Cyclic Operator Precedence Grammars for Parallel Parsing

Michele Chiari[3], Dino Mandrioli[1], Matteo Pradella[1,2]

[1] Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Piazza Leonardo Da Vinci 32, 20133 Milano, Italy
[2] IEIIT, Consiglio Nazionale delle Ricerche, via Ponzio 34/5, 20133 Milano, Italy
{dino.mandrioli,matteo.pradella}@polimi.it
[3] Institute of Computer Engineering, TU Wien, Treitlstraße 3, 1040 Vienna, Austria
michele.chiari@tuwien.ac.at

Abstract. Operator precedence languages (OPL) enjoy the local parsability property, which essentially means that a code fragment enclosed within a pair of markers —playing the role of parentheses— can be compiled with no knowledge of its external context. Such a property has been exploited to build parallel compilers for languages formalized as OPLs. It has been observed, however, that when the syntax trees of the sentences have a linear substructure, its parsing must necessarily proceed sequentially making it impossible to split such a subtree into chunks to be processed in parallel. Such an inconvenience is due to the fact that so far much literature on OPLs has assumed the hypothesis that equality precedence relation cannot be cyclic. This hypothesis was motivated by the need to keep the mathematical notation as simple as possible.
We present an enriched version of operator precedence grammars, called cyclic, that allows to use a simplified version of regular expressions in the right hand sides of grammar's rules; for this class of operator precedence grammars the acyclicity hypothesis of the equality precedence relation is no more needed to guarantee the algebraic properties of the generated languages. The expressive power of the cyclic grammars is now fully equivalent to that of other formalisms defining OPLs such as operator precedence automata, monadic second order logic and operator precedence expressions. As a result cyclic operator precedence grammars now produce also unranked syntax trees and sentences with flat unbounded substructures that can be naturally partitioned into chunks suitable for parallel parsing.

Keywords: Operator Precedence Languages · Cyclic Precedence Relations · Parallel Parsing

1 Introduction

Operator precedence languages (OPL) are a "historical" family of languages invented by R. Floyd [8] to support fast deterministic parsing. Together with their *operator precedence grammars (OPG)*, they are still used within modern compilers to parse expressions with operators ranked by priority. The key feature that makes them well amenable for efficient parsing and compilation is that the syntax tree of a sentence is determined exclusively by three binary precedence relations over the terminal alphabet that are easily pre-computed from the grammar productions. For readers unacquainted with OPLs, we anticipate an example: the arithmetic sentence $a + b \times c$ does not make manifest the natural structure $(a + (b \times c))$, but the latter is implied by the fact that the plus operator yields precedence to the times.

Early theoretical investigation [6], originally motivated by grammar inference goals, realized that, thanks to the tree structure assigned to strings by the precedence relations, many closure properties of regular languages hold for OPLs too, but only after a long intermission, renewed research [5] proved further algebraic properties of OPLs. Then, in [13] the *Operator Precedence automata (OPA)* were defined as a formalism, paired with OPGs as well as pushdown automata are paired with context-free grammars (CFG), to formalize the recognition and parsing of OPLs. In the same paper a *monadic second-order (MSO) logic* characterization of OPLs that naturally extends the classic one for regular languages was also produced. Furthermore [15] introduced the *operator precedence expressions (OPE)* which extend traditional regular expressions in the same way as the MSO logic for OPLs extends the traditional one for regular languages. The extension of fundamental algebraic and logic properties of regular languages to OPLs has been completed in [11] where a characterization of OPLs in terms of a syntactic congruence with a finite number of equivalence classes is given.

Another distinguishing property of OPLs is their *local parsability*, i.e. the fact that a chunk of text included within a pair of symmetric precedence relations can be deterministically parsed even without knowing its context [3]. This feature was exploited to produce a parallel parser generator which exhibited high performances w.r.t. traditional sequential parsers [2].

A different research branch —which however is not the core aspect of this paper— exploited the algebraic and logic properties of this family to extend to it the successful verification techniques based on model checking, with comparable performances with those obtained for regular languages [4]. In summary, this historical family of languages enjoys properties that allowed relevant modern applications such as automatic verification and parallel compilation.

It must be pointed out, however, that many —not all— of the algebraic properties discovered during such a research activity that lasted several decades were proved by assuming a hypothesis on the precedence relations defined on the input alphabet. Although from a theoretical point of view this hypothesis slightly affects the generative power of OPGs —but is not necessary, e.g., for OPAs and MSO logic so that these two formalisms are a little more powerful than OPGs—, so far no practical limitation due to it was discovered in terms of formalizing the syntax of real-life programming and data description languages. Thus, it has been constantly adopted in the various developments to avoid making the mathematical notation and technical details too cumbersome.

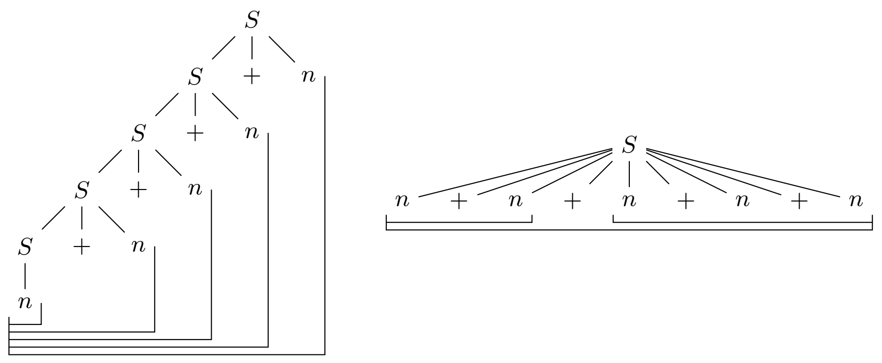


Fig. 1. Left-associative syntax tree (left) vs equal-level one (right) of the plus operator. The left syntax tree imposes a sequential left-to-right parsing and semantic processing whereas the right one can be split onto several branches to be partially processed independently and further aggregated.

The recent contribution [12], however, pointed out a weakness of the parallel compilation algorithm described in [2] which in some cases hampers partitioning the input to be parsed in well-balanced chunks, so that the benefits of parallelism are affected. Intuitively, the weakness is due to the fact that the "normal" precedence relation on the arithmetic operator $+$ compels to parse a sequence thereof by associating them either to the left or to right so that parsing becomes necessarily sequential in this case. The authors also proposed a special technique to overtake this difficulty by allowing for an acceptable level of ambiguity, which in the case of OPGs determines a conflict for some precedence relations on the terminal alphabet.

Such normal precedence relation, however, which either let $+$ yield precedence to, or take precedence over itself, has no correspondence with the semantics of the operation whose result does not depend on the order of association. So, why not giving the various occurrences of the $+$ operator the same precedence level as suggested by arithmetic's laws?[4] Figure 1 gives an intuitive idea of the different structures given to a sequence of $+$ operators by traditional OPGs and the natural semantics of the sum operation. More precise explanations will be given in the following sections.

The answer to this question comes exactly from the above mentioned hypothesis: it forbids cyclic sequences of symbols that are at the same level of precedence; so that $+$ cannot be at the same level of itself. What was a, deemed irrelevant, theoretical limit hid a conceptual inadequacy which had not practical impact until OPLs properties have been exploited to build parallel compilers [5] We felt

[4] Of course, assuming that the use of parentheses does not alter the normal precedence between the various operators.

[5] It must also be recalled that in programming languages very long arithmetic expressions are quite unusual; not so in data description languages such JSON.

therefore "compelled" to finally remove this relatively disturbing hypothesis: this is the object of the present paper.

As often claimed in previous literature, when discussing the impact of forbidding cycles of operators all at the same level of precedence, removing such a restriction requires allowing grammar *right hand sides (rhs) to include the Kleene * operator* (or the $^+$ one which excludes the empty string from its result). Thus, we introduce the *Cyclic operator-precedence grammars (C-OPG)* which include the above feature: whereas such a feature is often used in general context-free grammars to make the syntax of programming languages more compact but does not increase their expressive power, we show that C-OPGs are now fully equivalent to OPAs and other formalisms to define OPLs, such as the MSO logic. We also show that all results previously obtained under the above hypothesis still hold by using C-OPGs instead of traditional OPGs. Although the goal of this paper is not to develop parallel compilation algorithms rooted in C-OPGs, we show how they naturally overtake the difficulty pointed out by [12] and would allow to revisit their techniques, or to improve the efficiency of our previous parallel parser [2].

2 Background

We assume some familiarity with the classical literature on formal language and automata theory, e.g., [18,10]. Here, we just list and explain our notations for the basic concepts we use from this theory. The terminal alphabet is usually denoted by Σ, and the empty string is ε. For a string, or set, x, $|x|$ denotes the length, or the cardinality, of x. The character $\#$, not present in the terminal alphabet, is used as string *delimiter*, and we define the alphabet $\Sigma_\# = \Sigma \cup \{\#\}$.

2.1 Regular languages: automata, regular expressions, logic

A *finite automaton* (FA) $\mathcal{A}$ is defined by a 5-tuple $(Q, \Sigma, \delta, I, F)$ where Q is the set of states, δ the *state-transition relation* (or its *graph* denoted by $\longrightarrow$), $\delta \subseteq Q \times \Sigma \times Q$; I and F are the nonempty subsets of Q respectively comprising the initial and final states. If the tuple (q, a, q') is in the relation δ, the edge $q \xrightarrow{a} q'$ is in the graph. The transitive closure of the relation is defined as usual. Thus, for a string $x \in \Sigma^*$ such that there is a path from state q to q' labeled with x, the notation $q \xrightarrow{x} q'$ is equivalent to $(q, x, q') \in \delta^*$; if $q \in I$ and $q' \in F$, then the string x is *accepted* by $\mathcal{A}$, and $L(\mathcal{A})$ is the language of the accepted strings.

A *regular expression* (RE) over an alphabet Σ is a well-formed formula made with the characters of Σ, $\emptyset$, ε, the Boolean operators $\cup, \neg, \cap$, the concatenation $\cdot$, and the Kleene star operator *. We also use the operator $^+$, and we assume that its scope is always marked by parentheses. An RE E defines a language over Σ, denoted by $L(E)$, in the natural way.

Finite automata and regular expressions define the same family of languages named *regular* (or rational) languages (REG).

2.2 Grammars

Definition 1 (Grammar and language). *A* context-free (CF) grammar *is a tuple $G = (\Sigma, V_N, P, S)$ where Σ and V_N, with $\Sigma \cap V_N = \emptyset$, are resp. the terminal and the nonterminal alphabets, the total alphabet is $V = \Sigma \cup V_N$, $P \subseteq V_N \times V^*$ is the rule (or production) set, and $S \subseteq V_N$, $S \neq \emptyset$, is the axiom set. For a generic rule, denoted as $A \to \alpha$, where A and α are resp. called the left/right hand sides (lhs / rhs), the following forms are relevant:*

axiomatic	:	$A \in S$
terminal	:	$\alpha \in \Sigma^+$
empty	:	$\alpha = \varepsilon$
renaming	:	$\alpha \in V_N$
operator	:	$\alpha \notin V^* V_N V_N V^*$, *i.e., at least one terminal is interposed between any two nonterminals occurring in* α

A grammar is called backward deterministic *or a BD-grammar (or* invertible*) if $(B \to \alpha, C \to \alpha \in P)$ implies $B = C$.*

If all rules of a grammar are in operator form, the grammar is called an operator grammar *or O-grammar.*

For brevity we give for granted the usual definition of derivation *denoted by the symbols $\underset{G}{\Longrightarrow}$ (immediate derivation), $\overset{*}{\underset{G}{\Longrightarrow}}$ (reflexive and transitive closure of $\underset{G}{\Longrightarrow}$), $\overset{+}{\underset{G}{\Longrightarrow}}$ (transitive closure of $\underset{G}{\Longrightarrow}$), $\overset{m}{\underset{G}{\Longrightarrow}}$ (derivation in m steps); the subscript G will be omitted whenever clear from the context. We give also for granted the notion of* syntax tree (ST)*. As usual, the* frontier *of a syntax tree is the ordered left to right sequence of the leaves of the tree.*

The language *defined by a grammar starting from a nonterminal A is $L_G(A) = \left\{ w \mid w \in \Sigma^*, A \overset{*}{\underset{G}{\Longrightarrow}} w \right\}$.*

We call w a sentence *if $A \in S$. The union of $L_G(A)$ for all $A \in S$ is the language $L(G)$ defined by G.*

Two grammars defining the same language are equivalent*. Two grammars generating the same set of syntax trees are* structurally equivalent*.*

Notation: In the following, *unless otherwise explicitly stated*, lowercase letters at the beginning of the alphabet will denote terminal symbols, lowercase letters at the end of the alphabet will denote strings of terminals, Greek letters at the beginning of the alphabet will denote strings in V^*. Capital letters will be used for nonterminal symbols.

Any grammar can be effectively transformed into an equivalent BD-grammar, and also into an O-grammar [1,10] without renaming rules and without empty rules but possibly a single rule whose lhs is an axiom not otherwise occurring in any other production. *From now on, w.l.o.g., we exclusively deal with O-grammars without renaming and empty rules, with the only exception that, if ε is part of the language, there is a unique empty rule whose lhs is an axiom that does not appear in the rhs of any production.*

2.3 Operator precedence grammars

We define the operator precedence grammars (OPGs) following primarily [14].

Intuitively, operator precedence grammars are O-grammars whose parsing is driven by three *precedence relations*, called *equal*, *yield* and *take*, included in $\Sigma_\# \times \Sigma_\#$. They are defined in such a way that two consecutive terminals of a grammar's rhs —ignoring possible nonterminals in between— are in the equal relation, while the two extreme ones —again, whether or not preceded or followed by a nonterminal— are preceded by a yield and followed by a take relation, respectively; in this way a complete rhs of a grammar rule is identified and can be *reduced* to a corresponding lhs by a typical bottom-up parsing. More precisely, the three relations are defined as follows. Subsequently we show how they can drive the bottom-up parsing of sentences.

Definition 2 ([8]). *Let $G = (\Sigma, V_N, P, S)$ be an O-grammar. Let a, b denote elements in Σ, A, B in V_N, C either an element of V_N or the empty string ε, and α, β range over V^*. The left and right terminal sets of nonterminals are respectively:*

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

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

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

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

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

Consider a square matrix: $M = \{M_{a,b} \subseteq \{\doteq, \lessdot, \gtrdot\} \mid a, b \in \Sigma_\#\}$. Such a matrix is called *conflict-free* iff $\forall a, b \in \Sigma_\#$, $0 \le |M_{a,b}| \le 1$. A conflict-free matrix is called *total* or *complete* iff $\forall a, b \in \Sigma_\#$, $|M_{a,b}| = 1$. By convention, if $M_{\#,\#}$ is not empty, $M_{\#,\#} = \{\doteq\}$. A matrix is $\doteq$*-acyclic* if the transitive closure of the $\doteq$ relation over $\Sigma \times \Sigma$ is irreflexive.

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

Definition 3 (Operator precedence grammar). *A grammar G is an* operator precedence grammar *(OPG) iff the matrix $OPM(G)$ is conflict-free, i.e. the three OP relations are disjoint. An OPG is* $\doteq$-acyclic *if $OPM(G)$ is so. An* operator precedence language (OPL) *is a language generated by an OPG.*

Figure 2 (left) displays an OPG, G_{AE}, which generates simple, unparenthesized arithmetic expressions and its OPM (center). The left and right terminal sets of G_{AE}'s nonterminals E, T and F are, respectively: $\mathcal{L}(E) = \{+, \times, n\}$, $\mathcal{L}(T) = \{\times, n\}$, $\mathcal{L}(F) = \{n\}$, $\mathcal{R}(E) = \{+, \times, n\}$, $\mathcal{R}(T) = \{\times, n\}$, and $\mathcal{R}(F) = \{n\}$.
Remark. If the relation $\doteq$ is acyclic, then the length of the rhs of any rule of G is bounded by the length of the longest $\doteq$-chain in $OPM(G)$.

$$\begin{aligned} G_{AE} : S &= \{E\} \\ E &\to E + T \mid T \times F \mid n \\ T &\to T \times F \mid n \\ F &\to n \end{aligned}$$

	$+$	$\times$	n	$\#$
$+$	$\gtrdot$	$\lessdot$	$\lessdot$	$\gtrdot$
$\times$	$\gtrdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
n	$\gtrdot$	$\gtrdot$		$\gtrdot$
$\#$	$\lessdot$	$\lessdot$	$\lessdot$	

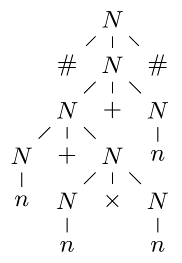


Fig. 2. G_{AE} (left), its OPM (center), and the syntax tree of $n + n \times n + n$ according to the OPM (right).

Unlike the arithmetic relations having similar typography, the OP relations do not enjoy any of the transitive, symmetric, reflexive properties. We kept the original Floyd's notation but we urge the reader not to be confused by the similarity of the two notations.

It is known that the family of OPLs is strictly included within the deterministic and reverse-deterministic CF family, i.e., the languages that can be deterministically parsed both from left to right and from right to left.

The key feature of OPLs is that a conflict-free OPM M defines a universe of *strings compatible with* M and associates to each of them a unique *syntax tree* whose internal nodes are unlabeled and whose leaves are elements of Σ, or, equivalently, a unique parenthesization. We illustrate such a feature through a simple example and refer the reader to previous literature for a thorough description of OP parsing [9,14].

Example 4. Consider the $OPM(G_{AE})$ of Figure 2 and the string $n + n \times n + n$. Display all precedence relations holding between consecutive terminal characters, *including the relations with the delimiters* $\#$ as shown below:

$$\# \lessdot n \gtrdot + \lessdot n \gtrdot \times \lessdot n \gtrdot + \lessdot n \gtrdot \#$$

each pair $\lessdot, \gtrdot$ (with no further $\lessdot, \gtrdot$ in between) includes a *possible* rhs of a production of *any OPG* sharing the OPM with G_{AE}, not necessarily a G_{AE} rhs. Thus, as it happens in typical bottom-up parsing, we replace —*possibly in parallel*— each string included within the pair $\lessdot, \gtrdot$ with a *dummy nonterminal* N; this is because nonterminals are irrelevant for OPMs. The result is the string $\#N + N \times N + N\#$. Next, we compute again the precedence relation between consecutive terminal characters by *ignoring nonterminals*: the result is $\# \lessdot N + \lessdot N \times N \gtrdot +N \gtrdot \#$.

This time, there is only one pair $\lessdot, \gtrdot$ including a potential rhs determined by the OPM (the fact that the external $\lessdot$ and $\gtrdot$ "look matched" is coincidental as it can be easily verified by repeating the previous procedure with the string $n + n \times n + n + n$). Again, we replace the pattern $N \times N$, with the dummy nonterminal N; notice that there is no doubt about associating the two N to the $\times$ rather than to one of the adjacent $+$ symbols: if we replaced, say, just the $\times$ with an N we would obtain the string $N + NNN + N$ which cannot be derived by an O-grammar. By recomputing the precedence relations we obtain the string $\# \lessdot N + N \gtrdot +N \gtrdot \#$. Finally, by applying twice the replacing of $N + N$ by N we obtain $\#N\#$.

The result of the whole bottom-up reduction procedure is synthetically represented by the *syntax tree* of Figure 2 (right) which shows the precedence of the multiplication operation over the additive one in traditional arithmetics. It also suggests a natural association to the left of both operations: if we reverted the order of the rhs of the rules rewriting E and T, the structure of the tree would have suggested associativity to the right of both operations which would not have altered the semantics of the two operations which can indifferently be associated to the left and to the right; not so is we dealt with, say, subtraction or division which instead *impose* association to the left.

Notice that the tree of Figure 2 has been obtained —uniquely and deterministically— by using exclusively the OPM, not the grammar G_{AE} although the string $n + n \times n + n \in L(G_{AE})$ [6].

Obviously, all sentences of $L(G_{AE})$ can be given a syntax tree by $OPM(G_{AE})$, but there are also strings in Σ^* that can be parsed according to the same OPM but are not in $L(G_{AE})$. E.g., the string $+++$ is parsed according to the $OPM(G_{AE})$ as a ST that associates the $+$ characters to the left. Notice also that, in general, not every string in Σ^* is assigned a syntax tree by an OPM; e.g., in the case of $OPM(G_{AE})$ the parsing procedure applied to nn is immediately blocked since there is no precedence relation between n and itself.

The following definition synthesizes the concepts introduced by Example 4.

Definition 5 (OP-alphabet and Maxlanguage).

- *A string in Σ^* is* compatible *with an OPM M iff the procedure described in Example 4 terminates by producing the pattern $\#N\#$. The set of all strings compatible with an OPM M is called the* maxlanguage *or the* universe *of M and is simply denoted as $L(M)$.*
- *Let M be a conflict-free OPM over $\Sigma_\# \times \Sigma_\#$. We use the same identifier M to denote the —partial— function M that assigns to strings in Σ^* their unique ST as informally illustrated in Example 4.*

[6] The above procedure that led to the syntax tree of Figure 2 could be easily adapted to become an algorithm that produces a new syntax tree whose internal nodes are labeled by G_{AE}'s nonterminals. Such an algorithm could be made deterministic by transforming G_{AE} into an equivalend BD grammar (sharing the same OPM).

- *The pair (Σ, M) where M is a conflict-free OPM over $\Sigma_\# \times \Sigma_\#$, is called an* OP-alphabet*. We introduce the concept of OP-alphabet as a pair to emphasize that it defines a universe of strings on the alphabet Σ —not necessarily covering the whole Σ^*— and implicitly assigns them a structure univocally determined by the OPM, or, equivalently, by the function M.*
- *The class of (Σ, M)-*compatible *OPGs and OPLs are:*

$$\mathscr{G}_M = \{G \mid G \text{ is an OPG and } OPM(G) \subseteq M\}, \quad \mathscr{L}_M = \{L(G) \mid G \in \mathscr{G}_M\}.$$

Various formal properties of OPGs and OPLs are documented in the literature, e.g., in [6,5,14]. The next proposition recalls those that are relevant for this article.

Proposition 6 (Algebraic properties of OPGs and OPLs).

1. *If an OPM M is total, then the corresponding homonymous function is total as well, i.e., $L(M) = \Sigma^*$.*
2. *Let (Σ, M) be an OP-alphabet where M is $\doteq$-acyclic. The class $\mathscr{G}_M$ contains an OPG, called the* maxgrammar *of M, denoted by $G_{max,M}$, which generates the maxlanguage $L(M)$. For all grammars $G \in \mathscr{G}_M$, $L(G) \subseteq L(M)$.*
3. *The closure properties of the family $\mathscr{L}_M$ of (Σ, M)-compatible OPLs defined by a total OPM are the following:*
   - *$\mathscr{L}_M$ is closed under union, intersection and set-difference, therefore also under complement (if a maxgrammar of M exists).*
   - *$\mathscr{L}_M$ is closed under concatenation.*
   - *if M is $\doteq$-acyclic, $\mathscr{L}_M$ is closed under Kleene star.*

Remark. Thanks to the fact that a conflict-free OPM assigns to each string at most one ST —and exactly one if the OPM is complete— the above closure properties of OPLs w.r.t. Boolean operations automatically extend to sets of their STs. The same does not apply to the case of concatenation which in general may produce significant reshaping of the original STs [5]. Furthermore, any complete, conflict-free, $\doteq$-acyclic OPM defines a *universe of STs* whose frontiers are Σ^*.

2.4 Chains and Operator Precedence Automata (OPA)

The notion of *chain* introduced next is an alternative way to represent STs where internal nodes are irrelevant and "anonymized".

Definition 7 (Chains). *Let (Σ, M) be an OP-alphabet.*

- *A* simple chain *is a word $a_0 a_1 a_2 \dots\ a_n a_{n+1}$, written as ${}^{a_0}[a_1 a_2 \dots a_n]^{a_{n+1}}$, such that: $a_0, a_{n+1} \in \Sigma \cup \{\#\}$, $a_i \in \Sigma$ for every $i : 1 \leq i \leq n$, $M_{a_0 a_{n+1}} \neq \emptyset$, and $a_0 \lessdot a_1 \doteq a_2 \dots a_{n-1} \doteq a_n \gtrdot a_{n+1}$.*
- *A* composed chain *is a word $a_0 x_0 a_1 x_1 a_2 \dots a_n x_n a_{n+1}$, with $x_i \in \Sigma^*$, where ${}^{a_0}[a_1 a_2 \dots a_n]^{a_{n+1}}$ is a simple chain, and either $x_i = \varepsilon$ or ${}^{a_i}[x_i]^{a_{i+1}}$ is a chain (simple or composed), for every $i : 0 \leq i \leq n$. Such a composed chain will be written as ${}^{a_0}[x_0 a_1 x_1 a_2 \dots a_n x_n]^{a_{n+1}}$.*
- *The body of a chain ${}^{a}[x]^{b}$, simple or composed, is the word x.*

– *Given a chain* ${}^a[x]^b$ *the* depth $d(x)$ *of its body* x *is defined recursively:* $d(x) = 1$ *if the chain is simple, whereas* $d(x_0 a_1 x_1 \dots a_n x_n) = 1 + \max_i d(x_i)$*. The depth of a chain is the depth of its body.*

For instance, the ST of Figure 2 (right) is biunivocally represented by the composed chain ${}^\#[x_0 + x_1]^\#$, where, in turn x_0 is the body of the composed chain ${}^\#[y_0 + y_1]^+$, y_0 is the body of the simple chain ${}^\#[e]^+$, y_1 is the body of the composed chain ${}^+[z_0 \times z_1]^+$, etc. The depth of the main chain is 5.

As well as an OPG selects a set of STs within the universe defined by its OPM, an *operator precedence automaton (OPA)* selects a set of chains within the universe defined by an OP-alphabet.

Definition 8 (Operator precedence automaton). *A nondeterministic* operator precedence automaton (OPA) *is given by a tuple:* $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ *where:*

– (Σ, M) *is an operator precedence alphabet,*
– Q *is a set of states (disjoint from* Σ*),*
– $I \subseteq Q$ *is a set of initial states,*
– $F \subseteq Q$ *is a set of final states,*
– δ*, named transition function, is a triple of functions:*

$$\delta_{shift} : Q \times \Sigma \to \wp(Q) \qquad \delta_{push} : Q \times \Sigma \to \wp(Q) \qquad \delta_{pop} : Q \times Q \to \wp(Q)$$

We represent a nondeterministic OPA by a graph with Q as the set of vertices and $\Sigma \cup Q$ as the set of edge labelings. The edges of the graph are denoted by different shapes of arrows to distinguish the three types of transitions: there is an edge from state q to state p labeled by $a \in \Sigma$ denoted by a dashed (respectively, normal) arrow if and only if $p \in \delta_{\text{shift}}(q, a)$ (respectively, $p \in \delta_{\text{push}}(q, a)$) and there is an edge from state q to state p labeled by $r \in Q$ and denoted by a double arrow if and only if $p \in \delta_{\text{pop}}(q, r)$.

To define the semantics of the automaton, we introduce some notations.

We use letters $p, q, p_i, q_i, \dots$ to denote states in Q. Let Γ be $\Sigma \times Q$ and let Γ' be $\Gamma \cup \{\bot\}$; we denote symbols in Γ' as $[a,\ q]$ or $\bot$. We set $symbol([a,\ q]) = a$, $symbol(\bot) = \#$, and $state([a,\ q]) = q$. Given a string $\Pi = \bot \pi_1 \pi_2 \dots \pi_n$, with $\pi_i \in \Gamma$, $n \geq 0$, we set $symbol(\Pi) = symbol(\pi_n)$, including the particular case $symbol(\bot) = \#$.

A *configuration* of an OPA is a triple $C = \langle \Pi,\ q,\ w \rangle$, where $\Pi \in \bot \Gamma^*$, $q \in Q$ and $w \in \Sigma^* \#$. The first component represents the contents of the stack, the second component represents the current state of the automaton, while the third component is the part of input still to be read.

A *computation* or *run* of the automaton is a finite sequence of *moves* or *transitions* $C_1 \vdash C_2$; there are three kinds of moves, depending on the precedence relation between the symbol on top of the stack and the next symbol to read:

push move: if $symbol(\Pi) \lessdot\ a$ then $\langle \Pi,\ p,\ ax \rangle \vdash \langle \Pi[a,\ p],\ q,\ x \rangle$, with $q \in \delta_{\text{push}}(p, a)$;

shift move: if $a \doteq b$ then $\langle \Pi[a,\ p],\ q,\ bx \rangle \vdash \langle \Pi[b,\ p],\ r,\ x \rangle$, with $r \in \delta_{\text{shift}}(q, b)$;

pop move: if $a \gtrdot b$ then $\langle \Pi[a,\ p],\ q,\ bx\rangle \vdash \langle \Pi,\ r,\ bx\rangle$, with $r \in \delta_{\text{pop}}(q,p)$.

Notice that shift and pop moves are never performed when the stack contains only $\bot$.

Push and shift moves update the current state of the automaton according to the transition function δ_{push} and δ_{shift}, respectively: push moves put a new element on the top of the stack consisting of the input symbol together with the current state of the automaton, whereas shift moves update the top element of the stack by changing its input symbol only. The pop move removes the symbol on the top of the stack, and the state of the automaton is updated by δ_{pop} on the basis of the pair of states consisting of the current state of the automaton and the state of the removed stack symbol; notice that in this move the input symbol is used only to establish the $\gtrdot$ relation and it remains available for the following move.

A configuration $\langle \bot,\ q_I,\ x\#\rangle$ is *initial* if $q_I \in I$; a configuration $\langle \bot,\ q_F,\ \#\rangle$ is *accepting* if $q_F \in F$. The language accepted by the automaton is:

$$L(\mathcal{A}) = \left\{ x \mid \langle \bot,\ q_I,\ x\#\rangle \overset{*}{\vdash} \langle \bot,\ q_F,\ \#\rangle, q_I \in I, q_F \in F \right\}.$$

Example 9. The OPA depicted in Figure 3 (top, left) based on the OPM at the (top, right) accepts the language of arithmetic expressions enriched w.r.t $L(G_{AE})$ in that it introduces the use of explicit parentheses to alter the natural precedence of arithmetic operations. The same figure (bottom) also shows an accepting computation on input $n + n \times (\!|n + n|\!)$.

Definition 10. *Let $\mathcal{A}$ be an OPA. A* support *for a simple chain* $^{a_0}[a_1 a_2 \dots a_n]^{a_{n+1}}$ *is any path in $\mathcal{A}$ of the form*

$$q_0 \xrightarrow{a_1} q_1 \dashrightarrow \dots \dashrightarrow q_{n-1} \overset{a_n}{\dashrightarrow} q_n \overset{q_0}{\Longrightarrow} q_{n+1} \qquad (1)$$

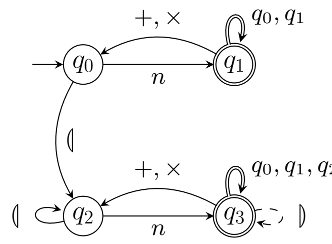
Notice that the label of the last (and only) pop is exactly q_0, i.e. the first state of the path; this pop is executed because of relations $a_0 \lessdot a_1$ and $a_n \gtrdot a_{n+1}$.
A support for the composed chain $^{a_0}[x_0 a_1 x_1 a_2 \dots a_n x_n]^{a_{n+1}}$ *is any path in $\mathcal{A}$ of the form*

$$q_0 \overset{x_0}{\rightsquigarrow} q'_0 \xrightarrow{a_1} q_1 \overset{x_1}{\rightsquigarrow} q'_1 \overset{a_2}{\dashrightarrow} \dots \overset{a_n}{\dashrightarrow} q_n \overset{x_n}{\rightsquigarrow} q'_n \overset{q'_0}{\Longrightarrow} q_{n+1} \qquad (2)$$

where for every $i : 0 \le i \le n$:

- *if $x_i \neq \varepsilon$, then $q_i \overset{x_i}{\rightsquigarrow} q'_i$ is a support for the (simple or composed) chain* $^{a_i}[x_i]^{a_{i+1}}$
- *if $x_i = \varepsilon$, then $q'_i = q_i$.*

Notice that the label of the last pop is exactly q'_0.
The support of a chain with body x will be denoted by $q_0 \overset{x}{\rightsquigarrow} q_{n+1}$.



	$+$	$\times$	$\llparenthesis$	$\rrparenthesis$	n	$\#$
$+$	$\gtrdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
$\times$	$\gtrdot$	$\gtrdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
$\llparenthesis$	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$	$\lessdot$	
$\rrparenthesis$	$\gtrdot$	$\gtrdot$		$\gtrdot$		$\gtrdot$
n	$\gtrdot$	$\gtrdot$		$\gtrdot$		$\gtrdot$
$\#$	$\lessdot$	$\lessdot$	$\lessdot$		$\lessdot$	

stack	state	current input
$\bot$	q_0	$n + n \times \llparenthesis n + n \rrparenthesis \#$
$\bot[n,\ q_0]$	q_1	$+n \times \llparenthesis n + n \rrparenthesis \#$
$\bot$	q_1	$+n \times \llparenthesis n + n \rrparenthesis \#$
$\bot[+,\ q_1]$	q_0	$n \times \llparenthesis n + n \rrparenthesis \#$
$\bot[+,\ q_1][n,\ q_0]$	q_1	$\times \llparenthesis n + n \rrparenthesis \#$
$\bot[+,\ q_1]$	q_1	$\times \llparenthesis n + n \rrparenthesis \#$
$\bot[+,\ q_1][\times,\ q_1]$	q_0	$\llparenthesis n + n \rrparenthesis \#$
$\bot[+,\ q_1][\times,\ q_1][\llparenthesis,\ q_0]$	q_2	$n + n \rrparenthesis \#$
$\bot[+,\ q_1][\times,\ q_1][\llparenthesis,\ q_0][n,\ q_2]$	q_3	$+n \rrparenthesis \#$
$\bot[+,\ q_1][\times,\ q_1][\llparenthesis,\ q_0]$	q_3	$+n \rrparenthesis \#$
$\bot[+,\ q_1][\times,\ q_1][\llparenthesis,\ q_0][+,\ q_3]$	q_2	$n \rrparenthesis \#$
$\bot[+,\ q_1][\times,\ q_1][\llparenthesis,\ q_0][+,\ q_3][n,\ q_2]$	q_3	$\rrparenthesis \#$
$\bot[+,\ q_1][\times,\ q_1][\llparenthesis,\ q_0][+,\ q_3]$	q_3	$\rrparenthesis \#$
$\bot[+,\ q_1][\times,\ q_1][\llparenthesis,\ q_0]$	q_3	$\rrparenthesis \#$
$\bot[+,\ q_1][\times,\ q_1][\rrparenthesis,\ q_0]$	q_3	$\#$
$\bot[+,\ q_1][\times,\ q_1]$	q_3	$\#$
$\bot[+,\ q_1]$	q_3	$\#$
$\bot$	q_3	$\#$

Fig. 3. An OPA (top, left), its OPM (top, right) and an example of computation for the language of Example 9 (bottom). Arrows $\longrightarrow$, $\dashrightarrow$ and $\Longrightarrow$ denote push, shift and pop transitions, respectively. To avoid confusion with the overloaded parenthesis symbols, the parentheses used as terminal symbols are denoted as $\llparenthesis$ and $\rrparenthesis$.

Notice that the context a, b of a chain ${}^a[x]^b$ is used by the automaton to build its support only because $a \lessdot x$ and $x \gtrdot b$; thus, the chain's body contains all information needed by the automaton to build the subtree whose frontier is that string, once it is understood that its first move is a push and its last one is pop. This is a distinguishing feature of OPLs, not shared by other deterministic languages: we call it the *locality principle* of OPLs, which has been exploited to build parallel and/or incremental OP parsers [3].

3 Cyclic Operator Precedence Grammars (C-OPGs) and their equivalence with OPAs

Proposition 6 shows that *some, but not all*, of the algebraic properties of OPLs depend critically on the $\doteq$-acyclicity hypothesis. This is due to the fact that without such a hypothesis the rhs of an OPG have an unbounded length but cannot be infinite: e.g., no OPG can generate the language $\{a, b\}^*$ if $a \doteq b$ and $b \doteq a$. In most cases cycles of this type can be "broken" as it has been done up to now, e.g., to avoid the $+ \doteq +$ relation in arithmetic expressions by associating the operator indifferently to the right or to left. Thus, although it is known that, from a theoretical point of view, the $\doteq$-acyclicity hypothesis affects the expressive power of OPGs,[7] we accepted so far the $\doteq$-acyclicity hypothesis to keep the notation as simple as possible. Recently, however, it has been observed [12] that such a restriction may hamper the benefits achievable by the parallel compilation techniques that exploit the local parsability property of OPLs [2]. Thus, it is time to introduce the necessary extension of OPGs so that the $\doteq$-acyclicity hypothesis can be avoided and they become fully equivalent to other formalisms to define OPLs.

Definition 11 (Cyclic Operator Precedence Grammar).

- *A* $^+$*-O-expression on* V^* *is an expression obtained from the elements of* V *by iterative application of concatenation and the usual* $^+$ *operator* [8]*, provided that any substring thereof has no two adjacent nonterminals; for convenience, and w.l.o.g., we assume that all subexpressions that are argument of the* $^+$ *operator are terminated by a terminal character.*
- *A Cyclic O-grammar (C-OG) is an O-grammar whose production rhs are* $^+$*-O-expressions.*
- *For a grammar rule* $A \to \alpha$ *of an C-OG, the* $\underset{G}{\Longrightarrow}$ *(immediate derivation) relation is defined as* $\beta A \gamma \Longrightarrow \beta\zeta\gamma$ *iff* ζ *is a string belonging to the language defined by the* $^+$*-O-expression* α*,* $L(\alpha)$*.*
- *The equal in precedence relation is redefined as* $a \doteq b$ *iff* $\exists A \to \alpha \wedge \exists \zeta = \eta a B b \theta \mid (B \in V_N \cup \{\varepsilon\} \wedge \zeta \in L(\alpha))$*. The other precedence relations remain defined as for non-cyclic O-grammars.*
- *A C-OG is a cyclic operator precedence grammar (C-OPG) iff its OPM is conflict-free.*

As a consequence of the definition of the immediate derivation relation for C-OPGs the syntax-trees derived therefrom can be unranked, i.e., their internal nodes may have an unbounded number of children.

Example 12. The C-OPG depicted in Figure 4 with its OPM generates a fairly complete language of —not necessarily fully— parenthesized arithmetic expressions involving the four basic operations: as usual the multiplicative operations take precedence over the additive ones; furthermore subtraction takes precedence over sum and division over multiplication. The key novelty w.r.t. the traditional way of formalizing arithmetic expressions by means of OPGs are the $+ \doteq +$ and $\times \doteq \times$ OP relations; on the contrary we kept the structure that associates subtraction and division to the left, so that the grammar's STs —an example thereof is given in Figure 5— now fully reflect the semantics of the arithmetic operations.

[7] The language $\{a^n(bc)^n\} \cup \{b^n(ca)^n\} \cup \{c^n(ab)^n\} \cup (abc)^+$ cannot be generated by an OPG because the $a \doteq b \doteq c \doteq a$ relations are necessary [7].

[8] For our purposes $^+$ is more convenient than * without affecting the generality.

$G_{\text{A-AE}}$:

$S = \{P, T, M, N, F, D, E\}$
$P \to (T+)^+T$
$T \to (F\times)^+F \mid M - N \mid D/E$
$M \to M - N \mid (F\times)^+F \mid D/E$
$N \to (F\times)^+F \mid D/E$
$F \to D/E$
$D \to D/E$
$\{P, T, M, N, F, D, E\} \to \llparenthesis\{P, T, M, N, F, D, E\}\rrparenthesis \mid n$

	$+$	$-$	$\times$	$/$	$\llparenthesis$	$\rrparenthesis$	n	$\#$
$+$	$\doteq$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
$-$	$\gtrdot$	$\gtrdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
$\times$	$\gtrdot$	$\gtrdot$	$\doteq$	$\lessdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
$/$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
$\llparenthesis$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$	$\lessdot$	
$\rrparenthesis$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$		$\gtrdot$		$\gtrdot$
n	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$		$\gtrdot$		$\gtrdot$
$\#$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$		$\lessdot$	

Fig. 4. A C-OPG (left) and its OPM (right). The notation $\{P, T, M, F, D, E\} \to \llparenthesis\{P, T, M, N, F, D, E\}\rrparenthesis$ is an abbreviation meaning that anyone of the nonterminals at the left can be rewritten as a pair of parentheses enclosing anyone of the same nonterminals.

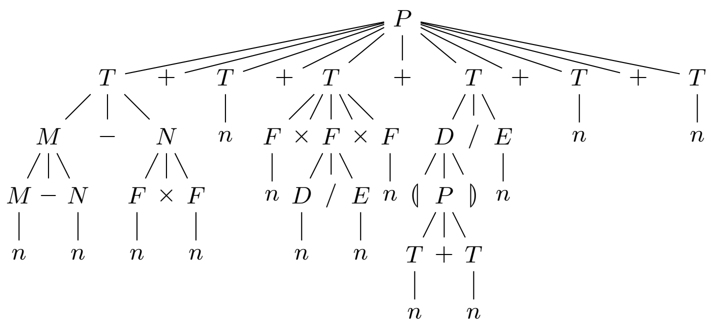


Fig. 5. A ST generated by the C-OPG of Figure 4.

Notice that $G_{\text{A-AE}}$ is purposely ambiguous to keep it compact. To support deterministic parsing it should be transformed into a BD —and therefore unambiguous— form.

By looking at the ST of Figure 5 and comparing it with the original Figure 1, one can get a better envision of why the introduction of cyclic $\doteq$ can support more effective parallel compilation algorithms for OPLs. Parallel parsing and compilation for OPLs is rooted in the *local parsability property* of this family: thanks to this property any fragment of input string enclosed within a pair of corresponding $\lessdot$ and $\gtrdot$ OP relations can be processed in parallel with other similar fragments. However, if, say, the $+$ operator is associated to the left (or right) as in the case of the left ST of Figure 1 the parsing of a sequence of $+$ must necessarily proceed sequentially from left to right. Conversely, if the ST has a structure like that of the right tree of Figure 1 the sequence of $+$ —whether intermixed or not with other subtrees— can be arbitrarily split into several branches which can be parsed and compiled in parallel and, after that, can be joined into a unique subtree as imposed by the $\doteq$ OP relation between the corresponding extreme terminals of contiguous branches.

3.1 Equivalence between C-OPGs and OPAs

The equivalence is obtained by a slight modification of the analogous proof given in [13] where the additional hypothesis of M being $\doteq$-acyclic was exploited.

First, we describe a procedure to build an OPA equivalent to a C-OPG. Then, we provide the converse construction.

Theorem 13 (From C-OPGs to OPAs). *Let (Σ, M) be an OP-alphabet. For any C-OPG defined thereon an equivalent OPA can be effectively built.*

Proof. A nondeterministic OPA[9] $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ from a given C-OPG G with the same precedence matrix M as G is built in such a way that a successful computation thereof corresponds to building bottom-up a syntax tree of G: the automaton performs a push transition when it reads the first terminal of a new rhs; it performs a shift transition when it reads a terminal symbol inside a rhs, i.e. a leaf with some left sibling leaf. It performs a pop transition when it completes the recognition of a rhs, then it guesses (nondeterministically) the nonterminal at the lhs. Each state contains two pieces of information: the first component represents the prefix of the rhs under construction, whereas the second component is used to recover the rhs *previously under construction* (see Figure 6) whenever all rhs nested below have been completed. Precisely, the construction of $\mathcal{A}$ is defined as follows.

Let $\hat{P}$ be the set of rhs γ where all $^+$ and related parentheses have been erased. Let $\tilde{P}$ be the set of strings $\tilde{\gamma} \in V^+$ belonging to the language of some rhs γ of P that is inductively defined as follows:
if $(\eta)^+$ is a subexpression of γ such that η is a single string $\in V^+$ then $\tilde{\eta} = \{\eta, \eta\eta\}$;

[9] Any nondeterministic OPA can be transformed into a deterministic one at the cost of quadratic exponential increase in the size of the state space [13].

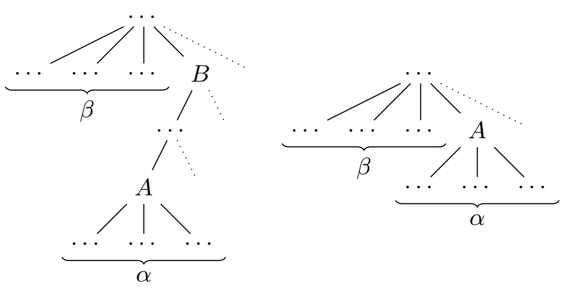


Fig. 6. When parsing α, the prefix previously under construction is β.

if $\eta = \alpha_1(\beta_1)^+\alpha_2(\beta_2)^+ \dots \alpha_n$ where $\alpha_i \in V^*$, then $\tilde{\eta} = \{\eta_1, \eta_1\eta_1\}$ where $\eta_1 = \alpha_1\tilde{\beta}_1\alpha_2\tilde{\beta}_2 \dots \alpha_n$.

E.g., let η be $(Ba(bc)^+)^+$; then $\tilde{\eta} = \{Babc, Babcbc, BabcBabc, BabcbcBabc, BabcBabcbc, BabcbcBabcbc\}$ and $\hat{\eta} = \{Babc\}$

Let $\mathbb{P} = \{\alpha \in V^*\Sigma \mid \exists A \to \eta \in P \land \exists \alpha, \beta \mid \alpha\beta \in \tilde{\eta}\}$ be the set of prefixes, ending with a terminal symbol, of strings $\in \tilde{P}$; define $\mathbb{Q} = \{\varepsilon\} \cup \mathbb{P} \cup N$, $Q = \mathbb{Q} \times (\{\varepsilon\} \cup \mathbb{P})$, $I = \{\langle\varepsilon, \varepsilon\rangle\}$, and $F = S \times \{\varepsilon\} \cup \{\langle\varepsilon, \varepsilon\rangle$ if $\varepsilon \in L(G)\}$. Note that $|\mathbb{Q}| = 1 + |\mathbb{P}| + |N|$ is $O(m^h)$ where m is the maximum length of the rhs in P, and h is the maximum nesting level of $^+$ operators in rhs; therefore $|Q|$ is $O(m^{2h})$.

The transition functions are defined by the following formulas, for $a \in \Sigma$ and $\alpha, \alpha_1, \alpha_2 \in \mathbb{Q}$, $\beta, \beta_1, \beta_2 \in \{\varepsilon\} \cup \mathbb{P}$, and where for any expression ξ, $\bar{\xi}$ is obtained from ξ by erasing parentheses and $^+$ operators:

– $$\delta_{\text{shift}}(\langle\alpha, \beta\rangle, a) \ni \begin{cases} \text{if } \alpha \notin N : & \begin{cases} \langle\bar{\eta}\bar{\zeta}, \beta\rangle & \text{if} \begin{pmatrix} \exists A \to \gamma \mid \gamma = \eta(\zeta)^+\theta \land \\ \alpha a = \bar{\eta}\bar{\zeta}\bar{\zeta}, \alpha a\bar{\theta} \in L(\gamma) \cap \tilde{P} \end{pmatrix} \\ \langle\alpha a, \beta\rangle & \text{otherwise} \end{cases} \\ \text{if } \alpha \in N : & \begin{cases} \langle\bar{\eta}\bar{\zeta}, \beta\rangle & \text{if} \begin{pmatrix} \exists A \to \gamma \mid \gamma = \eta(\zeta)^+\theta \land \\ \beta\alpha a = \bar{\eta}\bar{\zeta}\bar{\zeta}, \alpha a\bar{\theta} \in L(\gamma) \cap \tilde{P} \end{pmatrix} \\ \langle\beta\alpha a, \beta\rangle & \text{otherwise} \end{cases} \end{cases}$$

– $$\delta_{\text{push}}(\langle\alpha, \beta\rangle, a) \ni \begin{cases} \langle a, \alpha\rangle & \text{if } \alpha \notin N \\ \langle\alpha a, \beta\rangle & \text{if } \alpha \in N \end{cases}$$

– $\delta_{\text{pop}}(\langle\alpha_1, \beta_1\rangle, \langle\alpha_2, \beta_2\rangle) \ni \langle A, \gamma\rangle$

for every A such that $\left[\begin{array}{l} \text{if } \alpha_1 \notin N : A \to \alpha \in P \land \alpha_1 \in L(\alpha) \cap \hat{P} \\ \text{if } \alpha_1 \in N : A \to \delta \in P \land \beta_1\alpha_1 \in L(\delta) \cap \hat{P} \end{array}\right.$

and $\gamma = \begin{cases} \alpha_2 \text{ if } \alpha_2 \notin N \\ \beta_2 \text{ if } \alpha_2 \in N. \end{cases}$

The states reached by push and shift transitions have the first component in $\mathbb{P}$. If state $\langle\alpha, \beta\rangle$ is reached after a push transition, then α is the prefix of the rhs (deprived of the $^+$ operators) that is currently under construction and β is the prefix previously under construction; in this case α is either a terminal symbol or a nonterminal followed by a terminal one.

If the state is reached after a shift transition, and the α component of the previous state was not a single nonterminal, then the new α is the concatenation of the first component of the previous state with the read character. If, instead, the α component of the previous state was a single nonterninal —which was produced by a pop transition— then the new α also includes the previous β (see Figure 6) and β is not changed from the previous state. However, if the new α becomes such that a suffix thereof is a double occurrence of a string $\zeta \in L((\zeta)^+)$ —hence $\alpha \in \mathbb{P}$— then the second occurrence of ζ is cut from the new α, which therefore becomes a prefix of an element of $\hat{P}$.

The states reached by a pop transition have the first component in N: if $\langle A, \gamma \rangle$ is such a state, then A is the corresponding lhs, and γ is the prefix previously under construction.

For instance, imagine that a C-OPG contains the rules $A \to (Ba(bc)^+)^+a$ and $B \to h$ and that the corresponding OPA $\mathcal{A}$ parses the string *habcbchabca*: after scanning the prefix *habcb* $\mathcal{A}$ has reduced h to B and has *Babcb* as the first component of its state; after reading the new c it recognizes that the suffix of the first state component would become a second instance of *bc* belonging to $(bc)^+$; thus, it goes back to *Babc*. Then, it proceeds with a new reduction of h to B and, when reading with a shift the second a appends *Ba* to its current β which was produced by the previous pop so that the new α becomes *BabcBa*; after shifting b it reads c and realizes that its new α would become *BabcBabc*, i.e., an element of $(Ba(bc)^+)^+$ and therefore "cuts" it to the single instance thereof, i.e., *Babc*. Finally, after having shifted the last a it is ready for the last pop.

Notice that the result of δ_{shift} and δ_{push} is a singleton, whereas δ_{pop} may produce several states, in case of repeated rhs. Thus, if G is BD, the corresponding $\mathcal{A}$ is deterministic.

The equivalence between G and $\mathcal{A}$ derives from the following Lemmas 14 and 15, when $\beta = \gamma = \varepsilon$, $\Pi = \perp$ and A is an axiom. Their statements are identical to Lemmas 3.2 and 3.3 given in [13] for the original construction applied to non-C-OPGs.

Lemma 14. *Let x be the body of a chain and $\beta, \gamma \in \mathbb{P} \cup \{\varepsilon\}$. Then $\langle \beta, \gamma \rangle \overset{x}{\leadsto} q$ implies the existence of $A \in N$ such that $A \overset{*}{\Rightarrow} x$ in G and $q = \langle A, \beta \rangle$.*

Lemma 15. *Let x be the body of a chain and $A \in N$. Then, $A \overset{*}{\Rightarrow} x$ in G implies $\langle \beta, \gamma \rangle \overset{x}{\leadsto} \langle A, \beta \rangle$ for every $\beta, \gamma \in \mathbb{P} \cup \{\varepsilon\}$.*

The proof of the above lemmas is based on a natural induction on the depth h of the chains visited by the OPA and is not repeated here. We simply point out the only key difference due to the fact that simple chains generated by C-OPGs can have unbounded length.

During a series of shift transitions scanning a simple chain the β component of the state remains unchanged, whereas the α component at each transition appends the read character —always under the constraint that it belongs to $\mathbb{P}$— up to the point where the last append would repeat a suffix of α —say a string y in case of simple chains— identically; if y occurs under the $^+$ operator of a rhs of

stack	state	current input
$\bot$	$\langle \varepsilon, \varepsilon \rangle$	$n + n + n/n/n + n + n\#$
$\bot[n,\ \langle \varepsilon, \varepsilon \rangle]$	$\langle n, \varepsilon \rangle$	$+n + n/n/n + n + n\#$
$\bot$	$\langle T, \varepsilon \rangle$	$+n + n/n/n + n + n\#$
$\bot[+,\ \langle T, \varepsilon \rangle]$	$\langle T+, \varepsilon \rangle$	$n + n/n/n + n + n\#$
$\bot[+,\ \langle T, \varepsilon \rangle][n,\ \langle T+, \varepsilon \rangle]$	$\langle n, T+ \rangle$	$+n/n/n + n + n\#$
$\bot[+,\ \langle T, \varepsilon \rangle]$	$\langle T, T+ \rangle$	$+n/n/n + n + n\#$
$\bot[+,\ \langle T, \varepsilon \rangle]$	$\langle \mathbf{T}+, \mathbf{T}+ \rangle$	$n/n/n + n + n\#$
$\bot[+,\ \langle T, \varepsilon \rangle][n,\ \langle T+, T+ \rangle]$	$\langle n, T+ \rangle$	$/n/n + n + n\#$
$\bot[+,\ \langle T, \varepsilon \rangle]$	$\langle D, T+ \rangle$	$/n/n + n + n\#$
$\bot[+,\ \langle T, \varepsilon \rangle][/,\ \langle D, T+ \rangle]$	$\langle D/, T+ \rangle$	$n/n + n + n\#$
$\bot[+,\ \langle T, \varepsilon \rangle][/,\ \langle D, T+ \rangle][n,\ \langle /, D \rangle]$	$\langle n, D/ \rangle$	$/n + n + n\#$
$\bot[+,\ \langle T, \varepsilon \rangle][/,\ \langle D, T+ \rangle]$	$\langle E, D/ \rangle$	$/n + n + n\#$
$\bot[+,\ \langle T, \varepsilon \rangle]$	$\langle D, T+ \rangle$	$/n + n + n\#$
$\bot[+,\ \langle T, \varepsilon \rangle][/,\ \langle D, T+ \rangle]$	$\langle D/, T+ \rangle$	$n + n + n\#$
$\bot[+,\ \langle T, \varepsilon \rangle][/,\ \langle D, T+ \rangle][n,\ \langle D/, T+ \rangle]$	$\langle n, D/ \rangle$	$+n + n\#$
$\bot[+,\ \langle T, \varepsilon \rangle][/,\ \langle D, T+ \rangle]$	$\langle E, D/ \rangle$	$+n + n\#$
$\bot[+,\ \langle T, \varepsilon \rangle]$	$\langle T, T+ \rangle$	$+n + n\#$
$\bot[+,\ \langle T, \varepsilon \rangle]$	$\langle \mathbf{T}+, \mathbf{T}+ \rangle$	$n + n\#$
$\bot[+,\ \langle T, \varepsilon \rangle][n,\ \langle T+, T+ \rangle]$	$\langle n, T+ \rangle$	$+n\#$
$\bot[+,\ \langle T, \varepsilon \rangle]$	$\langle T, T+ \rangle$	$+n\#$
$\bot[+,\ \langle T, \varepsilon \rangle]$	$\langle \mathbf{T}+, \mathbf{T}+ \rangle$	$n\#$
$\bot[+,\ \langle T, \varepsilon \rangle][n,\ \langle T+, T+ \rangle]$	$\langle n, T+ \rangle$	$\#$
$\bot[+,\ \langle T, \varepsilon \rangle]$	$\langle T, T+ \rangle$	$\#$
$\bot$	$\langle P, \varepsilon \rangle$	$\#$

Fig. 7. A run of the OPA built from the C-OPG of Figure 4 accepting the sentence $n + n + n/n/n + n + n$. The states truncated by erasing a repeated suffix ζ occurring under the scope of a $^+$ operator are emphasized in boldface.

which α is a prefix then the second occurrence of y is erased and the state is the same as after reading its first occurrence, thus closing a loop in the same way as it happens with finite state machines. As a consequence the OPA can repeat the loop reading y any number of times just as the C-OPG can generate, in an immediate derivation, any number of occurrences of y thanks to the $^+$ operator.

The converse statement that if $A \Longrightarrow x$ then $\langle \beta, \gamma \rangle \overset{x}{\rightsquigarrow} \langle A, \beta \rangle$ for every $\beta, \gamma \in \mathbb{P} \cup \{\varepsilon\}$ follows directly from the construction of $\mathcal{A}$.

The generalization of the above reasoning to the case of composed chains perfectly parallels the induction step of the corresponding lemmas of [13].

Example 16. Figure 7 displays a run of the OPA obtained from the C-OPG of Figure 4 accepting the sentence $n + n + n/n/n + n + n$.

The construction of a C-OPG equivalent to a given OPA is far simpler than the converse one, thanks to the explicit structure associated to words by the precedence matrix. The key difference w.r.t. the analogous construction given in [13] is that even simple chains can have unbounded length due to the fact that the $\doteq$ relation may be circular.

First, in analogy with the definition of $\tilde{P}$, we define *essential supports* for both simple and composed chains, as those supports where possible cyclic behaviors of the OPA along a sequence of terminals —whether with interposing nonterminals or not— occur exactly twice.

Definition 17. *(Essential chain supports) An essential support of a simple chain* ${}^{a_0}[a_1 a_2 \dots a_n]^{a_{n+1}}$ *is any path in* $\mathcal{A}$ *of the form*

$$q_0 \xrightarrow{a_1} q_1 \dashrightarrow \dots \dashrightarrow q_{n-1} \overset{a_n}{\dashrightarrow} q_n \overset{q_0}{\Longrightarrow} q_{n+1} \tag{3}$$

where any cycle $q_i a_{i1}...a_{ik} q_i$ *is repeated exactly twice.*
Essential supports for composed chains are defined similarly.

For instance, with reference to the OPA built from the C-OPG of Figure 4 an essential support of the chain ${}^{\#}[n+n+n+n+n]^{\#}$ is:

$$\langle \varepsilon, \varepsilon \rangle \overset{n}{\rightsquigarrow} \langle T, \varepsilon \rangle \xrightarrow{+} \langle T+, \varepsilon \rangle \overset{n}{\rightsquigarrow} \langle T, T+ \rangle \overset{+}{\dashrightarrow} \langle T+, T+ \rangle \overset{n}{\rightsquigarrow} \langle T, T+ \rangle \overset{+}{\dashrightarrow}$$
$$\langle T+, T+ \rangle \overset{n}{\rightsquigarrow} \langle T, T+ \rangle \overset{\langle T, \varepsilon \rangle}{\Longrightarrow} \langle P, \varepsilon \rangle$$

Lemma 18. *The essential supports of simple chains of any OPA have an effectively computable bounded length.*

Theorem 19 (From OPAs to C-OPGs). *Let* (Σ, M) *be an OP-alphabet. For any OPA defined thereon an equivalent C-OPG can be effectively built.*

Proof. Given an OPA $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$, we show how to build an equivalent OPG G having operator precedence matrix M. The equivalence between $\mathcal{A}$ and G should then be rather obvious.

G's nonterminals are the 4-tuples $(a, q, p, b) \in \Sigma \times Q \times Q \times \Sigma$, written as $\langle {}^a p, q^b \rangle$. G's rules are built as follows:

– for every essential support of a simple chain, the rule

$$\langle {}^{a_0} q_0, {q_{n+1}}^{a_{n+1}} \rangle \longrightarrow a_1 a_2 \dots a_n \ ;$$

where every double sequence $a_{i1}...a_{ik} a_{i1}...a_{ik}$ is recursively replaced by $(a_{i1}...a_{ik})^+$ by proceeding from the innermost cycles to the outermost ones, is in P; furthermore, if $a_0 = a_{n+1} = \#$, q_0 is initial, and q_{n+1} is final, then $\langle {}^{\#} q_0, {q_{n+1}}^{\#} \rangle$ is in S;

– for every essential support of a composed chain, add the rule

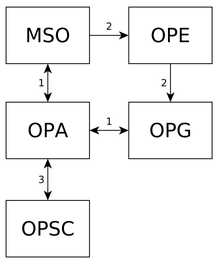
$$\langle {}^{a_0} q_0, {q_{n+1}}^{a_{n+1}} \rangle \longrightarrow \Lambda_0 a_1 \Lambda_1 a_2 \dots a_n \Lambda_n \ ;$$

where, for every $i = 0, 1, \dots, n$, $\Lambda_i = \langle {}^{a_i} q_i, {q'_i}^{a_{i+1}} \rangle$ if $x_i \neq \varepsilon$ and $\Lambda_i = \varepsilon$ otherwise, by replacing double cyclic sequences $\alpha_i \alpha_i$ with $(\alpha_i)^+$ in the same way as for simple chains; furthermore, if $a_0 = a_{n+1} = \#$, q_0 is initial, and q_{n+1} is final, then add $\langle {}^{\#} q_0, {q_{n+1}}^{\#} \rangle$ to S, and, if ε is accepted by $\mathcal{A}$, add $A \to \varepsilon$, A being a new axiom not otherwise occurring in any other rule.

Notice that the above construction is effective thanks to Lemma 18 and to the fact that subchains of composed chains are replaced by nonterminals Λ_i.

Remark. The definition of C-OPG and the constructions of Theorems 13 and 19 have been given with the same approach as used in [13], i.e., avoiding possible optimizations to keep technical details as simple and essential as possible. For instance, we allowed only the $^+$ operator in grammar's rhs as the minimum necessary extension to obtain the expressive power of OPAs: allowing, e.g., to use set union within the scope of a $^+$ operator would have allowed in some cases more compact grammars but such a choice would have also caused an explosion of the cardinality of the $\tilde{P}$ set. For the same reason we excluded a priori renaming and ε-rules in our grammars.

4 Other equivalences among OPL formalisms



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

- MSO denotes OPLs defined through MSO formulas
- OPA denotes OPLs defined through OP automata [13]
- OPSC denotes OPLs defined through a Syntactic Congruence with a finite number of equivalence classes [11]
- OPE denotes OPLs defined through OPEs [16]

Arrows between boxes denote language family inclusion; they are labeled by the reference pointing to paper where the property has been proved (1 is [13], 2 is [16], 3 is [11]).

Fig. 8. The relations among the various characterizations of OPLs.

Figure 8 displays the five equivalent formalisms introduced in the literature to define OPLs. The reciprocal inclusions between MSO and OPA, between OPA and the finite equivalence classes of OPL syntactic congruence, and the inclusion of MSO in OPE have been proved without the hypothesis of non-circularity of the $\doteq$ relation; the reciprocal inclusions between C-OPG and OPA have been restated in Section 3; it remains to consider the inclusion of OPE in OPG which was proved in [15] under the restrictive hypothesis. The proof used the claim that deriving an OPG from an OPE may exploit the closure properties of OPLs, in particular, w.r.t. the Kleene * operator; such a closure, however, was proved in [5] by using OPGs, again, under the restrictive hypothesis.

Although we will see, in the next section, that all closure properties of OPLs still hold even when there are circularities in the $\doteq$ relation, here it is convenient to consider the case of the Kleene * operator of OPEs separately. The reason is that in OPEs the Kleene * operator is now applied to subexpressions independently on the OP relations between their last and first terminal character.

Thus, it is first convenient to rewrite the OPE in a normal form using the $^+$ operator instead of the * one to avoid having to deal explicitly with the case of the ε string. Then, subexpressions of type $(\alpha)^+$ where the last terminal of α is not in relation $\doteq$ with the first one are replaced by the same procedure defined in [5] to prove effectively the closure w.r.t. the * operator. The new rules will produce a right or left-linear subtree of the occurrences of α depending on the OP relation between the two extreme terminals of α and will avoid the use of the * and $^+$ operators which are not permitted in the original OPGs.

The remaining substrings including the $^+$ operator are the new rhs of the C-OPG. The other technicalities of the construction of an OPG equivalent to an OPE are identical to those given in [15] and are not repeated here.

5 OPLs closure properties revisited

All major closure properties of OPLs have been originally proved by referring to their generating OPGs and some of them, in particular the closure w.r.t. the Kleene* operator, required the $\doteq$-acyclicity hypothesis. Thus, it is necessary to prove them again. However, since some of those proofs are technically rather involved, here we simply observe that it is easier to restate the same properties by exploiting OPAs which are now fully equivalent to C-OPGs.

Observe, in fact, that, thanks to the determinization of nondeterministic OPAs, closure w.r.t. boolean operations is "for free". Closure w.r.t. concatenation can be seen as a corollary of the closure proved in [13] of the concatenation between an OPL whose strings have finite length and an ω-OPL, i.e., a language of infinite strings. The construction is based on a nondeterministic guess of the position where a string of the first language could end —and a nontrivial technique to decide whether it could accepted even in the absence of the # delimiter—. Then, the closure w.r.t. Kleene * is obtained simply by allowing the OPA to repeat such a guess any number of times until the real $^\#$ is found.

6 Conclusion

We have filled up a longstanding "hole" in the theory of OPLs under the pressure of recent practical applications in the field of parallel compilation that showed how such a hole could hamper the benefits of parallelism [12]. The new grammatical formalism of C-OPGs, now fully equivalent to OPAs, MSO-logic, and OPEs in the definition of OPLs, can therefore be exploited to revisit the parallel compilation techniques of [12] (a rather unusual case where theory comes *after* its motivating application) or to improve the efficiency of techniques based on the less powerful OPGs [2].

Other algebraic and logic properties of OPLs, e.g., aperiodicity, star-freeness, first-order definability [17,15] can be re-investigated at the light of this generalization.

References

1. Autebert, J., Berstel, J., Boasson, L.: Context-free languages and pushdown automata. In: Handbook of Formal Languages (1), pp. 111–174 (1997). https://doi.org/10.1007/978-3-642-59136-5_3
2. Barenghi, A., Crespi Reghizzi, S., Mandrioli, D., Panella, F., Pradella, M.: Parallel parsing made practical. Sci. Comput. Program. **112**(3), 195–226 (2015), doi: 10.1016/j.scico.2015.09.002
3. Barenghi, A., Crespi Reghizzi, S., Mandrioli, D., Pradella, M.: Parallel parsing of operator precedence grammars. Inf. Process. Lett. **113**(7), 245–249 (2013). https://doi.org/10.1016/j.ipl.2013.01.008
4. Chiari, M., Mandrioli, D., Pradella, M.: Model-checking structured context-free languages. In: Silva, A., Leino, K.R.M. (eds.) Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12760, pp. 387–410. Springer (2021). https://doi.org/10.1007/978-3-030-81688-9_18, `https://doi.org/10.1007/978-3-030-81688-9_18`
5. Crespi Reghizzi, S., Mandrioli, D.: Operator Precedence and the Visibly Pushdown Property. J. Comput. Syst. Sci. **78**(6), 1837–1867 (2012)
6. Crespi Reghizzi, S., Mandrioli, D., Martin, D.F.: Algebraic Properties of Operator Precedence Languages. Information and Control **37**(2), 115–133 (May 1978)
7. Crespi Reghizzi, S., Pradella, M.: Beyond operator-precedence grammars and languages. Journal of Computer and System Sciences **113**, 18–41 (2020). https://doi.org/10.1016/j.jcss.2020.04.006
8. Floyd, R.W.: Syntactic Analysis and Operator Precedence. J. ACM **10**(3), 316–333 (1963)
9. Grune, D., Jacobs, C.J.: Parsing techniques: a practical guide. Springer, New York (2008)
10. Harrison, M.A.: Introduction to Formal Language Theory. Addison Wesley (1978)
11. Henzinger, T.A., Kebis, P., Mazzocchi, N., Saraç, N.E.: Regular methods for operator precedence languages. In: Etessami, K., Feige, U., Puppis, G. (eds.) 50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany. LIPIcs, vol. 261, pp. 129:1–129:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPIcs.ICALP.2023.129
12. Li, L., Taura, K.: Associative operator precedence parsing: A method to increase data parsing parallelism. In: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, HPC Asia 2023, Singapore, 27 February 2023 - 2 March 2023. pp. 75–87. ACM (2023). https://doi.org/10.1145/3578178.3578233, `https://doi.org/10.1145/3578178.3578233`
13. Lonati, V., Mandrioli, D., Panella, F., Pradella, M.: Operator precedence languages: Their automata-theoretic and logic characterization. SIAM J. Comput. **44**(4), 1026–1088 (2015)
14. Mandrioli, D., Pradella, M.: Generalizing input-driven languages: Theoretical and practical benefits. Computer Science Review **27**, 61–87 (2018). https://doi.org/10.1016/j.cosrev.2017.12.001
15. Mandrioli, D., Pradella, M., Crespi Reghizzi, S.: Star-freeness, first-order definability and aperiodicity of structured context-free languages. In: Proceedings of ICTAC 2020. Lecture Notes in Computer Science, Springer (2020)

16. Mandrioli, D., Pradella, M., Crespi Reghizzi, S.: Aperiodicity, star-freeness, and first-order definability of structured context-free languages. Logical Methods in Computer Science (2023), `https://arxiv.org/abs/2006.01236`, to appear
17. McNaughton, R., Papert, S.: Counter-free Automata. MIT Press, Cambridge, USA (1971)
18. Salomaa, A.K.: Formal Languages. Academic Press, New York, NY (1973)