$ there exists by rule 5. a production $[A_1B_i] \to [A_1B_j]\beta_j$. Thus $S \overset{*}{\Rightarrow} x.y$. For the second item the proof is symmetrical.*

**Lemma 2.** *It states two symmetrical properties:*

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

Similar to Lemma 2, the next lemma takes care of the condition $a \doteq b$.

**Lemma 3.** *It states two symmetrical properties:*

- *If there are derivations*
$$S_1 \overset{*}{\Rightarrow} x_1 A \Rightarrow \alpha a A_1 \overset{*}{\Rightarrow} x_1 v w$$
*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 production $[A\,B] \rightarrow \alpha a A_1 b\beta$. (Proof: by rule 1. )*
- *If there are derivations*
$$S_1 \overset{*}{\Rightarrow} xA \Rightarrow x\alpha a \overset{*}{\Rightarrow} xva$$
*and*
$$S_2 \overset{*}{\Rightarrow} By_1 \Rightarrow B_1 b\beta y_1 \overset{*}{\Rightarrow} wzy_1$$
*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 production $[AB] \rightarrow \alpha a B_1 \beta$.*

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

**Lemma 4.** *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*
$$S \overset{*}{\Rightarrow} x[AB]y \overset{*}{\Rightarrow} xw.zy$$

*Proof.* *By induction. The initialization is covered by Lemmas 1, 2, 3, and we explain the induction step. Assume by inductive hypothesis that*
$$S_1 \overset{*}{\Rightarrow} xA_1 \Rightarrow x\alpha a A \overset{*}{\Rightarrow} x\alpha a x_1$$
$$S_2 \overset{*}{\Rightarrow} B_1 y \Rightarrow Bb\beta y \overset{*}{\Rightarrow} y_1 b\beta y$$
*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 production $[A_1 B_1] \rightarrow \alpha a [AB] b\beta$.*

2. $a \gtrdot b$. *Similarly, by construction Case 4.*
3. $a \lessdot b$. *Similarly, by construction Case 5.*

*Example 2.*

$$G_1 = \{S_1 \to A_0, \quad A_0 \to aA_1 \quad A_1 \to aA_1b \mid ab\}$$
$$G_2 = \{S_2 \to B, \quad B \to bc \mid Bbc\}$$

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 | $b$ |

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

We list the results of the steps.
*Initialization*

$$V_N = \{A_0, A_1, B\}; \quad P = \{A_0 \to aA_1, \quad A_1 \to aA_1b \mid ab, \quad B \to bc \mid Bbc\}$$

1. Initial case $\doteq$
$A_0, A_1$ 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 production

$$[A_0B] \to aA_1bc$$

Then no production 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 5. case $\lessdot$, from $S_1 \to A_0, B \to Bbc$, from the relation $\perp \lessdot b$, and the existence of nonterminal $[A_0B]$, create the production

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

6. Axiom
From $S_1 \to A_0, S_2 \to B$
6(a). Since $[A_0B]$ exists, create production

$$S \to [A_0B]$$

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

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

Grammar normalization deletes productions $A_0 \to aA_1, B \to bc \mid Bbc$. No repeated right parts are present and the homogeneous condition is already met.

## Appendix 3. Proof of closure under Kleene star

**Theorem 2.** Let $G = (V_N, \Sigma, P, S)$ be a FG such that $OPM(G)$ is $\doteq$-acyclic and $L = L(G)$. Then a FG $\widehat{G} = (\widehat{V}_N, \Sigma, \widehat{P}, \widehat{S})$ with $OPM(\widehat{G}) \supseteq OPM(G)$ can be effectively built such that $L(\widehat{G}) = L^*$.

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

### 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 before normalization the grammar contained the rules:

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

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:

$$\{[AB], [CD], E\} \to \alpha, \quad \{[AB]\} \to \beta, \quad \{[CD]\} \to \gamma, \quad \{E\} \to \delta \tag{5}$$

In addition, in order to preserve language equivalence, for any occurrence of a nonterminal $N$ in a righthand part, 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 right parts, a compound nonterminal $N \in V_{N,2}$ may not contain more than one singleton, i.e., $|N \cap V_N| \leq 1$.

The next lemmas are straightforward consequences of disciplined normalization.

**Lemma 5.** *The set of strings generated by grammar $G_2$ starting from a compound nonterminal $N$ is*

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

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

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

In other words the above change in the nonterminal and production 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 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 6.** *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 $A.B$ in $G$ are derivable in $G_2$ by some nonterminal $N$ containing $[AB]$.*

*Iteration* Initially we set the iteration index to $I = 2$.

1. Apply Algorithm 5 to the language $L(G_I).L(G) \cup L(G_I)$, 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 production $\{[NH]\} \to \alpha$ , the new algorithm writes instead the production $Q \to \alpha$ where

$$Q \text{ is } \{[AH], [CH] \mid \text{ for some } B \in V_N, \text{ it is } [AB] \in N, \text{ and } C \in V_N \text{ is in } N\} \quad (6)$$

   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 $N' = \{[AD], [CD], E\}$ 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 (5) 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 production right-hand sides is bounded by the hypothesis that the OPM is $\doteq$-acyclic.

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

**Lemma 7.** *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 occur:*
  - *Nonterminal $[AB]$ was created as left part of a production of type* (6) *without dropping a nonterminal.*

- *Nonterminal $[AB]$ was created in* (6) *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 denote that it generates such substrings that may encompass one or more whole sentences of $L$.
Notice also that Algorithm 5 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; 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 productions could be generated: $\{[AB]\} \to \alpha\gamma\beta$ as the result of collapsing $[[AX]B]$ into $[AB]$ from productions such as

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

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

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

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

To complete the proof we now state the converse lemma.

**Lemma 9.** *$L^+$ is contained in $L(\widehat{G})$.*

*Proof* Consider a string $x = x_1.x_2.\ldots.x_n, x_i \in L$. We want to show that it is possible to parse it according to $\widehat{G}$. Without loss of generality, we assume that $M = \text{OPM}(G)$ is total so that a bottom-up deterministic parsing of any string in $\Sigma^*$ is always possible according to a FG in $C_M$ (as implied by Statement 2).

1) A first pass of parsing (possibly absent) can be 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 $x_{i,p}\delta x_{i,s} \overset{*}{\Rightarrow} x_i$. This pass can be performed by

using productions originally in $G$, up to some possible renaming of nonterminals due to the normalization.

2) A second pass may involve productions generated during the construction of $G_2$ (again, up to some renaming of nonterminals and splitting of their set of righthand sides) in order to apply reductions in the "boundaries" between $x_i$ and $x_{i+1}$, i.e., $\delta \overset{*}{\Rightarrow} x_{i,s}.x_{i+1,p}$. This pass produces a string $\alpha = \alpha_2\alpha_3 \ldots \alpha_n$ such that $\alpha_i \overset{*}{\Rightarrow} x_{i-1,s}.x_{i,p}$, and no further reductions of the previous types 1) and 2) are possible. This means that at least in one case $\alpha_i \doteq \alpha_{i+1}$. Notice that, as a particular case, the parsing could be completed after this phase if, e.g.,

$$S \overset{*}{\Rightarrow} Sx_n \overset{*}{\Rightarrow} Sx_{n-1}.x_n \ldots \overset{*}{\Rightarrow} x_1.x_2 \ldots x_n$$

3) If the above case does not occur, let us first consider the case

3.1) $\ldots \lessdot \alpha_{i,s} \doteq \alpha_{i+1,p} \gtrdot \ldots$, where $\alpha_{i,s}$ is a suffix of $\alpha_i$ and $\alpha_{i+1,p}$ a prefix of $\alpha_{i+1}$, i.e., only two consecutive $\alpha$'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 $x_{i-1}.x_i \in L^2$, its parsing – by using the productions of $G_2$ – must produce some

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

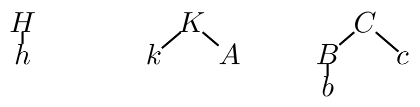
i.e., $\zeta \overset{*}{\Rightarrow} y_i$; notice that no further reduction $N \rightarrow \alpha_{i,s}\zeta_p$, $\zeta_p$ being a *proper* prefix of $\zeta$, would be possible, since, otherwise, the same reduction could be applied even during the second pass above without involving $\alpha_{i+1,p}$.
Consider now the parsing of $x_{i-1}.x_i.x_{i+1}$ according to $G_3$. On the one hand we have reductions (building bottom-up the derivations) $\eta\alpha_{i,s} \overset{*}{\Rightarrow} z_{i-1}.y_{i-1}.z_i$; on the other, $\alpha_{i+1,p}\vartheta \overset{*}{\Rightarrow} y_i.z_{i+1}.y_{i+1}$. Both reductions up to this point have been applied by using productions 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 $N_i = \{[AB], \ldots H, \ldots\}$ 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 $G_2$.

3.1 (a) If both $\zeta$ and $\gamma$ are $\neq \varepsilon$, this implies $\zeta \doteq \gamma$ and the existence in $G$ of a production $C \rightarrow B\gamma$ with $B$ the second symbol of $[AB]$ ($A$ being the same as in $\xi A$). Thus, on the basis of the construction of $G_3$ the production $[[N_h]C] \rightarrow \alpha_{i,s}.\alpha_{i+1,p}$ subsequently transformed into $N_k \rightarrow \alpha_{i,s}.\alpha_{i+1,p}$ is in $G_3$ and the corresponding reduction can be applied. It produces a string

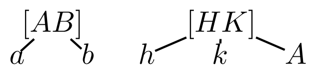
$$\ldots \chi N_k \phi.\alpha_{i+2} \ldots \Rightarrow \chi\alpha_{i,s}\alpha_{i+1,p}\phi.\alpha_{i+2} \overset{*}{\Rightarrow} x_{i-1,s}.x_{i,p}.x_{i,s}.x_{i+1,p}.x_{i+1,s}.x_{i+2,p}$$

The next figure provides an example of the reductions (Case 3.1(a)) applied in the parsing according to $G, G_2$, and $G_3$, respectively.

grammar $G$:

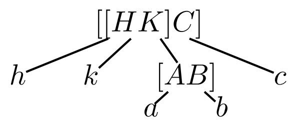


grammar $G_2$:

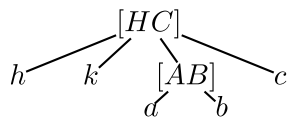


parsing $\lessdot h \doteq k\ [AB]\ c \gtrdot\ e$

grammar of $L(G_2).L(G_1)$:



grammar $G_3$ :



3.1 (b) If instead $\gamma$ is $\varepsilon$ (the case $\xi = \varepsilon$ is symmetric), this means that $B\vartheta \overset{*}{\Rightarrow} z_{i+1}.y_{i+1}$ in $G$ and $\xi \gtrdot \vartheta$. Thus a production $N_h \rightarrow \alpha_{i,s}\xi A$ is in $G_2$ with $[HA_1]$ belonging to $N_h$ and $A_1 \rightarrow \alpha A$ being a production of $G$ for some $A_1, H$. Therefore, again, a production $[[N_h]B] \rightarrow \alpha_{i,s}\alpha_{i+1,p}$, later transformed into $N_k \rightarrow \alpha_{i,s}.\alpha_{i+1,p}$, 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 productions 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 $\alpha$'s, of the type $\ldots \rightarrow \lessdot\alpha_i\doteq\alpha_{i+1}\doteq\ldots\doteq\alpha_{i+k}\gtrdot$ (with no $\lessdot, \gtrdot$ in between) becomes necessary, with

$$\ldots \rightarrow \lessdot\alpha_i\doteq\alpha_{i+1}\doteq\ldots\doteq\alpha_{i+k}\gtrdot \overset{*}{\Rightarrow} x_j.x_{j+1}\ldots x_{j+h}$$

where $j \geq i, h \geq k$.
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 $\alpha_i \overset{*}{\Rightarrow} x_{i-1,s}.x_{i,p}$, etc.
Thus the construction of $G_3$ has built the production $N_k \rightarrow \alpha_{i,s}.\alpha_{i+1,p}$ defined as above (notice: $\alpha_{i+1,p}$ coincides with $\alpha_{i+1}$ since $\alpha_{i+1}$ 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 production $N_s \rightarrow \alpha_{i,s}.\alpha_{i+1,p}.\alpha_{i+2,p}$ (again $\alpha_{i+2,p}$ coincides with $\alpha_{i+2}$) and so on, until producing $N_t \rightarrow \lessdot\alpha_i\doteq\alpha_{i+1}\doteq\ldots\doteq\alpha_{i+k}\gtrdot$. Since at every iteration new productions are generated and since $k$ is bounded by the hypothesis that there is no circularity in the $\doteq$ relation, the production $N_t \rightarrow \lessdot\alpha_i\doteq\alpha_{i+1}\doteq\ldots\doteq\alpha_{i+k}\gtrdot$ 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. $\square$