OPERATOR PRECEDENCE LANGUAGES: THEIR AUTOMATA-THEORETIC AND LOGIC CHARACTERIZATION

VIOLETTA LONATI*, DINO MANDRIOLI†, FEDERICA PANELLA†, AND MATTEO PRADELLA†

Abstract. Operator precedence languages were introduced half a century ago by Robert Floyd to support deterministic and efficient parsing of context-free languages. Recently, we renewed our interest in this class of languages thanks to a few distinguishing properties that make them attractive for exploiting various modern technologies. Precisely, their local parsability enables parallel and incremental parsing, whereas their closure properties make them amenable for automatic verification techniques, including model checking.
In this paper we provide a fairly complete theory of this class of languages: we introduce a class of automata with the same recognizing power as the generative power of their grammars; we provide a characterization of their sentences in terms of monadic second order logic as it has been done in previous literature for more restricted language classes such as regular, parenthesis, and input-driven ones; we investigate preserved and lost properties when extending the language sentences from finite length to infinite length (ω-languages). As a result, we obtain a class of languages that enjoys many nice properties of regular languages (closure and decidability properties, logic characterization) but is considerably larger than other families –typically parenthesis and input-driven ones– with the same properties, covering "almost" all deterministic languages.[1]

Key words. Operator Precedence, Visibly Pushdown Languages, Monadic Second Order Logic, Omega-languages.

AMS subject classifications. 03D05, 68Q45.

Introduction. Operator precedence grammars and languages (OPGs and OPLs) certainly deserve an important place in the history of formal languages and compilers. They were invented by Robert Floyd [23] with the major motivation of enabling efficient, deterministic parsing of programming languages. In fact Floyd's intuition was inspired by arithmetic expressions whose structure is determined either by explicit parentheses or by the conventional, "hidden" *precedence* of multiplicative operators over additive ones. By generalizing this observation Floyd defined three basic relations between terminal symbols, namely *yields* and *takes precedence* and *equal in precedence* (respectively denoted by symbols $\lessdot$, $\gtrdot$, $\doteq$), in such a way that the right hand side (r.h.s.) of an operator precedence grammar rule is enclosed within a pair $\lessdot$, $\gtrdot$, and $\doteq$ holds between consecutive terminal symbols thereof (in OPGs nonterminal symbols are "transparent", i.e., irrelevant, w.r.t. the precedence relations [23]).

Subsequently, under the main motivation of grammar inference, it was shown that, once an operator precedence matrix (OPM) is given such that at most one relation holds between any two terminal characters, the family of OPLs sharing the given OPM is a Boolean algebra [19]. This result somewhat generalizes closure properties enjoyed by regular languages and by context-free languages whose structure, i.e., the syntax tree, is immediately visible in the terminal sentences, such as parenthesis languages [31] and tree-automata languages [11]. Such interesting algebraic properties enabled original inference algorithms, such as those proposed in [20]. After these initial results the theoretical investigation of OPLs was almost abandoned, most likely because of the advent of more general grammars, mainly the LR ones [26], which support parsing algorithms for the whole class of deterministic context-free languages. Nevertheless OPG-based parsing remains of some interest thanks to its simplicity and efficiency and is still used, at least partially, in many practical cases [24].

In the last decades, instead, an independent branch of research generated a flourishing of new results in terms of logic characterization of language families, ignited by the pioneering

*DI, Università degli Studi di Milano (lonati@di.unimi.it)
†DEIB, Politecnico di Milano ({dino.mandrioli, federica.panella, matteo.pradella}@polimi.it)
[1]Preliminary versions of some results presented in this paper appeared in [28, 29, 34].

results by Büchi and others [12, 32] on the monadic second order (MSO) logic characterization of regular languages over finite or infinite words (ω-languages) and motivated mainly by the breakthrough application of model-checking, which is rooted in closure properties and decidability of the emptiness problem, besides correspondence between automata-theoretic and logic language characterization. The present state of the art exhibits plenty of language families and related characterization in terms of various forms of logic formalisms (first-order, propositional, temporal logic and more specialized ones [17, 1]); most of them are motivated by the wish to extend model-checking techniques, i.e., decidability of system properties, beyond the natural scope of finite state machines.

Within such a rich literature, Visibly Pushdown Languages (VPLs) [4], previously known as Input-Driven Languages (IDLs) [5] certainly deserve a major role. In a nutshell IDLs alias VPLs are based on, and extend original parenthesis languages [31], e.g. by allowing for unmatched closed and open parentheses at the beginning and end of a sentence, respectively. Throughout the years this research field produced a fairly complete study of this family of languages whose main features can be summarized as follows:

- Being essentially a generalization of parenthesis languages their structure is immediately transparent at the "surface sentence", unlike more general context-free languages; arithmetic expressions, e.g., which are found in practically every programming language, do not reflect in the sequence of the leaves of the syntax tree the internal structure of the tree, which can be built only by knowing that multiplication operators take precedence over the additive ones.
- They have a complete characterization in terms of pushdown-automata and context-free grammar families recognizing and generating, them, respectively.
- They are closed w.r.t. to all fundamental language operations (Boolean, concatenation, Kleene *,…), like regular languages and unlike more general CF families.
- Within the landscape of algorithms that are necessary to develop model-checking techniques –whose complexities span from NP to PSPACE, EXPTIME,… completeness– they exhibit "comparable" complexities: for instance, the core algorithm for determinizing nondeterministic visibly pushdown automata (VPAs) has $2^{O(s^2)}$ complexity w.r.t. the cardinality s of the original state space and the inclusion problem for VPLs of both finite and infinite strings is EXPTIME-complete.
- They are characterized in terms of a MSO logic that applies both to finite and infinite length words.

Similar results have been obtained also for other classes of languages on the basis of the strong motivation provided by "model-checking like" applications [10].

Recently, our interest in OPLs has been renewed thanks to two, seemingly unrelated, properties thereof. The former one is their *local parsability*, i.e. the fact that the typical shift-reduce parsing algorithm associated with them, determines the replacing of a r.h.s. by the corresponding left hand side (l.h.s) exclusively on the basis of the embracing $\lessdot$ and $\gtrdot$ relations, i.e., independently on parts of the string that may be arbitrarily far from the considered r.h.s. This property is not enjoyed by more powerful grammars such as LR ones and nowadays it may far compensate the minor loss of generative power because it makes easier and more efficient exploiting parallelism and incrementality in the parsing of large strings formalizing complex systems and their behavior. The exploitation of this property, however, is the target of a different and –so far– independent research whose first results are documented in [7] and [6].

In this paper, instead, we focus on another, equally stimulating property of OPLs. We realized, in fact, that the OPL family strictly includes the independently studied family of VPLs and other related ones such as balanced languages [8]. On the basis of this somewhat surprising remark we further investigated other closure properties of OPLs besides the Boolean ones that were originally proved in [19]: the result is that OPLs are, to the best of our knowledge, the largest class of languages that enjoys all major closure properties that are typical of regular languages [18][2] Herewith the goal of this paper: to apply to OPLs the same successful verification techniques formerly developed for regular languages, VPLs, and other –input driven– language families, we develop a complete automata-theoretic and logic characterization of OPLs. In fact resuming the study of this old family of languages showed unexpected similarities with, and generalizations of, the peculiar properties of seemingly unrelated and differently motivated classes of languages.

In our opinion OPLs offer a surprising combination of the merits of IDLs and of those of more general deterministic context-free languages. On the one hand, in fact, they are input driven since their analysis can be based exclusively on the input characters and their pairwise relations; but, unlike more traditional IDLs, they are well suited to formalize general programming languages and other languages of practical interest; such a distinguishing feature allows us to extend to them closure and decidability properties not enjoyed by more general context-free languages. On the other hand, their minor lack of power w.r.t. deterministic languages does not prevent them from including most programming languages of practical interest: previous efforts in fact produced compilers based on OPGs for various programming languages such as ALGOL 68 and Prolog [23, 21]; more recently we exploited the mentioned property of local parsability to produce parallel parsers for JSON and Lua [6].

Given the fairly numerous collection of strongly connected properties we structure the present paper into two main parts. The first one completes the path begun with [19] and resumed with [18] by providing a fairly complete theory of traditional OPLs defined on strings of finite length; precisely, we present:

- A new family of pushdown automata fully equivalent to OPGs; rather surprisingly, in fact, a precise automata-theoretic characterization of OPLs was missing in the original literature[3].
- A complete characterization of OPLs in terms of monadic second-order (MSO) logic so as to align this family with a now classic approach of the literature –rooted in the work by Büchi. This allows, at least potentially, for the definition of model-checking algorithms to prove properties of languages defined either by means of generating grammars or by means of recognizing automata. Given the prohibitive complexity of decision algorithms based on MSO logic, however, it is common practice in the literature to resort to model-checking algorithms based on less powerful but simpler logics. We will provide a few hints on pursuing such an approach in the conclusion.

In the second part of this paper we define ωOPLs, i.e. the OPLs of infinite words. Infinite words languages are becoming more and more relevant in the literature due to the need of modeling systems whose behavior proceeds indefinitely, such as operating systems, control systems, etc. After introducing and comparing various forms of acceptance of infinite words by our OPAs by paralleling classical literature of ω-regular languages, we re-investigate their main properties by pointing out which of them are preserved from the finite length case and which ones are lost. This includes also a further characterization of ωOPLs in terms of MSO logic.

In the conclusions we briefly hint at further research directions, noticeably investigating

[2]Other language families falling in between input-driven and context-free languages, such as the height-deterministic family [33] or the synchronized pushdown languages [14], enjoy some but not all of the basic closure properties; furthermore such families are in general nondeterministic.

[3]The OP automata studied in this paper are significantly simplified w.r.t. their original formulation proposed in [28].

the relations of OPLs with less powerful but less complex logics than MSO ones, as it has been or is being done for other (input-driven) language families.

Part I: Finite Words Operator Precedence Languages

This part is devoted to finite length OPLs. After stating basic definitions and terminology (Section I.1) and resuming previous results already available in the open literature (Section I.1.1), we introduce the new class of pushdown automata explicitly tailored at OPLs: in Section I.2 we give the basic definitions and provide examples to show their usefulness in modeling various cases of practical interest; then we show the equivalence between deterministic and nondeterministic versions of these automata, at the price, however, of an increase in state space size given by an exponential function with quadratic exponent; we also study the complexity of decision problems for OPLs. Section I.3 shows, in a constructive way, the equivalence between OPGs and the new class of automata; finally, Section I.4 presents a monadic second order logic characterization of OPLs.

I.1. Preliminaries. A *context-free* (CF) grammar is a 4-tuple $G = (N, \Sigma, P, S)$, where N is the nonterminal alphabet, Σ is the terminal one, P the rule (or production) set, and $S \subseteq N$ the set of axioms[4]. The empty string is denoted ε. An *empty rule* has ε as the right hand side (r.h.s.). A *renaming rule* has one nonterminal as r.h.s. A grammar is *reduced* if every rule can be used to generate some string in Σ^*. It is *invertible* if no two rules have identical r.h.s.

The *direct derivation* relation is denoted by $\Rightarrow$ and its reflexive transitive closure, the *derivation* relation, is denoted by $\overset{*}{\Rightarrow}$.

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

In this initial part we will use arithmetic expressions, which are a small fraction of practically all programming languages, as a running example to introduce and explain the basic definitions, properties and constructions referring to OPLs.

Example 1. *Arithmetic expressions considered in this paper include two operators, an additive one and a multiplicative one that takes precedence over the other one, in the sense that, during the interpretation of the expression, multiplications must be executed before sums; as usual parentheses are used to specify a different precedence hierarchy between the two operations. They are denoted by the special symbols* $\llparenthesis$ *and* $\rrparenthesis$ *to avoid overloading with the use of the same symbol in all other formulae of this paper. Figure I.1.1 presents a grammar and the derivation tree of expression* $n + n \times \llparenthesis n + n \rrparenthesis$ *generated thereby; all nonterminals are also axioms.*

Notice that the structure of the syntax tree (uniquely) corresponding to the input expression reflects the precedence order which drives computing the value attributed to the expression. This structure, however, is not immediately visible in the expression: in fact Figure I.1.2 proposes a different grammar which generates the same expressions as the grammar of Figure I.1.1 but would associate to the same sentence the syntax tree displayed in the right part of the figure. Yet another (ambiguous) grammar could generate both. If instead we used a parenthesis grammar to generate arithmetic expressions, it would produce the string $\llparenthesis n + \llparenthesis n \times \llparenthesis n + n \rrparenthesis \rrparenthesis \rrparenthesis$ *instead of the previous one and the structure of the corresponding tree would*

[4]This less usual but equivalent definition of axioms as a set has been adopted for parenthesis languages [31] and other input-driven languages; we chose it for this paper to simplify some notations and constructions.

$$\begin{aligned} E &\to E + T \mid T \\ T &\to T \times F \mid F \\ F &\to n \mid \llparenthesis E \rrparenthesis \end{aligned}$$

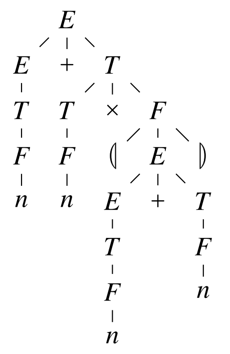


Figure I.1.1: A grammar generating arithmetic expressions with parentheses.

$$\begin{aligned} A &\to B \times A \mid B \\ B &\to B + C \mid C \\ C &\to n \mid \llparenthesis A \rrparenthesis \end{aligned}$$

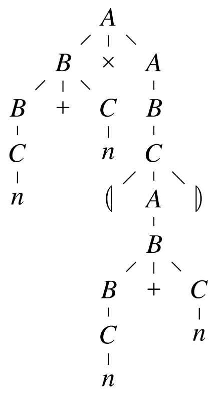


Figure I.1.2: A grammar generating the same arithmetic expression as that of Figure I.1.1 and the corresponding tree where, instead, + takes precedence over ×.

be immediately visible in the expression. For this reason we say that such general grammars "hide" the structure associated with a sentence –even when they are unambiguous– whereas parenthesis grammars and other input-driven ones make the structure explicit in the sentences they generate.

A rule is in *operator form* if its r.h.s. has no adjacent nonterminals; an *operator grammar* (OG) contains just such rules. Notice that both grammars of Figure I.1.1 and of Figure I.1.2 are OGs. Any CF grammar $G = (N, \Sigma, P, S)$ admits an equivalent OG $G' = (N', \Sigma, P', S)$, where the size of N' is $O(|\Sigma| \cdot (|\Sigma| + k \cdot |P|))$ and that of P' is $O(|\Sigma| \cdot (|N| + k \cdot |\Sigma| \cdot |P|))$, k being the maximum length of P's r.h.s.s [25, 38].

The coming definitions for operator precedence grammars (OPGs) [23] are from [19] and [18], where they are also called *Floyd Grammars* or FGs.

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

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

where $B \in N \cup \{\varepsilon\}$. The grammar name G will be omitted unless necessary to prevent

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

Figure I.1.3: The OPM of the grammar in Figure I.1.1.

confusion. For the grammar of Figure I.1.1 the left and right terminal sets of nonterminals E, T and F are, respectively:

$$\begin{aligned} \mathcal{L}(E) &= \{+, \times, n, \llparenthesis\} & \mathcal{R}(E) &= \{+, \times, n, \rrparenthesis\} \\ \mathcal{L}(T) &= \{\times, n, \llparenthesis\} & \mathcal{R}(T) &= \{\times, n, \rrparenthesis\} \\ \mathcal{L}(F) &= \{n, \llparenthesis\} & \mathcal{R}(F) &= \{n, \rrparenthesis\} \end{aligned}$$

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

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

Notice that, unlike the usual arithmetic relations denoted by similar symbols, the above precedence relations do not enjoy anyone of transitive, symmetric, reflexive properties. For an OG G, the *operator precedence matrix* (OPM) $M = OPM(G)$ is a $|\Sigma| \times |\Sigma|$ array that, for each ordered pair (a, b), stores the set M_{ab} of OP relations holding between a and b.

Figure I.1.3 displays the OPM associated with the grammar of Figure I.1.1 where, for an ordered pair (a, b), a is one of the symbols shown in the first column of the matrix and b one of those occurring in its first line.

Given two OPMs M_1 and M_2, we define set inclusion and union:

$$M_1 \subseteq M_2 \text{ if } \forall a, b : (M_1)_{ab} \subseteq (M_2)_{ab}, \qquad M = M_1 \cup M_2 \text{ if } \forall a, b : M_{ab} = (M_1)_{ab} \cup (M_2)_{ab}$$

Definition I.1.1 (Operator precedence grammar and language). *An OG G is an* operator precedence *or* Floyd grammar *(*OPG*) if, and only if, $M = OPM(G)$ is a conflict-free matrix, i.e., $\forall a, b$, $|M_{ab}| \leq 1$. An* operator precedence language (OPL) *is a language generated by an OPG.*

From the above definition it is immediate to verify that both grammars of Figure I.1.1 and of Figure I.1.2 are OPGs (with different OPMs).

Two matrices are *compatible* if their union is conflict-free. A matrix is *total* (or *complete*) if it contains no empty case. The following definition of Fischer Normal Form is adapted from the original one [22] to take into account that in our basic definition of CF grammar S is a set rather than a singleton.

Definition I.1.2 (Fischer Normal Form). *An OPG is in* Fischer normal form *(FNF) iff it is invertible, has no empty rule except possibly $A \to \varepsilon$, where A is an axiom not used elsewhere, and no renaming rules.*

Let $G = (N, \Sigma, P, S)$ be an OPG; then an equivalent OPG $\tilde{G} = (\tilde{N}, \Sigma, \tilde{P}, \tilde{S})$ in FNF, can be built such that $\tilde{N}$ is $\wp(N)$ and $|\tilde{P}|$ is $O(|P| \cdot 2^{|N| \cdot \lceil \frac{k}{2} \rceil})$, where k is the maximum length of P's r.h.s.s [25].

A FNF (manually) derived from the grammar of Figure I.1.1 is given below. Notice that in this case the size of the nonterminal alphabet and of the productions is much smaller than the worst case upper bound provided by the general construction.

$$\begin{aligned} E &\to E+T \mid E+F \mid T+T \mid F+F \mid F+T \mid T+F \\ T &\to T\times F \mid F\times F \\ F &\to n \mid (\!|E|\!) \mid (\!|T|\!) \mid (\!|F|\!) \end{aligned}$$

It is well-known that OPLs are a proper subfamily of deterministic context-free languages: for instance, it is impossible to generate the language $\{a^n b a^n\}$, without producing a precedence conflict $a \lessdot a$ and $a \gtrdot a$. Despite this theoretical limitation OPLs have been successfully used to formalize many programming languages and to support their compilers; in this paper we will also provide several other examples of potential application of this model in different fields.

OPMs play a fundamental role in deterministic parsing of OPLs. Thus in the view of defining automata to parse OPLs (Operator Precedence Automata or OPAs) we pair them with the alphabet. To this goal, we use a special symbol # not in Σ to mark the beginning and the end of any string. This is consistent with the typical operator parsing technique which requires the lookback and lookahead of one character to determine the precedence relation [24]. The precedence relations in the OPM are implicitly extended to include #: the initial # can only yield precedence, and other symbols can only take precedence over the ending #.

Definition I.1.3 (Operator precedence alphabet). *An* operator precedence (OP) alphabet *is a pair (Σ, M) where Σ is an alphabet and M is a conflict-free operator precedence matrix, i.e. a $|\Sigma \cup \{\#\}|^2$ array that associates at most one of the operator precedence relations: $\doteq$, $\lessdot$ or $\gtrdot$ with each ordered pair (a,b).*

If $M_{ab} = \{\circ\}$, with $\circ \in \{\lessdot, \doteq, \gtrdot\}$,we write $a \circ b$. For $u, v \in \Sigma^*$ we write $u \circ v$ if $u = xa$ and $v = by$ with $a \circ b$. The relations involving the # delimiter are constrained as stated above.

The notion of chain introduced by the following definitions provides a formal description of the intuitive concept of "invisible or hidden structure" discussed in Example 1.

Definition I.1.4 (Chains). *Let (Σ, M) be a precedence 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 $i : 1 \le i \le n$, $M_{a_0 a_{n+1}} \neq \emptyset$, 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 $i : 0 \le i \le n$. 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.*

Example 2. *The "hidden" structure induced by the operator precedence alphabet of Example 1 for the expression $\#n + n \times (\!|n+n|\!)\#$ is represented in Figure I.1.4, where ${}^{\#}[x_0 + x_1]^{\#}$, ${}^{+}[y_0 \times y_1]^{\#}$, ${}^{\times}[(\!|w_0|\!)]^{\#}$, ${}^{(\!|}[z_0 + z_1]^{|\!)}$ are composed chains and ${}^{\#}[n]^{+}$, ${}^{+}[n]^{\times}$, ${}^{(\!|}[n]^{+}$, ${}^{+}[n]^{|\!)}$ are simple chains.*

Definition I.1.5 (Depth of a chain). *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 composed chain ${}^{\#}[x_0 + x_1]^{\#}$ in Example 2 has depth 5. Thus, if for an OPG G it is $OPM(G) = M$, the depth of a chain body x is the height of the syntax tree, if any, whose frontier is x.

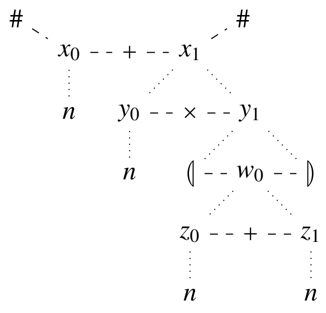


Figure I.1.4: Structure of the chains in the expression $\#n + n \times \llparenthesis n + n \rrparenthesis\#$ of Example 2 .

Definition I.1.6 (Compatible word). *A word w over (Σ, M) is* compatible *with M iff the two following conditions hold:*

- *for each pair of letters c, d, consecutive in w, $M_{cd} \neq \emptyset$*
- *for each factor (substring) x of $\#w\#$ such that $x = a_0 x_0 a_1 x_1 a_2 \ldots\ a_n x_n a_{n+1}$, if $a_0 \lessdot a_1 \doteq a_2 \ldots a_{n-1} \doteq a_n \gtrdot a_{n+1}$ and, for every $0 \leq i \leq n$, either $x_i = \varepsilon$ or ${}^{a_i}[x_i]^{a_{i+1}}$ is a chain (simple or composed), then $M_{a_0 a_{n+1}} \neq \emptyset$.*

For instance, the word $n + n \times \llparenthesis n + n \rrparenthesis$ is compatible with the operator precedence alphabet of Example 1, whereas $n + n \times \llparenthesis n + n \rrparenthesis \llparenthesis n + n \rrparenthesis$ is not.

The chains fully determine the structure of the words; in particular, given an OP alphabet, each word in Σ^* compatible with M is assigned a tree-structure by the OPM M. If M is complete, then each word is compatible with M and the OPM M assigns a structure to any word in Σ^*. For this reason we say that OPLs somewhat generalize the notion of IDL, since their parsing is driven by the OPM which is defined on the terminal alphabet, but they also allow to generate sentences whose structure is “invisible” before parsing.

The equal in precedence relations of an OP alphabet are connected with an important parameter of the grammar, namely the length of the right hand sides of the rules. Clearly, a rule $A \to A_1 a_1 \ldots A_t a_t A_{t+1}$, where each A_i is a possibly missing nonterminal, is associated with relations $a_1 \doteq a_2 \doteq \ldots \doteq a_t$. If the $\doteq$ relation is *cyclic*, i.e., there exist $a_1, a_2, \ldots, a_n \in \Sigma$ $(n \geq 1)$ such that $a_1 \doteq a_2 \doteq \ldots \doteq a_n \doteq a_1$, there is *a priori* no finite bound on the length of the r.h.s. of a production. Otherwise the length is bounded by $2 \cdot c + 1$, where $c \geq 1$ is the length of the longest $\doteq$-chain.

Previous literature [18, 28] assumed that all precedence matrices of OPLs are $\doteq$-cycle free. In the case of OPGs this prevents the risk of r.h.s. of unbounded length [19], but could be replaced by the weaker restriction of production's r.h.s. of bounded length, or could be removed at all by allowing such unbounded forms of grammars –e.g. with regular expressions as r.h.s. In our experience, such assumption helps to simplify notations and some technicalities of proofs; moreover we found that its impact in practical examples is minimal. In this paper we accept a minimal loss of generation[5] power and assume the simplifying assumption of $\doteq$-acyclicity. We will see, however, that this hypothesis has an impact only on constructions involving grammars but is irrelevant for the OP automata defined in this paper.

I.1.1. Previous results. Herein we present some basic properties of OPLs that have already been stated in previous literature. Preliminarily, notice that, since the union of two

[5]An example language that cannot be generated with an $\doteq$-acyclic OPM is the following: $\mathcal{L} = \{a^n (bc)^n \mid n \geq 0\} \cup \{b^n (ca)^n \mid n \geq 0\} \cup \{c^n (ab)^n \mid n \geq 0\}$

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

Legend
Σ_c denotes "calls"
i.e. a generalized version of open parentheses;
Σ_r denotes "returns"
i.e. a generalized version of closed parentheses;
Σ_i denotes internal characters
i.e., characters that are not pushed onto the stack and are managed exclusively by finite state control.

Figure I.1.5: A partitioned matrix, where Σ_c, Σ_r,Σ_i are set of terminal characters. A precedence relation in position Σ_α, Σ_β means that relation holds between all symbols of Σ_α and all those of Σ_β.

acyclic OPMs might be cyclic, when we consider, in the sequel, the union $M = M_1 \cup M_2$ of two OPMs M_1 and M_2 we always assume that M too is acyclic.

Statement I.1.1. *[19] OPLs are closed with respect to Boolean operations. Precisely, given two OPLs L_1, L_2 with compatible OPMs M_1 and M_2, $L_1 \cap L_2$ and $L_1 \cup L_2$ are OPLs whose OPM is contained in $M_1 \cup M_2$; furthermore, let L_1^{max} be the OPL of all strings compatible with M_1, then $L_1^{max} \setminus L_1$ is an OPL whose OPM is contained in M_1. In particular, if M_1 is a complete OPM, L_1^{max} is Σ^* (where each sentence has a structure determined by M_1); then $\Sigma^* \setminus L_1$ is an OPL whose OPM is contained in M_1.*

Statement I.1.2. *[18] OPLs are closed with respect to concatenation and Kleene $*$ operation. Precisely, given two OPLs L_1, L_2 with compatible OPMs M_1 and M_2, $L_1.L_2$ and L_1^* are OPLs whose OPM is compatible with $M_1 \cup M_2$ (resp. M_1). Notice that in this case the construction of the new grammars may introduce new precedence relations not existing in the original matrices. Furthermore, OPLs are closed under alphabetical homomorphisms that preserve conflict-freedom.*

Statement I.1.3. *[18] OPLs strictly include the family of VPLs. Precisely, VPLs are the subfamily of OPLs whose OPM is a* partitioned matrix*, i.e. a matrix whose structure is depicted in Figure I.1.5.*

I.2. Operator Precedence Automata. Next, we introduce a family of pushdown automata that recognize exactly OPLs. OPLs being naturally oriented towards bottom-up parsing, their accepting automata exhibit a typical shift-reduce attitude; they are considerably simpler, however, than other classical automata of this type such as LR ones.

Definition I.2.1 (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,*
- *$\delta : Q \times (\Sigma \cup Q) \to \wp(Q)$ is the transition function, which is the union of three 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 $\mathit{symbol}([a,\ q]) = a$, $\mathit{symbol}(\bot) = \#$, and $\mathit{state}([a,\ q]) = q$. Given a string $\Pi = \bot\pi_1\pi_2 \dots \pi_n$, with $\pi_i \in \Gamma$, $n \geq 0$, we set $\mathit{symbol}(\Pi) = \mathit{symbol}(\pi_n)$, including the particular case $\mathit{symbol}(\bot) = \#$.

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 $\mathit{symbol}(\Pi) \lessdot a$ 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.

We say that a configuration $\langle \bot,\ q_I,\ x\# \rangle$ is *initial* if $q_I \in I$ and a configuration $\langle \bot,\ q_F,\ \# \rangle$ is *accepting* if $q_F \in F$. The language accepted by the automaton is defined as:

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

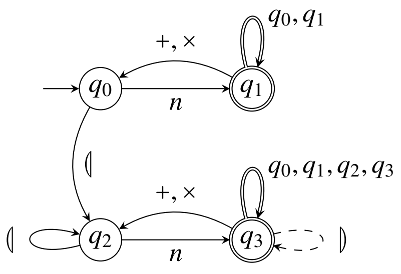
Example 3. *The OPA depicted in Figure I.2.1 accepts the language of arithmetic expressions generated by the OPG of Example 1. The same figure also shows an accepting computation on input* $n + n \times (\!|n + n|\!)$.

Therefore, an OPA selects an appropriate subset within the "universe" of strings in Σ^* compatible with M. This property somewhat resembles the fundamental Chomsky-Shützenberger Theorem, in that a universe of nested structures –a Dyck language– is restricted by means of an "intersection" with a finite state mechanism. For instance, the automaton of Figure I.2.1 recognizes well-nested parenthesized arithmetic expressions and could be modified in such a way that parentheses are used only when needed to give the expression the desired meaning, i.e., a pair of parentheses containing a + is necessary only if it is adjacent to a $\times$; parentheses enclosing only $\times$ should be avoided.

The following definitions will be used throughout the paper to characterize OPA behavior: we use arrows $\longrightarrow$, $\dashrightarrow$ and $\Longrightarrow$ to denote push, shift and pop transitions, respectively.

Definition I.2.2. *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*

$$q_0 \xrightarrow{a_1} q_1 \dashrightarrow \dots \dashrightarrow q_{n-1} \overset{a_n}{\dashrightarrow} q_n \overset{q_0}{\Longrightarrow} q_{n+1} \tag{I.2.1}$$



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

Figure I.2.1: Automaton and example of computation for the language of Example 3. Recall that shift, push and pop transitions are denoted by dashed, normal and double arrows, respectively.

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*

$$q_0 \overset{x_0}{\rightsquigarrow} q_0' \xrightarrow{a_1} q_1 \overset{x_1}{\rightsquigarrow} q_1' \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n \overset{x_n}{\rightsquigarrow} q_n' \overset{q_0'}{\Longrightarrow} q_{n+1} \qquad \text{(I.2.2)}$$

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

- *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 $q_0 \overset{x}{\rightsquigarrow} q_{n+1}$.

EXAMPLE 4. *Figure I.2.2 illustrates the supports of the chains that, for the OPA described*

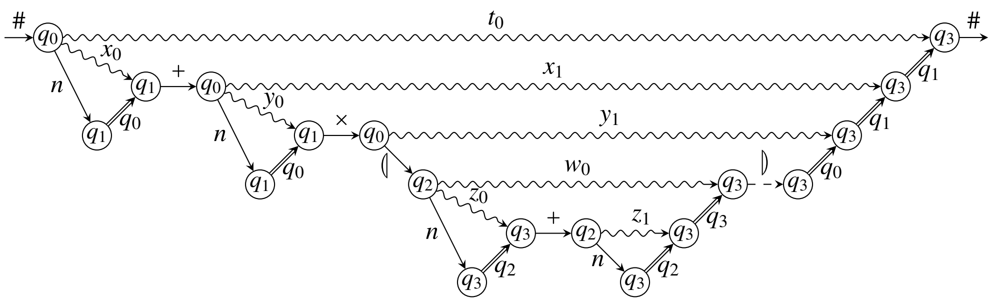


Figure I.2.2: Structure of chains and supports for the expression of Example 4.

in Example 3, compose the structure of the expression $n + n \times (\!|n + n|\!)$.

The chains fully determine the structure of the computation of any automaton on a word compatible with M. Indeed, let $\Pi \in \bot\Gamma^*$ with $symbol(\Pi) = a \lessdot x \gtrdot b$: an OPA $\mathcal{A}$ performs the computation $\langle \Pi,\ q,\ xb\rangle \vdash \langle \Pi,\ p,\ b\rangle$ without changing the portion Π of the stack, if and only if ${}^{a}[x]^{b}$ is a chain over (Σ, M) with a support $q \overset{x}{\rightsquigarrow} p$ in $\mathcal{A}$. The depth of x corresponds to the maximum number of push/pop pairs nested in the computation, i.e. the maximum height reached by the stack in one of the traversed configurations, minus the height of the stack in the starting configuration.

Notice that the context a, b of a chain 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 is exploited elsewhere e.g. to build parallel and/or incremental OP parsers [7].

With reference to Example 3 and Figure I.2.2, the parsing of substring $n + n$ within the context $(\!|, |\!)$ is given by the computation

$$\langle \Pi,\ q_2,\ n + n\ |\!)\#\rangle \overset{*}{\vdash} \langle \Pi,\ q_3,\ |\!)\#\rangle \quad \text{with} \quad \Pi = \bot[+,\ q_1][\times,\ q_1][(\!|,\ q_0]$$

which corresponds to support $q_2 \overset{n}{\rightsquigarrow} q_3 \xrightarrow{+} q_2 \overset{n}{\rightsquigarrow} q_3 \overset{q_3}{\Longrightarrow} q_3$ of the composed chain ${}^{(\!|}[n + n]^{|\!)}$, where $q_2 \overset{n}{\rightsquigarrow} q_3$ is the support $q_2 \xrightarrow{n} q_3 \overset{q_2}{\Longrightarrow} q_3$ of the simple chains ${}^{(\!|}[n]^{+}$ and ${}^{+}[n]^{|\!)}$.

I.2.1. Examples. In this section we illustrate an example of application of OPLs, which cannot be modeled by traditional classes of languages with an "explicit" structure such as parenthesis languages and VPLs. We shall present in Part 2 examples in other interesting contexts (such as operating systems) which can be naturally modeled by OPAs recognizing ω-languages, and are not recognizable by VPAs as well. Other examples of application of OPLs to model systems in various application fields outside the traditional one of programming languages are given in [35].

Indeed, the most distinguishable feature of the structure of VPLs is that in their OPMs the $\doteq$ relation occurs always and only between open and closed parentheses (Σ_c and Σ_r elements in [3] notation, respectively). Unlike traditional parenthesis languages, however, in VPLs parentheses can remain unmatched, but only at the beginning (Σ_r elements) and end (Σ_c elements) of the input string, respectively. This initial extension, however, is not sufficiently general to cover several interesting cases where an "event" of special type, e.g. a rollback or an exception, should force flushing the stack of many pending elements, say write operations or procedure calls.

Example 5. *OPAs can be used to model the run-time behavior of database systems, e.g., for modeling sequences of users' transactions with possible rollbacks. Other systems that exhibit an analogous behavior are revision control (or* versioning*) systems (such as subversion or git). As an example, consider a system for version management of files where a user can perform the following operations on documents: save them, access and modify them, undo one (or more) previous changes, restoring the previously saved version.*

The following alphabet represents the user's actions: sv (for save*), wr (for* write*, i.e. the document is opened and modified), ud (for a single* undo *operation), rb (for a* rollback *operation, where all the changes occurred since the previously saved version are discarded).*

An OPA that models the traces of possible actions of the user on a given document is a single-state automaton $\langle \Sigma, M, \{q\}, \{q\}, \{q\}, \delta \rangle$, where $\Sigma = \{sv, rb, wr, ud\}$, M is:

$$M = \begin{array}{c|ccccc} & sv & rb & wr & ud & \# \\ \hline sv & \lessdot & \doteq & \lessdot & & \gtrdot \\ rb & \gtrdot & \gtrdot & \gtrdot & \gtrdot & \gtrdot \\ wr & \lessdot & \gtrdot & \lessdot & \doteq & \gtrdot \\ ud & \gtrdot & \gtrdot & \gtrdot & \gtrdot & \gtrdot \\ \# & \lessdot & & \lessdot & & \doteq \end{array}$$

and $\delta_{push}(q, a) = q, \forall a \in \{sv, wr\}$, $\delta_{shift}(q, a) = q, \forall a \in \{rb, ud\}$ and $\delta_{pop}(q, q) = q$.

A more specialized model of this system might impose that the user regularly backs her work up, so that no more than N changes that are not undone (denoted wr as before) can occur between any two consecutive checkpoints sv (without any rollback rb between them). Figure I.2.3 shows the corresponding OPA with $N = 2$, with the same OPM M.

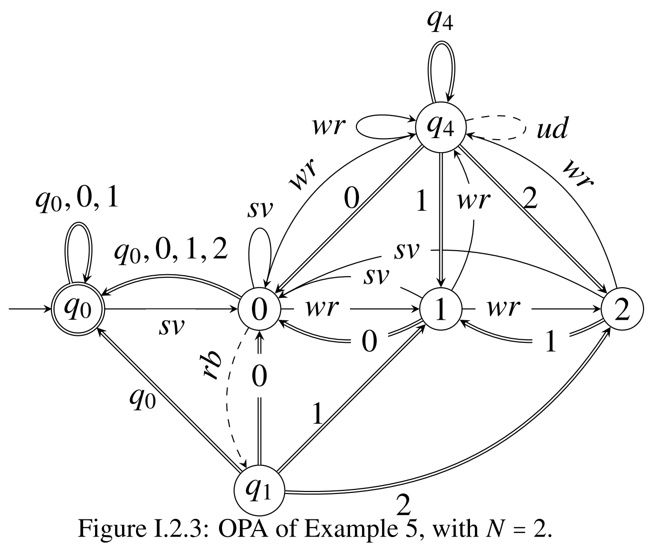


Figure I.2.3: OPA of Example 5, with $N = 2$.

States 0, 1 *and* 2 *denote respectively the presence of zero, one and two unmatched changes between two symbols sv.*

An example of computation on the string sv wr ud rb sv wr wr ud sv wr rb wr sv is shown in Figure I.2.4.

stack	state	current input
$\bot$	q_0	$sv\ wr\ ud\ rb\ sv\ wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \#$
$\bot[sv,\ q_0]$	0	$wr\ ud\ rb\ sv\ wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \#$
$\bot[sv,\ q_0][wr,\ 0]$	q_4	$ud\ rb\ sv\ wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \#$
$\bot[sv,\ q_0][ud,\ 0]$	q_4	$rb\ sv\ wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \#$
$\bot[sv,\ q_0]$	0	$rb\ sv\ wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \#$
$\bot[rb,\ q_0]$	q_1	$sv\ wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \#$
$\bot$	q_0	$sv\ wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \#$
$\bot[sv,\ q_0]$	0	$wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \#$
$\bot[sv,\ q_0][wr,\ 0]$	1	$wr\ ud\ sv\ wr\ rb\ wr\ sv\ \#$
$\bot[sv,\ q_0][wr,\ 0][wr,\ 1]$	q_4	$ud\ sv\ wr\ rb\ wr\ sv\ \#$
$\bot[sv,\ q_0][wr,\ 0][ud,\ 1]$	q_4	$sv\ wr\ rb\ wr\ sv\ \#$
$\bot[sv,\ q_0][wr,\ 0]$	1	$sv\ wr\ rb\ wr\ sv\ \#$
$\bot[sv,\ q_0][wr,\ 0][sv,\ 1]$	0	$wr\ rb\ wr\ sv\ \#$
$\bot[sv,\ q_0][wr,\ 0][sv,\ 1][wr,\ 0]$	1	$rb\ wr\ sv\ \#$
$\bot[sv,\ q_0][wr,\ 0][sv,\ 1]$	0	$rb\ wr\ sv\ \#$
$\bot[sv,\ q_0][wr,\ 0][rb,\ 1]$	q_1	$wr\ sv\ \#$
$\bot[sv,\ q_0][wr,\ 0]$	1	$wr\ sv\ \#$
$\bot[sv,\ q_0][wr,\ 0][wr,\ 1]$	2	$sv\ \#$
$\bot[sv,\ q_0][wr,\ 0][wr,\ 1][sv,\ 2]$	0	$\#$
$\bot[sv,\ q_0][wr,\ 0][wr,\ 1]$	q_0	$\#$
$\bot[sv,\ q_0][wr,\ 0]$	q_0	$\#$
$\bot[sv,\ q_0]$	q_0	$\#$
$\bot$	q_0	$\#$

Figure I.2.4: Example of computation for the specialized system of Example 5

I.2.2. Determinism vs Nondeterminism. An important property of OPAs is the equivalence between the deterministic and the nondeterministic version thereof. This result also implies the closure of OPLs under complementation, yielding an alternative proof to the traditional one presented in [19].

The deterministic version of OPAs is defined along the usual lines:

Definition I.2.3 (Deterministic OPA). *An OPA is deterministic if I is a singleton, and the ranges of δ_{push}, δ_{shift} and δ_{pop} are Q rather than $\wp(Q)$.*

It is well-known that the equivalence between nondeterministic and deterministic machines usually does not extend from finite state to pushdown ones. VPAs are however a noticeable exception. The construction described in [4] is extended here to cover OPAs too. Our construction ensures that two different pop moves of two different runs of the nondeterministic automaton never "mix up" their initial and final states in the deterministic one by keeping track of the path of the automaton since the push move that marks the origin of the chain to be reduced by the next pop move. Precisely, the states of the deterministic automaton $\tilde{\mathcal{A}}$ are sets of pairs of states, instead of sets of single states, of the nondeterministic automaton $\mathcal{A}$: $\tilde{\mathcal{A}}$ simulates $\mathcal{A}$ along the first component of the pair, whereas the second component stores the state that gave origin to a push transition and it is propagated through shift moves. The deterministic pop operations will simulate only the nondeterministic ones defined on the states corresponding to the first component of the current state and the state reached before the last push move, which corresponds to the state on the top of the stack in an actual run of the nondeterministic automaton.

The following theorem formalizes the above informal reasoning.

Theorem I.2.4. *Given a nondeterministic OPA $\mathcal{A}$ with s states, an equivalent deterministic OPA $\tilde{\mathcal{A}}$ can effectively be built with $2^{O(s^2)}$ states.*

Proof. Let $\mathcal{A}$ be $\langle \Sigma, M, Q, I, F, \delta \rangle$; $\tilde{\mathcal{A}} = \langle \Sigma, M, \tilde{Q}, \tilde{I}, \tilde{F}, \tilde{\delta} \rangle$ is defined as follows:

- $\tilde{Q} = \wp(Q \times (Q \cup \{\top\}))$, where $Q \cap \{\top\} = \varnothing$ and $\top$ is a symbol that stands for the baseline of the computations; we will use $K, K_i, \bar{K}, K', \ldots$ to denote states in $\tilde{Q}$,
- $\tilde{I} = I \times \{\top\}$ is the initial state of $\tilde{\mathcal{A}}$,
- $\tilde{F} = \{K \mid K \cap (F \times \{\top\}) \neq \varnothing\}$,
- $\tilde{\delta} : \tilde{Q} \times (\Sigma \cup \tilde{Q}) \to \tilde{Q}$ is the transition function defined as follows.
  The push transition $\tilde{\delta}_{\text{push}} : \tilde{Q} \times \Sigma \to \tilde{Q}$ is defined by

$$\tilde{\delta}_{\text{push}}(K, a) = \bigcup_{(q,p) \in K} \{(h, q) \mid h \in \delta_{\text{push}}(q, a)\}$$

  The shift transition $\tilde{\delta}_{\text{shift}} : \tilde{Q} \times \Sigma \to \tilde{Q}$ is defined by

$$\tilde{\delta}_{\text{shift}}(K, a) = \bigcup_{(q,p) \in K} \{(h, p) \mid h \in \delta_{\text{shift}}(q, a)\}$$

  The pop transition $\tilde{\delta}_{\text{pop}} : \tilde{Q} \times \tilde{Q} \to \tilde{Q}$ is defined as follows:

$$\tilde{\delta}_{\text{pop}}(K_1, K_2) = \bigcup_{(r,q) \in K_1, (q,p) \in K_2} \{(h, p) \mid h \in \delta_{\text{pop}}(r, q)\}.$$

Notice that, if $|Q| = s$ is the number of states of the nondeterministic OPA $\mathcal{A}$, the deterministic OPA $\tilde{\mathcal{A}}$ that is obtained in this way has a set of states whose size is exponential in s^2, i.e. $|\tilde{Q}| = 2^{|Q| \cdot |Q \cup \{\perp\}|}$ which is $2^{O(s^2)}$.

The proof of equivalence between the two automata is by induction and is based on lemmata I.2.5 and I.2.6.

Lemma I.2.5. *Let y be the body of a chain with support $q \overset{y}{\leadsto} q'$ in $\mathcal{A}$. Then, for every $p \in Q$ and $K \in \tilde{Q}$, if $K \ni (q, p)$, there exists a support $K \overset{y}{\leadsto} K'$ in $\tilde{\mathcal{A}}$ with $K' \ni (q', p)$.*

Proof. We argue by induction on the depth h of y. If $h = 1$ then $y = a_1 a_2 \ldots a_n$ and the support can be rewritten as in (I.2.1) with $q_0 = q$ and $q_{n+1} = q'$. Set $K_0 = K$ and

$$\begin{aligned} K_1 &= \tilde{\delta}_{\text{push}}(K_0, a_1) \\ K_i &= \tilde{\delta}_{\text{shift}}(K_{i-1}, a_i), \text{ for every } i = 2, \ldots, n \\ K' &= \tilde{\delta}_{\text{pop}}(K_n, K) \end{aligned}$$

Then

$$K \xrightarrow{a_1} K_1 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} K_{n-1} \xrightarrow{a_n} K_n \xRightarrow{K} K' \tag{I.2.3}$$

is a support for $\mathcal{C}$ in $\tilde{\mathcal{A}}$. Moreover, since $K \ni (q, p)$, by the definition of $\tilde{\delta}$ we have:

$$\begin{array}{ll} K_1 \ni (q_1, q) & \text{since } \delta_{\text{push}}(q, a_1) \ni q_1, \\ K_i \ni (q_i, q) & \text{since } \delta_{\text{shift}}(q_{i-1}, a_i) \ni q_i, \\ K' \ni (q', p) & \text{since } \delta_{\text{pop}}(q_n, q) \ni q'. \end{array}$$

Now assume that the statement holds for supports with depth lower than h and let $y = x_0 a_1 x_1 a_2 \ldots a_n x_n$ have depth h. The support can be rewritten as in (I.2.2) with $q_0 = q$ and

$q_{n+1} = q'$, where $q'_i = q_i$ whenever x_i is the empty word, and every non-empty x_i has depth lower than h.

Then, by the inductive hypothesis and the definition of $\tilde{\delta}$, we can build a support

$$K \overset{x_0}{\leadsto} K'_0 \xrightarrow{a_1} K_1 \overset{x_1}{\leadsto} K'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} K_n \overset{x_n}{\leadsto} K'_n \overset{K'_0}{\Longrightarrow} K' \qquad \text{(I.2.4)}$$

where, (q,p) being in K, we have:

$K'_0 \ni (q'_0, p)$ by inductive hypothesis on the support $q = q_0 \overset{x_0}{\leadsto} q'_0$,
$K_1 \ni (q_1, q'_0)$ since $\delta_{\text{push}}(q'_0, a_1) \ni q_1$,
$K'_1 \ni (q'_1, q'_0)$ by inductive hypothesis on the support $q_1 \overset{x_1}{\leadsto} q'_1$,
$K_i \ni (q_i, q'_0)$ since $\delta_{\text{shift}}(q'_{i-1}, a_i) \ni q_i$, for every $i = 2, \dots, n$,
$K'_i \ni (q'_i, q'_0)$ by inductive hypothesis on the support $q_i \overset{x_i}{\leadsto} q'_i$,
$K' \ni (q', p)$ since $\delta_{\text{pop}}(q_n, q'_0) \ni q'$,

and this concludes the proof. □

Lemma I.2.6. *Let y be the body of a chain with support $K \overset{y}{\leadsto} K'$ in $\tilde{\mathcal{A}}$. Then, for every $p, q' \in Q$, if $K' \ni (q', p)$ there exists a support $q \overset{y}{\leadsto} q'$ in $\mathcal{A}$ with $(q,p) \in K$.*

Proof. First we present some remarks we will use in the proof.

i) By the definition of δ_{push}, if $\bar{K} \xrightarrow{a} K$ in $\tilde{\mathcal{A}}$, $(\bar{q}, q) \in K$, $(q,p) \in \bar{K}$, then $q \xrightarrow{a} \bar{q}$ in $\mathcal{A}$.
ii) By the definition of δ_{shift}, if $\bar{K} \dashrightarrow^{a} K$ in $\tilde{\mathcal{A}}$, $(r,q) \in K$, then there exists a state $\bar{q} \in Q$ such that $\bar{q} \xrightarrow{a} r$ in $\mathcal{A}$ and $(\bar{q}, q) \in \bar{K}$.
iii) By the definition of δ_{pop}, if $\bar{K} \overset{K}{\Longrightarrow} K'$ in $\tilde{\mathcal{A}}$ and $(q', p) \in K'$, then there exists a pair $(r, q) \in \bar{K}$ such that $(q,p) \in K$ and $r \overset{q}{\Longrightarrow} q'$ in $\mathcal{A}$.

We argue by induction on the depth h of y. If $h = 1$, then $y = a_1 a_2 \dots a_n$ and the support can be rewritten as in (I.2.3). Let $K' \ni (q', p)$; then, by Remark (iii) there exists a pair $(q_n, q) \in K_n$ such that $(q,p) \in K$ and $q_n \overset{q}{\Longrightarrow} q'$ in $\tilde{\mathcal{A}}$. Moreover, $(q_n, q) \in K_n$ and $K_{n-1} \dashrightarrow^{a_n} K_n$ imply by Remark (ii) the existence of a state $q_{n-1} \in Q$ such that $(q_{n-1}, q) \in K_{n-1}$ and $q_{n-1} \dashrightarrow^{a_n} q_n$. Similarly one can verify that for every $i = n-2, \dots 1$ there exists $q_i \in Q$ such that $(q_i, q) \in K_i$ and $q_i \dashrightarrow^{a_{i+1}} q_{i+1}$. Finally, $K \xrightarrow{a_1} K_1$, $(q_1, q) \in K_1$ and $(q,p) \in K$ imply by Remark (i) that $q \xrightarrow{a_1} q_1$ in $\mathcal{A}$. Thus, we built backward a path as in (I.2.1) with $q_0 = q$, $q_{n+1} = q'$, $(q,p) \in K$, and this concludes the proof of induction basis.

Now assume that the statement holds for chains with depth lower than h. Let $y = x_0 a_1 x_1 a_2 \dots a_n x_n$ have depth h and consider a support as in (I.2.4) where $K'_i = K_i$ whenever x_i is the empty word, and every non-empty x_i has depth lower than h. Let $(q', p) \in K'$. Since $K'_n \overset{K'_0}{\Longrightarrow} K'$, by Remark (iii) there exists a pair $(q'_n, q'_0) \in K'_n$ with $(q'_0, p) \in K'_0$ and $q'_n \overset{q'_0}{\Longrightarrow} q'$ in $\tilde{\mathcal{A}}$. If $x_n \neq \varepsilon$, by the inductive hypothesis, since $(q'_n, q'_0) \in K'_n$ there exists a support $q_n \overset{x_n}{\leadsto} q'_n$ with $(q_n, q'_0) \in K_n$.

Similarly one can see that, for all $i = n-1, \dots 2, 1$, there exist q'_i and q_i ($q'_i = q_i$ whenever x_i is empty) such that

$$q_i \overset{x_i}{\leadsto} q'_i \dashrightarrow^{a_{i+1}} q_{i+1}$$

with $(q'_i, q'_0) \in K'_i$ by Remark (ii) (since $K'_i \dashrightarrow^{a_{i+1}} K_{i+1}$ in $\tilde{\mathcal{A}}$ and $(q_{i+1}, q'_0) \in K_{i+1}$), and $(q_i, q'_0) \in K_i$ by the inductive hypothesis (since $K_i \overset{x_i}{\leadsto} K'_i$ in $\tilde{\mathcal{A}}$ and $(q'_i, q'_0) \in K'_i$).

In particular $q_1 \overset{x_1}{\leadsto} q'_1$ with $(q_1, q'_0) \in K_1$. Then, since also $K'_0 \xrightarrow{a_1} K_1$ and $(q'_0, p) \in K'_0$, by Remark (i) we get $q'_0 \xrightarrow{a_1} q_1$. Finally, since $(q'_0, p) \in K'_0$ and $K \overset{x_0}{\leadsto} K'_0$, if $x_0 \neq \varepsilon$ the inductive hypothesis implies the existence of a state $q \in Q$ such that $q \overset{x_0}{\leadsto} q'_0$ in $\tilde{\mathcal{A}}$ with $(q, p) \in K$. Hence we built a support as in (I.2.2) with $q_0 = q$, $q_{n+1} = q'$ and $(q, p) \in K$, and this concludes the proof. ∎

To complete the proof of Theorem I.2.4, we prove that there exists an accepting computation for y in $\mathcal{A}$ if and only if there exists an accepting computation for y in $\tilde{\mathcal{A}}$.

Let y be in $L(\mathcal{A})$. Then it admits a support $q \overset{y}{\leadsto} q'$ with $q \in I$ and $q' \in F$. Then for $K = I \times \{\top\} \ni (q_0, \top)$, Lemma I.2.5 implies the existence of a support $K \overset{y}{\leadsto} K'$ in $\tilde{\mathcal{A}}$ with $K' \ni (q', \top)$. $q' \in F$ implies $K' \in \tilde{F}$, hence y is accepted by $\tilde{\mathcal{A}}$.

Conversely, let y be in $L(\tilde{\mathcal{A}})$. Then y admits a support $\tilde{K} \overset{y}{\leadsto} K'$ in $\tilde{\mathcal{A}}$, with $K' \in \tilde{F}$. This means that there exists $q' \in F$ such that $(q', \top) \in K'$. Hence, by Lemma I.2.6, there exists a support $q \overset{y}{\leadsto} q'$ in $\mathcal{A}$ with $(q', \top) \in \tilde{K}$, and this implies $q \in I$. Thus the support $q \overset{y}{\leadsto} q'$ defines an accepting computation for y in $\mathcal{A}$. ∎

I.2.3. Complexity of OPL decision problems.

To conclude this section we point out that the basic decision problems for OPLs have the same order of complexity as those for VPLs; precisely:

- the emptiness problem is in PTIME, OPLs and VPLs being a subclass of context-free languages;
- the containment problem for deterministic OPAs is in PTIME too since it is reduced to the intersection, complement and emptiness problems which are all in PTIME in the deterministic case;
- the containment problem in the nondeterministic case is instead EXPTIME complete: the same arguments used in [4] for VPLs apply identically for OPLs.

I.3. Operator Precedence Automata and Grammars.

Our next result is the equivalence between OPGs and OPAs.

I.3.1. From OPGs to OPAs.

Theorem I.3.1. *Let $G = \langle N, \Sigma, P, S \rangle$ be an OPG; then an OPA $\mathcal{A}$ such that $L(\mathcal{A}) = L(G)$ can effectively be built. Furthermore, let m be the sum of the lengths of the r.h.s.s of G; then $\mathcal{A}$ has $O(m^2)$ states.*

Proof. First, we describe a procedure to build a nondeterministic OPA $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ from a given OPG G with the same precedence matrix M as G. Then we prove the equivalence between $\mathcal{A}$ and G.

The construction sharply differs from the traditional one involving CF grammars and general pushdown automata, which is instead quite straightforward. This is due to the remarkable peculiarities of OPAs –among them the locality principle– which make them, in turn, significantly different from the more powerful general pushdown automata and from the less powerful VPAs. To keep the construction as simple as possible, we avoid introducing any optimization. Also, without loss of generality, we assume that the grammar G has no empty nor renaming rules.

$\mathcal{A}$ is built in such a way that a successful computation thereof corresponds to building bottom-up a derivation tree in G: the automaton performs a push transition when it reads the first terminal of a new r.h.s. It performs a shift transition when it reads a terminal symbol inside a r.h.s, i.e. a leaf with some left sibling leaf. It performs a pop transition when it completes the recognition of a r.h.s., then guesses (nondeterministically) the nonterminal at the l.h.s. Each state contains two pieces of information: the first component represents the prefix of the r.h.s under construction, whereas the second component is used to recover the r.h.s *previously under construction* (see Figure I.3.1) whenever all r.h.s.s nested below have been completed.

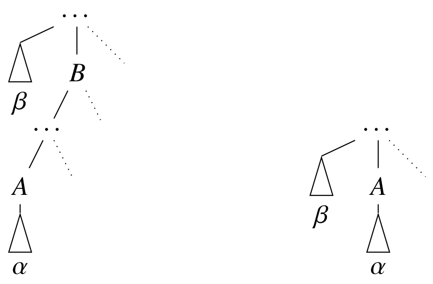


Figure I.3.1: When parsing α, the prefix previously under construction is β.

Precisely, the construction of $\mathcal{A}$ is defined as follows. Let

$$\mathbb{P} = \{\alpha \in (N \cup \Sigma)^*\Sigma \mid \exists A \to \alpha\beta \in P\}$$

be the set of prefixes, ending with a terminal symbol, of r.h.s. of G.; define $\mathbb{Q} = \{\varepsilon\} \cup \mathbb{P} \cup N$, $Q = \mathbb{Q}\times(\{\varepsilon\}\cup\mathbb{P})$, $I = \{\langle\varepsilon,\varepsilon\rangle\}$, and $F = S\times\{\varepsilon\}\cup\{\langle\varepsilon,\varepsilon\rangle \mid \varepsilon \in L(G)\}$. Note that $|\mathbb{Q}| = 1+|\mathbb{P}|+|N|$ is $O(m)$; therefore $|Q|$ is $O(m^2)$.

The transition functions are defined as follows, for $a \in \Sigma$ and $\alpha,\alpha_1,\alpha_2 \in \mathbb{Q}$, $\beta,\beta_1,\beta_2 \in \{\varepsilon\} \cup \mathbb{P}$:

- $\delta_{\text{shift}}(\langle\alpha,\beta\rangle,a) \ni \begin{cases} \langle\alpha a,\beta\rangle & \text{if } \alpha \notin N \\ \langle\beta\alpha a,\beta\rangle & \text{if } \alpha \in N \end{cases}$
- $\delta_{\text{push}}(\langle\alpha,\beta\rangle,a) \ni \begin{cases} \langle a,\alpha\rangle & \text{if } \alpha \notin N \\ \langle\alpha a,\beta\rangle & \text{if } \alpha \in N \end{cases}$
- $\delta_{\text{pop}}(\langle\alpha_1,\beta_1\rangle,\langle\alpha_2,\beta_2\rangle) \ni \langle A,\gamma\rangle$ for every A such that $\left[\begin{array}{ll} A \to \alpha_1 \in P, & \text{if } \alpha_1 \notin N \\ A \to \beta_1\alpha_1 \in P, & \text{if } \alpha_1 \in N \end{array}\right.$
  and $\gamma = \begin{cases} \alpha_2, & \text{if } \alpha_2 \notin N \\ \beta_2, & \text{if } \alpha_2 \in N. \end{cases}$

Notice that the result of δ_{shift} and δ_{push} is a singleton, whereas δ_{pop} may produce several states, in case of repeated r.h.s.s.

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 r.h.s. 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, then α is the concatenation of the first component of the previous state with the read character, and β is not changed from the previous state. 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 l.h.s, and γ is the prefix previously under construction.

The equivalence between G and $\mathcal{A}$ derives from the following Lemmata I.3.2 and I.3.3, when $\beta = \gamma = \varepsilon$, $\Pi = \bot$ and A is an axiom. ☐

Example 6. *Let G be the grammar introduced in Example 1. To apply the construction of Theorem I.3.1 first we need to transform G in such a way that there are no renaming rules.*

The new grammar has the following productions

$$\begin{aligned} E &\to E+T \mid T\times F \mid n \mid \llparenthesis E \rrparenthesis \\ T &\to T\times F \mid n \mid \llparenthesis E \rrparenthesis \\ F &\to n \mid \llparenthesis E \rrparenthesis \end{aligned}$$

where E, T, and F are axioms.

Figure I.3.2 shows an accepting computation of the equivalent automatom, together with the corresponding derivation tree. Notice that the computation shown in Figure I.3.2 is equal to that of Figure I.2.1 up to a renaming of the states; in fact the shape of syntax trees and consequently the sequence of push, shift and pop moves in OPLs depends only on the OPM, not on the visited states.

stack	state	current input
$\bot$	$\langle\varepsilon,\varepsilon\rangle$	$n+n\times\llparenthesis n+n\rrparenthesis\#$
$\bot[n,\ \langle\varepsilon,\varepsilon\rangle]$	$\langle n,\varepsilon\rangle$	$+n\times\llparenthesis n+n\rrparenthesis\#$
$\bot$	$\langle E,\varepsilon\rangle$	$+n\times\llparenthesis n+n\rrparenthesis\#$
$\bot[+,\ \langle E,\varepsilon\rangle]$	$\langle E+,\varepsilon\rangle$	$n\times\llparenthesis n+n\rrparenthesis\#$
$\bot[+,\ \langle E,\varepsilon\rangle][n,\ \langle E+,\varepsilon\rangle]$	$\langle n,\varepsilon\rangle$	$\times\llparenthesis n+n\rrparenthesis\#$
$\bot[+,\ \langle E,\varepsilon\rangle]$	$\langle T,E+\rangle$	$\times\llparenthesis n+n\rrparenthesis\#$
$\bot[+,\ \langle E,\varepsilon\rangle][\times,\ \langle T,E+\rangle]$	$\langle T\times,E+\rangle$	$\llparenthesis n+n\rrparenthesis\#$
$\bot[+,\ \langle E,\varepsilon\rangle][\times,\ \langle T,E+\rangle][\llparenthesis,\ \langle T\times,E+\rangle]$	$\langle\llparenthesis,E+\rangle$	$n+n\rrparenthesis\#$
$\bot[+,\ \langle E,\varepsilon\rangle][\times,\ \langle T,E+\rangle][\llparenthesis,\ \langle T\times,E+\rangle][n,\ \langle\llparenthesis,E+\rangle]$	$\langle n,E+\rangle$	$+n\rrparenthesis\#$
$\bot[+,\ \langle E,\varepsilon\rangle][\times,\ \langle T,E+\rangle][\llparenthesis,\ \langle T\times,E+\rangle]$	$\langle E,\llparenthesis\rangle$	$+n\rrparenthesis\#$
$\bot[+,\ \langle E,\varepsilon\rangle][\times,\ \langle T,E+\rangle][\llparenthesis,\ \langle T\times,E+\rangle][+,\ \langle E,\llparenthesis\rangle]$	$\langle E+,\llparenthesis\rangle$	$n\rrparenthesis\#$
$\bot[+,\ \langle E,\varepsilon\rangle][\times,\ \langle T,E+\rangle][\llparenthesis,\ \langle T\times,E+\rangle][+,\ \langle E,\llparenthesis\rangle][n,\ \langle E+,\llparenthesis\rangle]$	$\langle n,\llparenthesis\rangle$	$\rrparenthesis\#$
$\bot[+,\ \langle E,\varepsilon\rangle][\times,\ \langle T,E+\rangle][\llparenthesis,\ \langle T\times,E+\rangle][+,\ \langle E,\llparenthesis\rangle]$	$\langle T,E+\rangle$	$\rrparenthesis\#$
$\bot[+,\ \langle E,\varepsilon\rangle][\times,\ \langle T,E+\rangle][\llparenthesis,\ \langle T\times,E+\rangle]$	$\langle E,\llparenthesis\rangle$	$\rrparenthesis\#$
$\bot[+,\ \langle E,\varepsilon\rangle][\times,\ \langle T,E+\rangle][\rrparenthesis,\ \langle T\times,E+\rangle]$	$\langle\llparenthesis E\rrparenthesis,\llparenthesis\rangle$	$\#$
$\bot[+,\ \langle E,\varepsilon\rangle][\times,\ \langle T,E+\rangle]$	$\langle F,T\times\rangle$	$\#$
$\bot[+,\ \langle E,\varepsilon\rangle]$	$\langle T,E+\rangle$	$\#$
$\bot$	$\langle E,\varepsilon\rangle$	$\#$

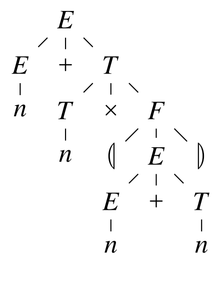


Figure I.3.2: Accepting computation of the automaton built in Theorem I.3.1.

LEMMA I.3.2. *Let x be the body of a chain and $\beta,\gamma\in\mathbb{P}\cup\{\varepsilon\}$. Then, for all $h\geq 1$, $\langle\beta,\gamma\rangle\overset{x}{\rightsquigarrow}q$ implies the existence of $A\in N$ such that $A\overset{*}{\Rightarrow}x$ in G and $q=\langle A,\beta\rangle$.*

Proof. We reason by induction on the depth h of x.

If $h = 1$, then $x = a_1 a_2 \dots a_n$ is the body of a simple chain, and the support is as in (I.2.1) with $q_0 = \langle \beta, \gamma \rangle$ and $q_{n+1} = q$. Then by the definition of push and shift transition functions we have $q_i = \langle a_1 \dots a_i, \beta \rangle$ for every $i = 1, 2, \dots n$, and by the definition of pop transition function (recall that $\beta \notin N$ by hypothesis) it is $q = \langle A, \beta \rangle$ for some A such that $A \to a_1 \dots a_n = x$ is in P. Hence $A \overset{*}{\Rightarrow} x$ and the statement is proved.

If $h > 1$, then as usual let $x = x_0 a_1 x_1 \dots a_n x_n$ and let its support be decomposed as in (I.2.2) with $q_0 = \langle \beta, \gamma \rangle$ and $q_{n+1} = q$. Also set $q_i = \langle \beta_i, \gamma_i \rangle$ for $i = 0, 1, \dots, n$ (in particular $\beta_0 = \beta$ and $\gamma_0 = \gamma$). Each non-empty x_i being the body of a chain with depth lower than h, the inductive hypothesis implies that there exists $X_i \in N$ such that $X_i \overset{*}{\Rightarrow} x_i$ in G, and $q_i = \langle X_i, \beta_i \rangle$. Thus, the support can be rewritten as

$$\langle \beta, \gamma \rangle \overset{x_0}{\rightsquigarrow} q'_0 \xrightarrow{a_1} \langle \beta_1, \gamma_1 \rangle \overset{x_1}{\rightsquigarrow} q'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} \langle \beta_n, \gamma_n \rangle \overset{x_n}{\rightsquigarrow} q'_n \overset{q'_0}{\Longrightarrow} q$$

where

$$q'_i = \begin{cases} \langle \beta_i, \gamma_i \rangle & \text{if } x_i = \varepsilon \\ \langle X_i, \beta_i \rangle & \text{otherwise} \end{cases}$$

for every i. Now, by the definition of push and shift transition functions, one can see that, for $i \neq 0$, $\beta_i = X_0 a_1 \dots X_{i-1} a_i$ holds regardless of whether x_i is empty or not (setting $X_i = \varepsilon$ if $x_i = \varepsilon$). Thus, to compute the state q reached with the final pop transition $\delta_{\text{pop}}(q'_n, q'_0)$, we have to consider four cases depending on whether x_0 and x_n are empty or not, which are exactly the four combinations considered in the definition of δ_{pop}. In any case, q has the form $\langle A, \beta \rangle$, where A is a nonterminal of G such that $A \to X_0 a_1 X_1 \dots X_{n_1} a_n X_n$. ☐

Lemma I.3.3. *Let x be the body of a chain and $A \in N$. Then, $A \overset{*}{\Rightarrow} x$ in G implies $\langle \beta, \gamma \rangle \overset{x}{\rightsquigarrow} \langle A, \beta \rangle$ for every $\beta, \gamma \in \mathbb{P} \cup \{\varepsilon\}$.*

Proof. We reason by induction on the depth h of the chain. If $h = 1$, then x is the body of a simple chain, hence $A \overset{*}{\Rightarrow} x$ means that $A \to x$ is a production. Thus, by the definition of δ (recall that $\beta \notin N$ by hypothesis), we obtain a support as in (I.2.1) with $q_0 = \langle \beta, \gamma \rangle$, $q_{n+1} = q$, and $q_i = \langle a_1 \dots a_i, \beta \rangle$ for every $i = 1, 2, \dots n$.

If $h > 1$, then x is the body of a composed chain with $x = x_0 a_1 x_1 \dots a_n x_n$. Hence $A \overset{*}{\Rightarrow} x$ in G implies that there exist $X_0, X_1, \dots, X_n \in \{\varepsilon\} \cup N$ (more precisely: $X_i = \varepsilon$ if $x_i = \varepsilon$) such that $A \to X_0 a_1 X_1 \dots a_n X_n$ and $X_i \overset{*}{\Rightarrow} x_i$. The first step of the computation is different depending on whether x_0 is empty or not. In any case, we have

$$\langle \beta, \gamma \rangle \overset{x_0}{\rightsquigarrow} q'_0 \xrightarrow{a_1} \langle X_0 a_1, \beta \rangle, \qquad \text{where} \qquad q'_0 = \begin{cases} \langle \beta, \gamma \rangle & \text{if } x_0 = \varepsilon \\ \langle X_0, \beta \rangle & \text{otherwise} \end{cases}$$

The computation goes on differently depending on whether $x_1, x_2, \dots, x_{n-1}$ are empty or not. However, by the inductive hypothesis and the definition of δ_{shift}, after reading a_i the automaton reaches state $\langle X_0 a_1 \dots X_{i-1} a_i, \beta \rangle$ for every $i = 1, \dots, n$, i.e., we have the path

$$\langle \beta, \gamma \rangle \overset{x_0}{\rightsquigarrow} q'_0 \xrightarrow{a_1} \langle X_0 a_1, \beta \rangle \overset{x_1}{\rightsquigarrow} q'_1 \xrightarrow{a_2} \langle X_0 a_1 X_1 a_2, \beta \rangle \overset{x_2}{\rightsquigarrow} q'_2 \xrightarrow{a_3} \dots \xrightarrow{a_n} \langle X_0 a_1 \dots X_{n-1} a_n, \beta \rangle.$$

If $x_n \neq \varepsilon$, the computation proceeds with the last inductive step

$$\langle X_0 a_1 \dots X_{n-1} a_n \ , \ \beta_n \rangle \overset{x_n}{\rightsquigarrow} \langle X_n \ , \ X_0 a_1 \dots X_{n-1} a_n \rangle.$$

Finally, the computation ends with a pop transition. There are four cases depending on whether x_0 and x_n are empty or not, which are exactly the four combinations considered in the definition of δ_{pop}. In any case, we build a support ending with state $\langle A,\beta\rangle$, and this concludes the proof. □

Corollary I.3.4. *If the source grammar is in FNF, then the corresponding automaton is deterministic.*

The thesis follows immediately by observing that the construction defined in Theorem I.3.1 is such that the values defined by δ_{push} and δ_{shift} are always singleton, whereas δ_{pop} produces as many states as many l.h.s.s have the same r.h.s. Thus, since the initial state is a singleton and grammars in FNF have no repeated r.h.s.s, the automaton resulting from the construction is already deterministic. This corollary has an interesting effect in terms of size of the produced automata as pointed out below.

Remark 1. *Given a grammar G with $|N|$ nonterminals the construction of Theorem I.3.1 produces an automaton with $O(m^2)$ states, where m is defined as in Theorem I.3.1; thus, if we build a deterministic OPA from a generic OPG G by first building a nondeterministic automaton and then transforming it in deterministic version, we obtain an automaton with $2^{O(m^4)}$ states; instead, if we first transform the original G in FNF we obtain an equivalent grammar $\tilde{G}$ with $O(2^{|N|})$ nonterminals and $\tilde{m} = O(2^{m|N|^2})$; then, by applying the construction of Theorem I.3.1 we directly obtain a deterministic automaton with $O(\tilde{m}^2) = O(2^{2m|N|^2})$ states.*

Nevertheless, the size of the complete automaton is clearly hardly manageable by human execution; thus we implemented a prototype (non-optimized) tool to perform the construction[6].

I.3.2. From OPAs to OPGs.

The construction of an 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.

Theorem I.3.5. *Let $\mathcal{A}$ be an OPA; then an OPG G such that $L(G) = L(\mathcal{A})$ can effectively be 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 support of type (I.2.1) of a simple chain, the rule

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

  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 support of type (I.2.2) 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; 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.

[6]The tool is called *Flup*, available at *https://github.com/bzoto/flup*.

Notice that the above construction is effective thanks to the hypothesis of $\doteq$-acyclicity of the OPM (remind that, as discussed in Section I.1, this hypothesis could be replaced by weaker ones). This implies that the length of the r.h.s. is bounded (see Section I.1); on the other hand, the cardinality of the nonterminal alphabet is finite (precisely it is $O(|\Sigma|^2 \cdot |Q|^2)$. Hence there is only a finite number of possible productions for G and only a limited number of chains to be considered. ☐

I.4. Monadic Second-order Logic Characterization. In his seminal paper [12] Büchi provided a logic characterization of regular languages: he defined a MSO syntax on the integers representing the position of characters within a string and, by means of clever arguments, gave algorithms to build a Finite State Machine (FSM) recognizing exactly the strings satisfying a given formula and, conversely, to build a formula satisfied by all and only the strings accepted by a given FSM. Subsequently, a rich literature considerably extended his work to more powerful language families –typically, context-free [13]– and different logic formalisms, e.g., first-order or tree logics [1, 10, 16]. To the best of our knowledge, MSO logic characterizations of CF languages refer to "visible structure languages" i.e. to languages whose strings make their syntactic structure immediately visible in their external appearance, such as "tree-languages" [39][7] and Visibly Pushdown Languages [3] which explicitly refer to this peculiar property in their name. In this section we provide a complete MSO logic characterization of OPLs, which, instead, include also invisible-structure languages, whose syntax trees associated with external strings must be built by means of suitable parsing algorithms, in which the OPM plays a major role.

Similarly to other approaches, in particular to the VPLs one, which in fact are a subclass of OPLs, we begin by defining a suitable binary predicate on the string positions. However the original definition of [4] which states the $\leadsto$ relation between the positions of two matching parentheses (calls and returns in VPLs terminology) cannot be naturally extended to the more general case of OPL strings. In fact the $\leadsto$ relation between two matching parentheses, which are extremes of the frontier of a sub-tree, is typically one-to-one (with the exclusion of the particular case of unmatched parentheses which however occur only at the begin and end of a string) whereas in general the relation between leftmost and rightmost leaves of an OPL sub-tree can be many-to-one or one-to-many or both. A further consequence of the more general structure of OPL trees is that, unlike FSMs, tree automata, and Visibly Pushdown Automata (VPAs), OPAs are not real-time automata as they may have to perform a series of pop moves without advancing their running head; this in turn produces the effect that, whereas in regular and VPLs each position is associated with a unique state visited by the machine during its behavior, for OPLs the same position may refer to several states –i.e. to several subsets of positions according to Büchi's approach.

Consequently, the approach we describe here departs from previous ones along two main directions:

- The binary relations between positions referring to a pop operation are attached to the look-back and look-ahead positions which in OP parsing embrace the r.h.s. to be reduced; thus, the formal definition of the relation will be based on the notion of chain.
- The sets of positions associated with the different automaton states are subdivided into three, not necessarily disjoint, subsets: one describing the state reached after a push or shift operation, and two to delimit the positions corresponding to each pop operation; in such a way we obtain a unique identification thereof.

[7] It is not coincidence if tree automata [39] have been defined by extending the original finite state ones.

I.4.1. A Monadic Second-Order Logic over Operator Precedence Alphabets. Let (Σ, M) be an OP alphabet. Let us define a countable infinite set of first-order variables $\boldsymbol{x}, \boldsymbol{y}, \ldots$ and a countable infinite set of monadic second-order (set) variables $\boldsymbol{X}, \boldsymbol{Y}, \ldots$. In the following we adopt the convention to denote first and second-order variables in boldface italic font.

Definition I.4.1 (Monadic Second-order Logic over (Σ, M)). *Let $\mathcal{V}_1$ be a set of first-order variables, and $\mathcal{V}_2$ be a set of second-order (or set) variables. The $MSO_{\Sigma,M}$ (*monadic second-order logic *over (Σ, M)) is defined by the following syntax (symbols Σ, M will be omitted unless necessary to prevent confusion):*

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

where $c \in \Sigma \cup \{\#\}$, $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{V}_1$, and $\boldsymbol{X} \in \mathcal{V}_2$.[8]

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

- $\#w\#, M, \nu_1, \nu_2 \models c(\boldsymbol{x})$ iff $\#w\# = w_1 c w_2$ and $|w_1| = \nu_1(\boldsymbol{x})$.
- $\#w\#, M, \nu_1, \nu_2 \models \boldsymbol{x} \in \boldsymbol{X}$ iff $\nu_1(\boldsymbol{x}) \in \nu_2(\boldsymbol{X})$.
- $\#w\#, M, \nu_1, \nu_2 \models \boldsymbol{x} \le \boldsymbol{y}$ iff $\nu_1(\boldsymbol{x}) \le \nu_1(\boldsymbol{y})$.
- $\#w\#, M, \nu_1, \nu_2 \models \boldsymbol{x} \curvearrowright \boldsymbol{y}$ iff $\#w\# = w_1 a w_2 b w_3$, $|w_1| = \nu_1(\boldsymbol{x})$, $|w_1 a w_2| = \nu_1(\boldsymbol{y})$, and $a w_2 b$ is a chain ${}^a[w_2]^b$.
- $\#w\#, M, \nu_1, \nu_2 \models \neg\varphi$ iff $\#w\#, M, \nu_1, \nu_2 \not\models \varphi$.
- $\#w\#, M, \nu_1, \nu_2 \models \varphi_1 \vee \varphi_2$ iff $\#w\#, M, \nu_1, \nu_2 \models \varphi_1$ or $\#w\#, M, \nu_1, \nu_2 \models \varphi_2$.
- $\#w\#, M, \nu_1, \nu_2 \models \exists \boldsymbol{x}.\varphi$ iff $\#w\#, M, \nu_1', \nu_2 \models \varphi$, for some ν_1' with $\nu_1'(\boldsymbol{y}) = \nu_1(\boldsymbol{y})$ for all $\boldsymbol{y} \in \mathcal{V}_1 \setminus \{\boldsymbol{x}\}$.
- $\#w\#, M, \nu_1, \nu_2 \models \exists \boldsymbol{X}.\varphi$ iff $\#w\#, M, \nu_1, \nu_2' \models \varphi$, for some ν_2' with $\nu_2'(\boldsymbol{Y}) = \nu_2(\boldsymbol{Y})$ for all $\boldsymbol{Y} \in \mathcal{V}_2 \setminus \{\boldsymbol{X}\}$.

To improve readability, we will drop M, ν_1, ν_2 and the delimiters # from the notation whenever there is no risk of ambiguity; furthermore we use some standard abbreviations in formulae, such as $\boldsymbol{x} + 1$, $\boldsymbol{x} - 1$, $\boldsymbol{x} = \boldsymbol{y}$, $\boldsymbol{x} \ne \boldsymbol{y}$, $\boldsymbol{x} < \boldsymbol{y}$.

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

$$L(\varphi) = \{w \in \Sigma^* \mid w \models \varphi\}.$$

Figure I.4.1 illustrates the meaning of the $\curvearrowright$ relation with reference to the string of Figure I.1.1: we have $0 \curvearrowright 2$, $2 \curvearrowright 4$, $5 \curvearrowright 7$, $7 \curvearrowright 9$, $5 \curvearrowright 9$, $4 \curvearrowright 10$, $2 \curvearrowright 10$, and $0 \curvearrowright 10$. Such pairs correspond to contexts where a reduce operation is executed during the parsing of the string (they are listed according to their execution order).

In general $\boldsymbol{x} \curvearrowright \boldsymbol{y}$ implies $\boldsymbol{y} > \boldsymbol{x} + 1$, and a position $\boldsymbol{x}$ may be in such a relation with more than one position and vice versa. Moreover, if w is compatible with M, then $0 \curvearrowright |w| + 1$.

Example 7. *Consider the language of Example 1. The following sentence states that all parentheses are well-matched:*

$$\forall \boldsymbol{x} \forall \boldsymbol{y} \left(\boldsymbol{x} \curvearrowright \boldsymbol{y} \Rightarrow \left(\llparenthesis(\boldsymbol{x}+1) \Rightarrow \begin{array}{c} \rrparenthesis(\boldsymbol{y}-1) \wedge \\ \neg\exists \boldsymbol{z}(\boldsymbol{z} < \boldsymbol{y} \wedge \boldsymbol{x} \curvearrowright \boldsymbol{z}) \wedge \\ \neg\exists \boldsymbol{v}(\boldsymbol{x} < \boldsymbol{v} \wedge \boldsymbol{v} \curvearrowright \boldsymbol{y}) \end{array} \right) \right).$$

Note that this property is guaranteed a priori by the structure of the OPM.

[8]This is the usual MSO over strings, augmented with the $\curvearrowright$ predicate.

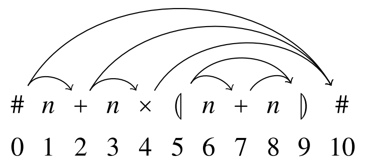


Figure I.4.1: The string of Figure I.1.1, with positions and relation $\curvearrowright$.

The following sentence instead defines the language where parentheses are used only when they are needed (i.e. to give precedence of + over ×).

$$\forall \boldsymbol{x} \forall \boldsymbol{y} \left(\begin{array}{c} \boldsymbol{x} \curvearrowright \boldsymbol{y} \wedge \\ \llparenthesis(\boldsymbol{x}+1) \wedge \rrparenthesis(\boldsymbol{y}-1) \end{array} \Rightarrow (\times(\boldsymbol{x}) \vee \times(\boldsymbol{y})) \wedge \exists \boldsymbol{z} \left(\begin{array}{c} \boldsymbol{x}+1 < \boldsymbol{z} < \boldsymbol{y}-1 \wedge +(\boldsymbol{z}) \wedge \\ \neg \exists \boldsymbol{u} \exists \boldsymbol{v} \left(\begin{array}{c} \boldsymbol{x}+1 < \boldsymbol{u} < \boldsymbol{z} \wedge \llparenthesis(\boldsymbol{u}) \wedge \\ \boldsymbol{z} < \boldsymbol{v} < \boldsymbol{y}-1 \wedge \rrparenthesis(\boldsymbol{v}) \wedge \\ \boldsymbol{u}-1 \curvearrowright \boldsymbol{v}+1 \end{array} \right) \end{array} \right) \right)$$

The following theorem states the main result of this section.

THEOREM I.4.2. *A language L over (Σ, M) is an OPL if and only if there exists a MSO sentence φ such that $L = L(\varphi)$.*

The proof is constructive and structured in the following two subsections.

I.4.2. From MSO to OPAs.

STATEMENT I.4.1. *Let (Σ, M) be an operator precedence alphabet and φ be a MSO sentence. Then $L(\varphi)$ can be recognized by an OPA over (Σ, M).*

Proof. The proof follows the one by Thomas [40] and is composed of two steps: first the formula is rewritten so that no predicate symbols nor first order variables are used; then an equivalent OPA is built inductively.

Let Σ be $\{a_1, a_2, \ldots, a_n\}$. For each predicate symbol a_i we introduce a fresh set variable $\boldsymbol{X}_i$, therefore formula $a_i(\boldsymbol{x})$ will be translated into $\boldsymbol{x} \in \boldsymbol{X}_i$. Following the standard construction of [40], we also translate every first order variable into a fresh second order variable with the additional constraint that the set it represents contains exactly one position. The only difference is that formulae like $\boldsymbol{x} \curvearrowright \boldsymbol{y}$ will be translated into formulae $\boldsymbol{X}_i \curvearrowright \boldsymbol{X}_j$, where $\boldsymbol{X}_i$, $\boldsymbol{X}_j$ are singleton sets. In this case, the semantics of $\curvearrowright$ is naturally extended to second order variables that are singletons.

Let φ' be the formula obtained from φ by such a translation, and consider any subformula ψ of φ': let $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n, \boldsymbol{X}_{n+1}, \ldots \boldsymbol{X}_{n+m(\psi)}$ be the (second order) free variables appearing in ψ. Recall that $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ represent symbols in Σ, hence they are never quantified.

As usual we interpret formulae over strings; in this case we use the alphabet

$$\Lambda(\psi) = \left\{ \alpha \in \{0,1\}^{n+m(\psi)} \mid \exists! i \text{ s.t. } 1 \leq i \leq n,\ \alpha_i = 1 \right\}$$

A string $w \in \Lambda(\psi)^*$, with $|w| = \ell$, is used to interpret ψ in the following way: the projection over the j-th component of $\Lambda(\psi)$ gives a valuation $\{1, 2, \ldots, \ell\} \to \{0, 1\}$ of $\boldsymbol{X}_j$, for every $1 \leq j \leq n + m(\psi)$.

For any $\alpha \in \Lambda(\psi)$, the projection of α over the first n components encodes a symbol in Σ, denoted as $symb(\alpha)$. The matrix M over Σ can be naturally extended to the OPM $M(\psi)$ over $\Lambda(\psi)$ by defining $M(\psi)_{\alpha,\beta} = M_{symb(\alpha), symb(\beta)}$ for any $\alpha, \beta \in \Lambda(\psi)$.

We now build an OPA $\mathcal{A}$ equivalent to φ'. The construction is inductive on the structure of the formula: first we define the OPA for all atomic formulae. We give here only the

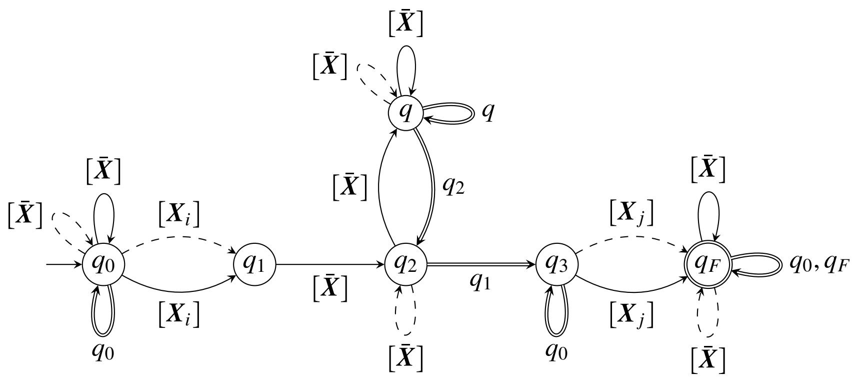


Figure I.4.2: OPA for atomic formula $\psi = \boldsymbol{X}_i \curvearrowright \boldsymbol{X}_j$

construction for $\curvearrowright$, since for the other ones the construction is standard and is the same as in [40].

Figure I.4.2 represents the OPA for atomic formula $\psi = \boldsymbol{X}_i \curvearrowright \boldsymbol{X}_j$ (notice that $i, j > n$, and that both $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ are singleton sets). For the sake of brevity, we use notation $[\boldsymbol{X}_i]$ to represent the set of all tuples $\Lambda(\psi)$ having the i-th component equal to 1; notation $[\bar{\boldsymbol{X}}]$ represents the set of all tuples in $\Lambda(\psi)$ having both i-th and j-th components equal to 0.

The semantics of $\curvearrowright$ requires for $\boldsymbol{X}_i \curvearrowright \boldsymbol{X}_j$ that there must be a chain ${}^a[w_2]^b$ in the input word, where a is the symbol at the only position in $\boldsymbol{X}_i$, and b is the symbol at the only position in $\boldsymbol{X}_j$. By definition of chain, this means that a must be read, hence in the position represented by $\boldsymbol{X}_i$ the automaton performs either a push or a shift move (see Figure I.4.2, from state q_0 to q_1), as pop moves do not consume input. After that, the automaton must read w_2. In order to process the chain ${}^a[w_2]^b$, reading w_2 must start with a push move (from state q_1 to state q_2), and it must end with one or more pop moves, before reading b (i.e. the only position in $\boldsymbol{X}_j$ – going from state q_3 to q_F).

This means that the automaton, after a generic sequence of moves corresponding to visiting an irrelevant (for $\boldsymbol{X}_i \curvearrowright \boldsymbol{X}_j$) portion of the syntax tree, when reading the symbol at position $\boldsymbol{X}_i$ performs either a push or a shift move, depending on whether $\boldsymbol{X}_i$ is the position of a leftmost leaf of the tree or not. Then it visits the subsequent subtree ending with a pop labeled q_1; at this point, if it reads the symbol at position $\boldsymbol{X}_j$, it accepts anything else that follows the examined fragment.

Then, a natural inductive path leads to the construction of the automaton associated with a generic MSO logic formula: the disjunction of two subformulae can be obtained by building the union automaton of the two corresponding automata; similarly for negation. The existential quantification of $\boldsymbol{X}_i$ is obtained by projection erasing the i-th component; since OPLs are closed under alphabetical homomorphisms preserving the OPM (see Statement I.1.2), and since the OPM is determined only by the first n components of the alphabet's elements which are never erased by quantification, such a projection produces a well defined automaton for any ψ. Finally, the alphabet of the automaton equivalent to φ' is $\Lambda(\varphi') = \{0, 1\}^n$, which is in bijection with Σ. □

I.4.3. From OPAs to MSO.

When considering a chain ${}^a[w]^b$ we assume $w = w_0 a_1 w_1 \dots a_\ell w_\ell$, with ${}^a[a_1 a_2 \dots a_\ell]^b$ being a simple chain (any w_g may be empty). We denote by s_g the position of symbol a_g, for $g = 1, 2, \dots, \ell$ and set $a_0 = a$, $s_0 = 0$, $a_{\ell+1} = b$, and $s_{\ell+1} = |w| + 1$. Furthermore, we define the following shortcut notations:

$$\boldsymbol{x} \circ \boldsymbol{y} := \bigvee_{M_{a,b}=\circ} a(\boldsymbol{x}) \wedge b(\boldsymbol{y}), \text{ for } \circ \in \{\lessdot, \doteq, \gtrdot\}$$

$$\mathrm{Tree}(\boldsymbol{x},\boldsymbol{z},\boldsymbol{v},\boldsymbol{y}) := \boldsymbol{x} \curvearrowright \boldsymbol{y} \wedge \left(\begin{array}{c} (\boldsymbol{x}+1=\boldsymbol{z} \ \vee \ \boldsymbol{x} \curvearrowright \boldsymbol{z}) \wedge \neg\exists \boldsymbol{t}(\boldsymbol{z} < \boldsymbol{t} < \boldsymbol{y} \wedge \boldsymbol{x} \curvearrowright \boldsymbol{t}) \\ \wedge \\ (\boldsymbol{v}+1=\boldsymbol{y} \ \vee \ \boldsymbol{v} \curvearrowright \boldsymbol{y}) \wedge \neg\exists \boldsymbol{t}(\boldsymbol{x} < \boldsymbol{t} < \boldsymbol{v} \wedge \boldsymbol{t} \curvearrowright \boldsymbol{y}) \end{array} \right)$$

If $\boldsymbol{x} \curvearrowright \boldsymbol{y}$ then there exist (unique) $\boldsymbol{z}$ and $\boldsymbol{v}$ such that $\mathrm{Tree}(\boldsymbol{x},\boldsymbol{z},\boldsymbol{v},\boldsymbol{y})$ is satisfied. In particular, if w is the body of a simple chain, then $0 \curvearrowright \ell+1$ and $\mathrm{Tree}(0,1,\ell,\ell+1)$ are satisfied; if it is the body of a composed chain, then $0 \curvearrowright |w|+1$ and $\mathrm{Tree}(0,s_1,s_\ell,s_{\ell+1})$ are satisfied. If $w_0 = \varepsilon$ then $s_1 = 1$, and if $w_\ell = \varepsilon$ then $s_\ell = |w|$. In the example of Figure I.4.1 relations $\mathrm{Tree}(2,3,3,4)$, $\mathrm{Tree}(2,4,4,10)$, $\mathrm{Tree}(4,5,9,10)$, $\mathrm{Tree}(5,7,7,9)$ are satisfied, among others.

Statement I.4.2. *Let (Σ, M) be an operator precedence alphabet and $\mathcal{A}$ be an OPA over (Σ, M). Then there exists an MSO sentence φ such that $L(\mathcal{A}) = L(\varphi)$.*

Proof. Let $\mathcal{A} = \langle \Sigma, M, Q, q_0, F, \delta \rangle$ be *deterministic* (this simplifying assumption does not cause loss of generality, since nondeterministic OPAs are equivalent to deterministic ones by Theorem I.2.4). W.l.o.g. we also assume that the transition function of $\mathcal{A}$ is total. We build a MSO sentence φ such that $L(\mathcal{A}) = L(\varphi)$. The main idea for encoding the behavior of the OPA is based on assigning the states visited during its run to positions along the same lines stated by Büchi [40] and extended for VPLs [4]. Unlike finite state automata and VPAs, however, OPAs do not work on-line. Hence, it is not possible to assign a single state to every position. Let $Q = \{q_0, q_1, \dots, q_N\}$ be the states of $\mathcal{A}$ with q_0 initial; as usual, we will use second order variables to encode them. We shall need three different sets of second order variables, namely $\boldsymbol{A}_0, \boldsymbol{A}_1, \dots, \boldsymbol{A}_N$, $\boldsymbol{B}_0, \boldsymbol{B}_1, \dots, \boldsymbol{B}_N$ and $\boldsymbol{C}_0, \boldsymbol{C}_1, \dots, \boldsymbol{C}_N$. Set $\boldsymbol{A}_i$ contains those positions of word w where state q_i may be assumed after a shift or push transition, i.e. after a transition that "consumes" an input symbol. Sets $\boldsymbol{B}_i$ and $\boldsymbol{C}_i$ encode a pop transition concluding the reading of the body $w_0 a_1 w_1 \dots a_l w_l$ of a chain whose support ends in a state q_i: set $\boldsymbol{B}_i$ contains the position of symbol a that precedes the corresponding push, whereas $\boldsymbol{C}_i$ contains the position of a_l, which is the symbol on top of the stack when the automaton performs the pop move. Figure I.4.3 presents such sets for the example automaton of Figure I.2.1, with the same input as in Figure I.4.1. Notice that each position, except the last one, belongs to exactly one $\boldsymbol{A}_i$, whereas it may belong to several $\boldsymbol{B}_i$ and at most one $\boldsymbol{C}_i$.

Then, sentence φ is defined as follows

$$\varphi := \exists \boldsymbol{e} \ \begin{array}{l} \exists \boldsymbol{A}_0, \boldsymbol{A}_1, \dots, \boldsymbol{A}_N \\ \exists \boldsymbol{B}_0, \boldsymbol{B}_1, \dots, \boldsymbol{B}_N \\ \exists \boldsymbol{C}_0, \boldsymbol{C}_1, \dots, \boldsymbol{C}_N \end{array} \left(\mathrm{Start}_0 \ \wedge \ \varphi_\delta \ \wedge \ \bigvee_{q_f \in F} \mathrm{End}_f \right), \tag{I.4.1}$$

where the first and last subformulae encode the initial and final states of the run, respectively; formula φ_δ is defined as $\varphi_{\delta_{\text{push}}} \wedge \varphi_{\delta_{\text{shift}}} \wedge \varphi_{\delta_{\text{pop}}}$ and encodes the three transition functions of the automaton, which are expressed as the conjunction of *forward* and *backward* formulae. Variable $\boldsymbol{e}$ is used to refer to the *end* of a string.

To complete the definition of φ, we incrementally introduce more notations.

$$\begin{aligned} \mathrm{Succ}_k(\boldsymbol{x},\boldsymbol{y}) &:= \boldsymbol{x}+1=\boldsymbol{y} \wedge \boldsymbol{x} \in \boldsymbol{A}_k \\ \mathrm{Next}_k(\boldsymbol{x},\boldsymbol{y}) &:= \boldsymbol{x} \curvearrowright \boldsymbol{y} \wedge \boldsymbol{x} \in \boldsymbol{B}_k \wedge \exists \boldsymbol{z}, \boldsymbol{v} \ (\mathrm{Tree}(\boldsymbol{x},\boldsymbol{z},\boldsymbol{v},\boldsymbol{y}) \wedge \boldsymbol{v} \in \boldsymbol{C}_k) \\ Q_i(\boldsymbol{x},\boldsymbol{y}) &:= \mathrm{Succ}_i(\boldsymbol{x},\boldsymbol{y}) \vee \mathrm{Next}_i(\boldsymbol{x},\boldsymbol{y}) \end{aligned}$$

The shortcut $Q_i(\boldsymbol{x},\boldsymbol{y})$ is used to represent that $\mathcal{A}$ is in state q_i when at position $\boldsymbol{x}$ and the next position to read, possibly after scanning a chain, is $\boldsymbol{y}$. Since the automaton is not real time,

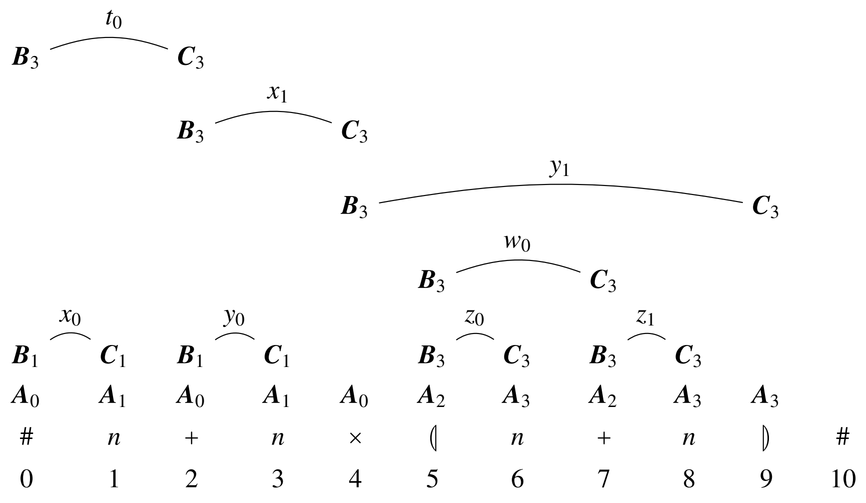

t_0
B_3 C_3
x_1
B_3 C_3
y_1
B_3 C_3
w_0
B_3 C_3
x_0 y_0 z_0 z_1
B_1 C_1 B_1 C_1 B_3 C_3 B_3 C_3
A_0 A_1 A_0 A_1 A_0 A_2 A_3 A_2 A_3 A_3
# n + n × (n + n) #
0 1 2 3 4 5 6 7 8 9 10

Figure I.4.3: The string of Figure I.1.1 with $\boldsymbol{B}_i$, $\boldsymbol{A}_i$, and $\boldsymbol{C}_i$ evidenced for the automaton of Figure I.2.1. Pop moves of the automaton are represented by linked pairs $\boldsymbol{B}_i$, $\boldsymbol{C}_i$; labels refer to supports of Figure I.2.2.

we must distinguish between push and shift moves (case $\text{Succ}_i(\boldsymbol{x},\boldsymbol{y})$), and pop moves (case $\text{Next}_i(\boldsymbol{x},\boldsymbol{y})$). For instance, with reference to Figures I.4.1 and I.4.3, $\text{Succ}_2(5,6)$, $\text{Next}_3(5,9)$, and $\text{Next}_3(5,7)$ hold.

The shortcuts representing the initial and final states of the parsing of a string of length $\boldsymbol{e}$ are defined as follows.

$$\text{Start}_i := 0 \in \boldsymbol{A}_i \wedge \neg \bigvee_{j \neq i} (0 \in \boldsymbol{A}_j)$$

$$\text{End}_f := \neg \exists \boldsymbol{y}(\boldsymbol{e}+1 < \boldsymbol{y}) \;\wedge\; \text{Next}_f(0, \boldsymbol{e}+1) \;\wedge\; \neg \bigvee_{j \neq f} (\text{Next}_j(0, \boldsymbol{e}+1)).$$

$\varphi_{\delta_{\text{push}}}$ is the conjunction of the following two formulae. The former one states the sufficient condition for a position to be in a set $\boldsymbol{A}_i$, when performing a push move.

$$\varphi_{push\_fw} := \forall \boldsymbol{x},\boldsymbol{y} \bigwedge_{i=0}^{N} \bigwedge_{k=0}^{N} \left(\boldsymbol{x} \lessdot \boldsymbol{y} \wedge c(\boldsymbol{y}) \wedge Q_i(\boldsymbol{x},\boldsymbol{y}) \wedge \delta_{\text{push}}(q_i, c) = q_k \;\Rightarrow \boldsymbol{y} \in \boldsymbol{A}_k\right)$$

The latter formula states the symmetric necessary condition

$$\varphi_{push\_bw} := \forall \boldsymbol{x},\boldsymbol{y} \bigwedge_{k=0}^{N} \left(\begin{array}{c} \boldsymbol{x} \lessdot \boldsymbol{y} \;\wedge\; c(\boldsymbol{y}) \;\wedge\; \boldsymbol{y} \in \boldsymbol{A}_k \\ \wedge \\ (\boldsymbol{x}+1 = \boldsymbol{y} \;\vee\; \boldsymbol{x} \curvearrowright \boldsymbol{y}) \end{array} \Rightarrow \bigvee_{i=0}^{N} \left(Q_i(\boldsymbol{x},\boldsymbol{y}) \wedge \delta_{\text{push}}(q_i, c) = q_k\right) \right)$$

$\varphi_{\delta_{\text{shift}}}$ is defined analogously, with respect to shift moves instead of push moves.

$$\varphi_{shift\_fw} := \;\forall \boldsymbol{x},\boldsymbol{y} \bigwedge_{i=0}^{N} \bigwedge_{k=0}^{N} (\;\boldsymbol{x} \doteq \boldsymbol{y} \;\wedge\; c(\boldsymbol{y}) \wedge Q_i(\boldsymbol{x},\boldsymbol{y}) \;\wedge\; \delta_{\text{shift}}(q_i, c) = q_k \;\Rightarrow \boldsymbol{y} \in \boldsymbol{A}_k)$$

$$\varphi_{shift\_bw} := \forall \boldsymbol{x},\boldsymbol{y} \bigwedge_{k=0}^{N} \left(\begin{array}{c} \boldsymbol{x} \doteq \boldsymbol{y} \;\wedge\; c(\boldsymbol{y}) \;\wedge\; \boldsymbol{y} \in \boldsymbol{A}_k \\ \wedge \\ (\boldsymbol{x}+1 = \boldsymbol{y} \;\vee\; \boldsymbol{x} \curvearrowright \boldsymbol{y}) \end{array} \Rightarrow \bigvee_{i=0}^{N} \left(Q_i(\boldsymbol{x},\boldsymbol{y}) \wedge \delta_{\text{shift}}(q_i, c) = q_k\right) \right)$$

Finally, to define $\varphi_{\delta_{\text{pop}}}$ we introduce the shortcut $\text{Tree}_{i,j}(\boldsymbol{x},\boldsymbol{z},\boldsymbol{v},\boldsymbol{y})$, which represents the fact that $\mathcal{A}$ is ready to perform a pop transition from state q_i having on top of the stack state q_j; such pop transition corresponds to the reduction of the portion of string between positions $\boldsymbol{x}$ and $\boldsymbol{y}$ (excluded).

$$\text{Tree}_{i,j}(\boldsymbol{x},\boldsymbol{z},\boldsymbol{v},\boldsymbol{y}) := \text{Tree}(\boldsymbol{x},\boldsymbol{z},\boldsymbol{v},\boldsymbol{y}) \wedge Q_i(\boldsymbol{v},\boldsymbol{y}) \wedge Q_j(\boldsymbol{x},\boldsymbol{z}).$$

Formula $\varphi_{\delta_{\text{pop}}}$ is thus defined as the conjunction of three formulae. As before, the forward formula gives the sufficient conditions for two positions to be in the sets $\boldsymbol{B}_k$ and $\boldsymbol{C}_k$, when performing a pop move, and the backward formulae state symmetric necessary conditions.

$$\varphi_{pop\_fw} := \forall \boldsymbol{x},\boldsymbol{z},\boldsymbol{v},\boldsymbol{y} \bigwedge_{i=0}^{N} \bigwedge_{j=0}^{N} \bigwedge_{k=0}^{N} \left(\text{Tree}_{i,j}(\boldsymbol{x},\boldsymbol{z},\boldsymbol{v},\boldsymbol{y}) \wedge \delta_{\text{pop}}(q_i,q_j) = q_k \Rightarrow \boldsymbol{x} \in \boldsymbol{B}_k \wedge \boldsymbol{v} \in \boldsymbol{C}_k\right)$$

$$\varphi_{pop\_bwB} := \forall \boldsymbol{x} \bigwedge_{k=0}^{N} \left(\boldsymbol{x} \in \boldsymbol{B}_k \Rightarrow \exists \boldsymbol{y},\boldsymbol{z},\boldsymbol{v} \bigvee_{i=0}^{N} \bigvee_{j=0}^{N} \text{Tree}_{i,j}(\boldsymbol{x},\boldsymbol{z},\boldsymbol{v},\boldsymbol{y}) \wedge \delta_{\text{pop}}(q_i,q_j) = q_k\right)$$

$$\varphi_{pop\_bwC} := \forall \boldsymbol{v} \bigwedge_{k=0}^{N} \left(\boldsymbol{v} \in \boldsymbol{C}_k \Rightarrow \exists \boldsymbol{x},\boldsymbol{y},\boldsymbol{z} \bigvee_{i=0}^{N} \bigvee_{j=0}^{N} \text{Tree}_{i,j}(\boldsymbol{x},\boldsymbol{z},\boldsymbol{v},\boldsymbol{y}) \wedge \delta_{\text{pop}}(q_i,q_j) = q_k\right)$$

Now notice that $\varphi \equiv \bigvee_{q_f \in F} \psi_{0,f}$, where

$$\psi_{i,k} := \exists \boldsymbol{e} \begin{array}{l} \exists \boldsymbol{A}_0,\boldsymbol{A}_1,\ldots,\boldsymbol{A}_N \\ \exists \boldsymbol{B}_0,\boldsymbol{B}_1,\ldots,\boldsymbol{B}_N \\ \exists \boldsymbol{C}_0,\boldsymbol{C}_1,\ldots,\boldsymbol{C}_N \end{array} (\text{Start}_i \wedge \varphi_\delta \wedge \text{End}_k)$$

Hence, the proof that $L(\mathcal{A}) = L(\varphi)$ is direct consequence of the following Lemmata I.4.3 and I.4.4, stating that $w \models \psi_{i,k}$ if and only if $q_i \overset{w}{\rightsquigarrow} q_k$ in $\mathcal{A}$, for every word w compatible with (Σ, M). ☐

LEMMA I.4.3. *Let w be the body of a chain ${}^{\#}[w]^{\#}$. If $q_i \overset{w}{\rightsquigarrow} q_k$ in $\mathcal{A}$, then $w \models \psi_{i,k}$.*

Proof. We prove the lemma by induction on the depth of chains. Note that, even if $\mathcal{A}$ is deterministic, some chains could have different supports. However, we will show that every support produces exactly one assignment that satisfies $\psi_{i,k}$.

Let w be the body of a simple chain with support

$$q_i = q_{t_0} \xrightarrow{a_1} q_{t_1} \xrightarrow{a_2} \ldots \xrightarrow{a_\ell} q_{t_\ell} \xRightarrow{q_{t_0}} q_k \tag{I.4.2}$$

We prove that $w \models \psi_{i,k}$ for $\boldsymbol{e},\boldsymbol{A}_0,\boldsymbol{A}_1,\ldots,\boldsymbol{A}_N,\boldsymbol{B}_0,\ldots,\boldsymbol{B}_N,\boldsymbol{C}_0,\ldots,\boldsymbol{C}_N$ defined as follows. First-order variable $\boldsymbol{e}$ equals $|w|$, $\boldsymbol{B}_h$ is empty except for $\boldsymbol{B}_k = \{0\}$; $\boldsymbol{C}_h$ is empty except for $\boldsymbol{C}_k = \{\ell\}$; for every $0 \le \boldsymbol{x} \le \ell$, let $\boldsymbol{A}_h$ contain $\boldsymbol{x}$ iff $t_{\boldsymbol{x}} = h$ (i.e., $\boldsymbol{x} \in \boldsymbol{A}_{t_x}$), and this also implies $Q_{t_x}(\boldsymbol{x},\boldsymbol{x}+1)$. Then Start_i and End_k are satisfied trivially since $\text{Tree}(0,1,\ell,\ell+1)$ holds. We now prove that also $\varphi_{\delta_{push}}$, $\varphi_{\delta_{shift}}$, and $\varphi_{\delta_{pop}}$ are satisfied; we omit to consider all cases where the antecedents are false.

- φ_{push} is satisfied for $\boldsymbol{x} = 0$ and $\boldsymbol{y} = 1$ since we have $a_1(1)$, $\# \lessdot a_1$, $Q_i(0,1)$, $1 \in \boldsymbol{A}_{t_1}$, and $\delta_{\text{push}}(q_i,a_1) = q_{t_1}$.
- φ_{shift} is satisfied $\forall 1 \le \boldsymbol{x} < \ell$ and $\boldsymbol{y} = \boldsymbol{x}+1$ since we have $a_{\boldsymbol{y}}(\boldsymbol{y})$, $a_{\boldsymbol{x}} \doteq a_{\boldsymbol{y}}$, $Q_{t_x}(\boldsymbol{x},\boldsymbol{y})$, $\boldsymbol{y} \in \boldsymbol{A}_{t_y}$, and $\delta_{\text{shift}}(q_{t_x},a_y) = q_{t_y}$.
- φ_{pop} is satisfied for $\boldsymbol{x} = 0$ and $\boldsymbol{y} = |w|+1 = \ell+1$ since we have $\text{Tree}_{t_\ell,i}(0,1,\ell,\ell+1)$, $0 \in \boldsymbol{B}_k$, $\ell \in \boldsymbol{C}_k$, and $\delta_{\text{pop}}(q_{t_\ell},q_i) = q_k$.

Let now w be the body of a composed chain with support

$$q_i = q_{t_0} \overset{w_0}{\rightsquigarrow} q_{f_0} \xrightarrow{a_1} q_{t_1} \overset{w_1}{\rightsquigarrow} q_{f_1} \xrightarrow{a_2} \dots \xrightarrow{a_g} q_{t_g} \overset{w_g}{\rightsquigarrow} q_{f_g} \dots \xrightarrow{a_\ell} q_{t_\ell} \overset{w_\ell}{\rightsquigarrow} q_{f_\ell} \overset{q_{f_0}}{\Longrightarrow} q_k \tag{I.4.3}$$

We prove that $w \vDash \psi_{i,k}$ for a suitable assignment. By the inductive hypothesis, for every $g = 0, 1, \dots, \ell$ such that $w_g \neq \varepsilon$ we have $w_g \vDash \psi_{t_g, f_g}$. Let $\boldsymbol{A}_0{}^g, \dots, \boldsymbol{A}_N{}^g, \boldsymbol{B}_0{}^g, \dots, \boldsymbol{B}_N{}^g, \boldsymbol{C}_0{}^g, \dots, \boldsymbol{C}_N{}^g$ be (the naturally shifted versions of) an assignment that satisfies ψ_{t_g, f_g}. In particular this implies $s_g \in \boldsymbol{A}_{t_g}$, $\mathrm{Next}_{f_g}(s_g, s_{g+1})$, and $s_g \in \boldsymbol{A}_{t_g} \cup \boldsymbol{B}_{f_g}$, for each g such that $w_g \neq \varepsilon$. Then define $\boldsymbol{A}_h, \boldsymbol{B}_h, \boldsymbol{C}_h$ as follows. Let $\boldsymbol{A}_h$ include all $\boldsymbol{A}_h{}^g$, $\boldsymbol{B}_h$ include all $\boldsymbol{B}_h{}^g$, $\boldsymbol{C}_h$ include all $\boldsymbol{C}_h{}^g$. Also let $\boldsymbol{B}_k$ contain s_0, $\boldsymbol{C}_k$ contain s_ℓ, and $\boldsymbol{A}_{t_g}$ contain s_g whenever w_g is empty; in particular this implies $Q_{f_g}(s_g, s_{g+1})$ for every $0 \le g < \ell$. Finally, $\boldsymbol{e}$ is defined as the length of w.

Then we show that $\psi_{i,k}$ is satisfied by checking every subformula. Start_i and End_k are satisfied trivially since $\mathrm{Tree}(0, s_1, s_\ell, |w| + 1)$ holds. By the inductive hypothesis, all other axioms are satisfied within every w_g. Thus, we only have to prove that they are satisfied in positions s_g, for $0 \le g \le \ell$. We omit to consider all cases where the antecedents are false.

- φ_{push} is satisfied for $\boldsymbol{x} = 0$ and $\boldsymbol{y} = s_1$ since we have $a_1(s_1)$, $\# \lessdot a_1$ $Q_{f_0}(0, s_1)$, $s_1 \in \boldsymbol{A}_{t_1}$, and $\delta_{\mathrm{push}}(q_{f_0}, a_1) = q_{t_1}$.
- φ_{shift} is satisfied for all $\boldsymbol{x} = s_g$ and $\boldsymbol{y} = s_{g+1}$ with $1 \le g < \ell$ since we have $a_g(s_g)$, $a_{s_g} \doteq a_{s_{g+1}}$, $Q_{f_g}(s_g, s_{g+1})$, $s_g \in \boldsymbol{A}_{t_g}$, and $\delta_{\mathrm{shift}}(q_{f_g}, a_g) = q_{t_g}$.
- φ_{pop} is satisfied for $\boldsymbol{x} = 0$ and $\boldsymbol{y} = |w| + 1$ since we have $\mathrm{Tree}_{f_\ell, f_0}(0, s_1, s_\ell, |w| + 1)$, $0 \in \boldsymbol{B}_k$, $\ell \in \boldsymbol{C}_k$, and $\delta_{\mathrm{pop}}(q_{t_\ell}, q_i) = q_k$.

Hence $w \vDash \psi_{i,k}$ for every w with a suitable support, and this concludes the proof. ☐

LEMMA I.4.4. *Let w be the body of a chain ${}^{\#}[w]^{\#}$. If $w \vDash \psi_{i,k}$ then $q_i \overset{w}{\rightsquigarrow} q_k$ in $\mathcal{A}$.*

Proof. Let $\boldsymbol{e} = |w|, \boldsymbol{A}_0, \dots, \boldsymbol{A}_N, \boldsymbol{B}_0, \dots, \boldsymbol{B}_N, \boldsymbol{C}_0, \dots, \boldsymbol{C}_N$ be an assignment that satisfies $\psi_{i,k}$. In particular this implies $0 \in \boldsymbol{A}_i \wedge \mathrm{Next}_k(0, |w| + 1)$, and such i, k are unique by definition of Start_i and End_k. Then the following properties hold.

(i) For each $0 \le \boldsymbol{x} \le |w|$, there exists a unique index i such that $\mathrm{Succ}_i(\boldsymbol{x}, \boldsymbol{x} + 1)$ holds true. This can be proved by induction on $\boldsymbol{x}$ by applying the formulae for δ_{push} and δ_{shift}.

(ii) For each $\boldsymbol{x}, \boldsymbol{y}$ such that $\boldsymbol{x} \curvearrowright \boldsymbol{y}$, let $\boldsymbol{z}, \boldsymbol{v}$ such that $\mathrm{Tree}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{v}, \boldsymbol{y})$ holds, then there exists a unique pair of indices i, j such that $\mathrm{Tree}_{i,j}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{v}, \boldsymbol{y})$ holds, and there exists a unique index k such that $\mathrm{Next}_k(\boldsymbol{x}, \boldsymbol{y})$. This can be proved by induction on the depth of the chain between positions $\boldsymbol{x}$ and $\boldsymbol{y}$, by applying the formulae for δ_{pop} and property (i).
Moreover, if $\mathrm{Tree}_{i,j}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{v}, \boldsymbol{y})$ holds, then $\mathrm{Next}_k(\boldsymbol{x}, \boldsymbol{y})$ holds if and only if $\delta_{\mathrm{pop}}(q_i, q_j) = q_k$.

Hence, by properties (i) and (ii), for each $\boldsymbol{x}, \boldsymbol{y}$ such that $\boldsymbol{x} + 1 = \boldsymbol{y}$ or $\boldsymbol{x} \curvearrowright \boldsymbol{y}$, there exists a unique i such that $Q_i(\boldsymbol{x}, \boldsymbol{y})$ holds true.

Now, for every g let t_g be the index such that $g \in \boldsymbol{A}_{t_g}$. t_g is unique by property (i) and in particular $t_0 = i$.

We proceed by induction on the depth h of w. Let $h = 1$ and $w = a_1 a_2 \dots a_\ell$ be the body of a simple chain. In this case t_g is the unique index such that $\mathrm{Succ}_{t_g}(g, g + 1)$. Then, by φ_{push_bw} with $\boldsymbol{y} = 1$, we have $\delta(q_{t_0}, a_1) = q_{t_1}$; and by φ_{shift_bw} with $1 \le g < \ell$, we have $\delta_{\mathrm{shift}}(q_{t_g}, a_{g+1}) = q_{t_{g+1}}$. Moreover, since $\mathrm{Tree}_{t_\ell, t_0}(0, 1, \ell, \ell + 1) \wedge \mathrm{Next}_k(0, \ell + 1)$, we get $\delta(q_{t_\ell}, q_{t_0}) = q_k$ by property (ii). Hence we have built a support of the type (I.4.2).

Let now be $h > 1$ and $w = w_0 a_1 w_1 \dots a_\ell w_\ell$. For $0 \le g \le \ell$, since $s_g \curvearrowright s_{g+1} \vee s_{g+1} = s_g + 1$, by properties (i) and (ii) above there exists a unique index f_g such that $Q_{f_g}(s_g, s_{g+1})$ holds. Notice that $w_g = \varepsilon$ implies $f_g = t_g$, otherwise we have $w_g \vDash \psi_{t_g, f_g}$ and, by the inductive hypothesis, there exists a support $q_{t_g} \overset{s_g}{\rightsquigarrow} q_{f_g}$ in $\mathcal{A}$. Thus, for every $0 \le g < \ell$, by applying φ_{push_bw} with $\boldsymbol{y} = s_{g+1}$ we get $\delta(q_{f_g}, a_{g+1}) = q_{t_{g+1}}$. Moreover, since $\mathrm{Tree}_{f_\ell, f_0}(0, s_1, s_\ell, |w| + 1) \wedge \mathrm{Next}_k(0, |w| + 1)$, by property (ii) above we get $\delta(q_{t_\ell}, q_i) = q_k$. Hence we have built a support of type (I.4.3) and this concludes the proof. ∎

Part II: Operator Precedence ω-Languages

Languages of infinite-length strings, called ω-languages, have been introduced to model nonterminating processes; thus they are becoming more and more relevant nowadays when most applications are "ever-running", often in a distributed environment. Again, the foundations of the theory of ω-languages are due to the pioneering work by Büchi [12] and others [32, 30, 37, 9]. Büchi, in particular, investigated their main algebraic properties in the context of finite state machines, pointing out commonalities and differences w.r.t. the finite length counterpart [12, 40]. His work has then been extended to larger classes of languages; among them, again noticeably, the class of VPLs; and again, in this part we face the same job for the class of OPLs. OPLs, in fact, are not only useful to model programs, which are typically of finite length, but are also well-suited to formalize possibly never-ending sequences of events: for instance, the previous Example 5 can be naturally extended to model the behavior of a database that is never put off.

This part is organized as follows. In Section II.1 we first extend to ω-languages a few basic definitions given in Part 1 for finite-length languages and generalize to OPAs the classical accepting criteria for ω-languages, then we show by means of an example the usefulness of ωOPAs to model and analyze various system types; Section II.2 shows the relations between the various classes of ωOPLs classified according to the acceptance criteria defined in the previous section; Section II.3 shows which closure properties are preserved and which ones are lost when moving from finite length languages to the various classes of ω-languages; finally, Section II.4 extends to ω-languages the characterization in terms of MSO logic.

II.1. Basic definitions of ω-languages. Preliminarily we introduce some further properties related to chains that are necessary when chains occur within infinite words.

Definition II.1.1.
Let (Σ, M) be a precedence alphabet and w a word on Σ compatible with M:

- *A chain in w is* maximal *if it does not belong to a larger composed chain. In a finite word w preceded and ended by #, only the outmost chain ${}^{\#}[w]^{\#}$ is maximal.*
- *An* open chain *is a sequence of symbols $b_0 \lessdot a_1 \doteq a_2 \doteq \ldots \doteq a_n$, for $n \geq 1$.*
- *A letter $a \in \Sigma$ in a word $\#w$ with $w \in \Sigma^*$ compatible with M, is* pending *if it does not belong to the body of a chain. In a word w preceded and ended by #, there are no pending letters.*

Furthermore, we generalize in a natural way to the infinite case the notion of string compatible with an OPM: given a precedence alphabet (Σ, M), we say that an ω-word w is *compatible* with the OPM M if every prefix of w is compatible with M. We denote by $L_M \subseteq \Sigma^\omega$ the ω-language comprising all infinite words $x \in \Sigma^\omega$ compatible with M.

Next, we adopt for OPAs operating on infinite strings the same acceptance criteria that have been adopted in the literature for regular and other classes of languages.

Definition II.1.2 (Büchi operator precedence ω-automaton). *A* nondeterministic Büchi operator precedence ω-automaton *(ωOPBA) is given by a tuple $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$, where Σ, Q, I, F, δ are defined as for OPAs; the operator precedence matrix M is restricted to be a $|\Sigma \cup \{\#\}| \times |\Sigma|$ array, since ω-words are not terminated by the delimiter #.*

Configurations and *(infinite) runs* are defined as for operator precedence automata on finite-length words. Then, let "$\exists^\omega i$" be a shorthand for "there exist infinitely many i" and let ρ be a run of the automaton on a given word $x \in \Sigma^\omega$. Define $Inf(\rho) = \{q \in Q \mid \exists^\omega i \, \langle \beta_i, \, q_i, \, x_i \rangle \in \rho, \; q_i = q\}$ as the set of states that occur infinitely often in configurations in ρ. A run ρ of an ωOPBA on an infinite word $x \in \Sigma^\omega$ is *successful* iff there exists a state $q_f \in F$ such that $q_f \in Inf(\rho)$. $\mathcal{A}$ *accepts* $x \in \Sigma^\omega$ iff there is a successful run of $\mathcal{A}$ on x. The ω-language *recognized* by $\mathcal{A}$ is $L(\mathcal{A}) = \{x \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } x\}$.

The classical notion of acceptance for Muller automata can be likewise defined for OPAs.

Definition II.1.3 (Muller operator precedence ω-automaton). *A* nondeterministic Muller operator precedence automaton *(ωOPMA) is a tuple $\langle \Sigma, M, Q, I, \mathcal{T}, \delta \rangle$ where Σ, M, Q, I, δ are defined as for ωOPBAs and $\mathcal{T}$ is a collection of subsets of Q, $\mathcal{T} \subseteq \wp(Q)$, called the* table *of the automaton.*
A run ρ of an ωOPMA on an infinite word $x \in \Sigma^\omega$ is successful *iff $Inf(\rho) \in \mathcal{T}$, i.e. the set of states occurring infinitely often in the configurations of ρ is a set in the table $\mathcal{T}$.*

Definition II.1.4. *A* nondeterministic Büchi operator precedence automaton accepting with empty stack *(ωOPBEA) is a variant of ωOPBA where a run ρ is successful iff there exists a state $q_f \in \mathcal{F}$ such that configurations with stack $\perp$ and state q_f occur infinitely often in ρ.*

Thus, a run of an ωOPBEA is successful iff the automaton traverses final states with an empty stack infinitely often. We will use the following simple normal form for ωOPBEA.

Definition II.1.5. *An ωOPBEA is in* normal form *if the set of states is partitioned into states that are always visited with empty stack and states that are never visited with empty stack.*

For all above classes of automata, say, ω-XXX, their deterministic counterpart ω-DXXX is defined as usual.

Example 8 (Managing interrupts). *Consider a software system that is designed to work forever and must serve requests issued by different users but subject to interrupts. Precisely, assume that the system manages two types of "normal operations" a and b, and two types of interrupts, with different levels of priority.*

We model its behavior by introducing an alphabet with two pairs of calls and returns, $call_a$, $call_b$, ret_a, ret_b, for operations a and b and symbols int_1, $serve_1$ denoting the lower level interrupt and its serving, respectively, and int_2, $serve_2$ denoting the higher level ones. Not only both interrupts discard possible pending calls not already matched by corresponding returns, but also the serving of a higher priority interrupt erases possible pending requests for lower priority ones, but not those that occurred before the higher priority interrupt just served: thus, a sequence such as $int_1 int_2$ int_1 int_1 $serve_2$ should produce popping the second and third int_1 without matching them, to match immediately int_2 with $serve_2$, but would leave the first occurrence of int_1 still pending; the next $serve_1$, if any, would match it, whereas possible further $serve_1$ would remain unmatched. Furthermore neither calls to, nor returns from, operations a and b can occur while any interrupt is pending.

Figure II.1.1 shows an OPM that assigns to sequences on the above alphabet a structure compatible with the described priorities. Then, a suitable ω-automaton can specify further constraints on such sequences; for instance the ωOPBA of Figure II.1.2 restricts the set of ω-sequences compatible with the matrix by imposing that all int_2 are eventually served by a corresponding $serve_2$; furthermore lower priority interrupts are not just discarded when a higher priority one is pending but they are simply disabled, i.e. they are not accepted as a correct system behavior.

For instance, the ω-word $call_a$ int_1 int_2 $int_1 \ldots$ is not accepted by the ωOPBAbecause int_1 is not accepted from state q_2 reached after reading int_2; similarly, $call_a$ int_1 int_2 $serve_2$ $call_a$ is rejected since, after serving int_2 the automaton would be back in state q_1 with int_1 pending (the prefix $call_a$ int_1 int_2 $serve_2$ is compatible with the OPM and int_1 is pending therein) but no $call_a$ is admitted in q_1 since there is no precedence relation between int_1 and $call_a$. On the contrary the ω-word $call_a$ $int_1(int_2$ $serve_2$ $serve_1$ $call_a$ $call_a$ $ret_a)^\omega$ is accepted: in fact the automaton reaches q_1 after reading $call_a$ (and popping it) followed by int_1; then, after receiving and serving the higher priority interrupt, it would serve the pending instance of int_1

returning to q_0; from this point on it would enter an infinite loop during which it would process the input string $(call_a\ call_a\ ret_a\ int_2\ serve_2\ serve_1)^\omega$ traversing the states $q_0 \xrightarrow{call_a} q_0 \xrightarrow{call_a} q_0 \overset{ret_a}{\dashrightarrow} q_0 \overset{q_0}{\Longrightarrow} q_0 \xrightarrow{int_2} q_2 \overset{serve_2}{\dashrightarrow} q_2 \overset{q_2}{\Longrightarrow} q_0 \overset{q_0}{\Longrightarrow} q_0 \xrightarrow{serve_1} q_1 \overset{q_0}{\Longrightarrow} q_0$ leaving the first $call_a$ and $serve_1$ unmatched. Notice that all finite prefixes $call_a\ int_1(int_2\ serve_2\ serve_1\ call_a\ call_a\ ret_a)^n$ $int_2\ serve_2\ serve_1\ call_a\ call_a$, with $n > 0$, end with the open chain $call_a \lessdot call_a$. Finally, observe that the automaton would accept some strings beginning with $serve_1$ which might appear somewhat counterintuitive but is consistent with the general philosophy of admitting unmatched elements; it would be easy, however, to forbid such a string beginning.

We call $L_{interrupt}$ the language recognized by this ωOPBA.

	$call_a$	ret_a	$call_b$	ret_b	int_1	int_2	$serve_1$	$serve_2$
$call_a$	$\lessdot$	$\doteq$	$\lessdot$		$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
ret_a	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
$call_b$	$\lessdot$		$\lessdot$	$\doteq$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
ret_b	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
int_1					$\lessdot$	$\lessdot$	$\doteq$	$\gtrdot$
int_2					$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$
$serve_1$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
$serve_2$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
#	$\lessdot$		$\lessdot$		$\lessdot$	$\lessdot$	$\lessdot$	

Figure II.1.1

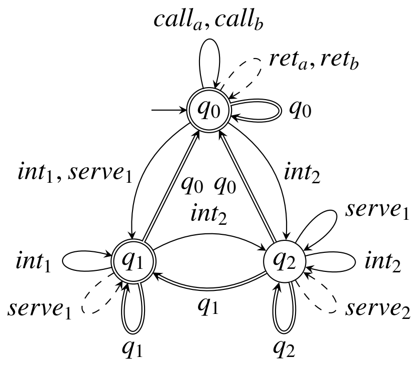


Figure II.1.2: ωOPBA recognizing the language of Example 8.

A more sophisticated policy that could easily be formalized by means of a suitable ω-automaton is a "weak fairness requirement" imposing that, after a first $call_a$ not matched by ret_a but interrupted by a int_1 or int_2, a second $call_a$ cannot be interrupted by a new lower priority interrupt int_1 (but can still be interrupted at any time by higher priority ones).

This example too retains some typical features of VPLs, namely the possibility of having unmatched calls or returns but, again, it strongly generalizes them in that unmatched elements can occur in various places of the whole string, e.g., due to the occurrence of interrupts or other exceptional events.

Further examples illustrating the modeling capabilities of OPLs both on finite and infinite strings are reported in [35].

II.2. Relationships among classes of ωOPLs. Here we study the relationships among languages recognized by the different classes of operator precedence ω-automata and visibly pushdown ω-automata (with Büchi acceptance criterion), denoted as ωBVPA. Such relations are summarized by the diagram in Figure II.2.1, where solid lines denote strict inclusion and dashed lines link classes that are not comparable.

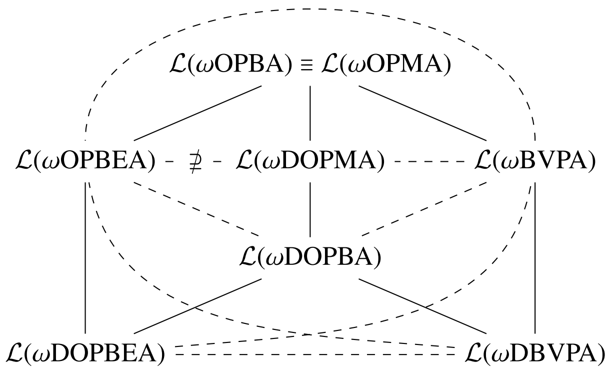

$\mathcal{L}(\omega\text{OPBA}) \equiv \mathcal{L}(\omega\text{OPMA})$
$\mathcal{L}(\omega\text{OPBEA})$
$\nsupseteq$
$\mathcal{L}(\omega\text{DOPMA})$
$\mathcal{L}(\omega\text{BVPA})$
$\mathcal{L}(\omega\text{DOPBA})$
$\mathcal{L}(\omega\text{DOPBEA})$
$\mathcal{L}(\omega\text{DBVPA})$

Figure II.2.1: Containment relations for ωOPLs. Solid lines denote strict inclusion of the lower class in the upper one; dashed lines link classes which are not comparable. It is still open whether $\mathcal{L}(\omega\text{OPBEA}) \subseteq \mathcal{L}(\omega\text{DOPMA})$ or not.

In the following, we first present the proofs of the weak containment relations holding among the various classes: most of them follow trivially from the definitions, except for the equality between $\mathcal{L}(\omega\text{OPBA})$ and $\mathcal{L}(\omega\text{OPMA})$. Then we will prove strict inclusions and incomparability relations by means of a suitable set of examples that separate the various classes.

II.2.1. Weak inclusion results.

Theorem II.2.1. *The following inclusion relations hold:*

$$\mathcal{L}(\omega BVPA) \subseteq \mathcal{L}(\omega OPBA), \qquad \mathcal{L}(\omega DBVPA) \subseteq \mathcal{L}(\omega DOPBA).$$

Proof. Let $\mathcal{A} = \langle Q_A, I_A, \Gamma_A, \delta_A, F_A \rangle$ be an ωBVPA[9] over a partitioned alphabet $\Sigma = (\Sigma_c, \Sigma_r, \Sigma_i)$. An ωOPBA $\mathcal{B}$ that recognizes the same language as $\mathcal{A}$ is defined in a straightforward way as follows: $\mathcal{B} = \langle \Sigma, M, Q_B, I_B, \delta_B, F_B \rangle$ where

- $Q_B = Q_A \times \Gamma_A$,
- $I_B = I_A \times \{\top\}$,
- $F_B = F_A \times \Gamma_A$,

[9] Among the many equivalent definitions for VPAs we adopt here the original one in [3].

- M is the precedence matrix induced by the partition on Σ:

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

- the transition function $\delta : Q_B \times (\Sigma \cup Q_B) \to \wp(Q_B)$ is defined as follows, where $q_1, q_2 \in Q_A$.
  The push transition $\delta_{\text{Bpush}} : Q_B \times \Sigma \to \wp(Q_B)$ is defined by:
  - for $a \in \Sigma_c$, $\delta_{\text{Bpush}}(\langle q_1, \gamma_1\rangle, a) = \{\langle q_2, \gamma_2\rangle \mid (q_1, a, q_2, \gamma_2) \in \delta_A\}$
  - for $a \in \Sigma_i$, $\delta_{\text{Bpush}}(\langle q_1, \gamma\rangle, a) = \{\langle q_2, \gamma\rangle \mid (q_1, a, q_2) \in \delta_A\}$
  - for $a \in \Sigma_r$, $\delta_{\text{Bpush}}(\langle q_1, \top\rangle, a) = \{\langle q_2, \top\rangle \mid (q_1, a, \top, q_2) \in \delta_A\}$.

  The shift transition $\delta_{\text{Bshift}} : Q_B \times \Sigma \to \wp(Q_B)$ is defined by
  - for $a \in \Sigma_r$, $\delta_{\text{Bshift}}(\langle q_1, \gamma\rangle, a) = \{\langle q_2, \gamma\rangle \mid (q_1, a, \gamma, q_2) \in \delta_A\}$, i.e., the ωOPBA simulates the pop move of the ωBVPA by setting, as state q_2, a state reached by the ωBVPA while reading the return symbol a.

  The pop transition $\delta_{\text{pop}} : Q_B \times Q_B \to \wp(Q_B)$ is defined as follows:
  - $\delta_{\text{Bpop}}(\langle q_1, \gamma_1\rangle, \langle q_2, \gamma_2\rangle) = \{\langle q_1, \gamma_2\rangle\}$, i.e., restores the state reached by the ωBVPA after its pop move.

If the original ωBVPA is deterministic, so is the ωOPBA obtained with the above construction, and this yields the second relation. ◻

Proposition II.2.2. *The following inclusion relations hold:*

$$\mathcal{L}(\omega OPBEA) \subseteq \mathcal{L}(\omega OPBA),$$

$$\mathcal{L}(\omega DOPBEA) \subseteq \mathcal{L}(\omega DOPBA) \subseteq \mathcal{L}(\omega DOPMA) \subseteq \mathcal{L}(\omega OPMA).$$

Proof. The first inclusion follows trivially from the definition of ωOPBA and ωOPBEA in normal form: given an ωOPBEA whose set of states is partitioned into states that are always visited with empty stack and states that are never visited with empty stack, we can define an equivalent ωOPBA that has as final states the final states of the ωOPBEA that are always visited with empty stack.

The inclusion follows similarly for the deterministic counterparts of these classes of ωOPAs, since this ωOPBA is deterministic if the ωOPBEA is deterministic.

About the relations involving Muller automata, $\mathcal{L}(\omega\text{DOPBA}) \subseteq \mathcal{L}(\omega\text{DOPMA})$ derives form the fact that any ωDOPBA $\mathcal{B} = \langle \Sigma, M, Q, q_0, F, \delta\rangle$ is equivalent to an ωDOPMA $\mathcal{A} = \langle \Sigma, M, Q, q_0, \mathcal{T}, \delta\rangle$ whose acceptance component $\mathcal{T}$ consists of all subsets of Q including some final state of $\mathcal{B}$, namely $\mathcal{T} = \{P \subseteq Q \mid P \cap F \neq \varnothing\}$; the last relation is obvious. ◻

In the case of classical finite-state automata on infinite words, nondeterministic Büchi automata and nondeterministic Muller automata are equivalent and define the class of ω-regular languages. Traditionally, Muller automata have been introduced to provide an adequate acceptance mode for deterministic automata on ω-words. In fact, deterministic Büchi automata cannot recognize all ω-regular languages, whereas deterministic Muller automata are equivalent to nondeterministic Büchi ones [40].

For VPAs on infinite words, instead, the paper [4] showed that the classical determinization algorithm of Büchi automata into deterministic Muller automata is no longer valid, and deterministic Muller ωVPAs are strictly less powerful than nondeterministic Büchi ωVPAs. A similar relationship holds for ωOPAs too.

Theorem II.2.3. *$\mathcal{L}(\omega OPBA) = \mathcal{L}(\omega OPMA)$.*

Proof. Each ωOPBA is equivalent to an ωOPMA having the same underlying OPA and acceptance component $\mathcal{T}$ consisting of all subsets of states including some final state of B (as for their deterministic counterpart, see proof of Proposition II.2.2).

Conversely, any ω-language recognized by an ωOPMA $\mathcal{A} = \langle \Sigma, M, Q, I, \mathcal{T}, \delta \rangle$ can be recognized by an ωOPBA $\mathcal{B}$ with the same precedence matrix and with $O(s2^s)$ states, where s is the number of states of $\mathcal{A}$. We can assume that $\mathcal{T}$ is a singleton. Indeed, $L(\mathcal{A})$ can be expressed as

$$L(\mathcal{A}) = \bigcup_{T \in \mathcal{T}} L(\mathcal{A}_T), \text{ where } \mathcal{A}_T = \langle \Sigma, M, Q, I, \{T\}, \delta \rangle.$$

Since $\mathcal{L}(\omega$OPBA) is closed under union (a property that will be proved later, with Theorem II.3.6), if each language $L(\mathcal{A}_T)$ is accepted by an ωOPBA, then $L(\mathcal{A})$ too is accepted by an ωOPBA.

Thus, let $\mathcal{T}$ be the singleton $\{T\}$. Let us build an ωOPBA $\mathcal{B} = \langle \Sigma, M, \tilde{Q}, I, F, \tilde{\delta} \rangle$ that accepts the same language as $\mathcal{A}$ as follows. $\tilde{Q}$ includes elements of two types: states of $\mathcal{A}$, and states (q, R) where $q \in Q$ and $R \subseteq Q$ is a set (that we informally call "box"), which will be used to test whether the run of $\mathcal{A}$ is successful.

Intuitively, the automaton $\mathcal{B}$ simulates $\mathcal{A}$, reading the input string x, along a sequence of states q, and then guesses nondeterministically the point after which a successful run ρ of $\mathcal{A}$ on x stops visiting the states that occur only finitely often in the run, and ρ begins to visit all and only the states in the set T. After this point $\mathcal{B}$ switches to the states of the form (q, R) and collects in R the states visited by $\mathcal{A}$ during the run, "emptying the box" as soon as it contains exactly the set T. Every time it empties the box, $\mathcal{B}$ resumes collecting the states that $\mathcal{A}$ will visit from that point onwards. If the final states of $\mathcal{B}$ are defined as those ones when it collects exactly the set T, then $\mathcal{B}$ will visit infinitely often these final states iff $\mathcal{A}$ visits all and only the states in T infinitely often.

More formally, $\mathcal{B}$ is defined by:

- $\tilde{Q} = Q \cup (Q \times \wp(Q))$,
- $F = \{(q, T) \mid q \in T\}$,
- $\tilde{\delta} : \tilde{Q} \times (\Sigma \cup \tilde{Q}) \to \wp(\tilde{Q})$, where the push function is defined by:
  - $\tilde{\delta}_{\text{push}}(q, a) = \delta_{\text{push}}(q, a) \cup \{\langle p, \{p\} \rangle \mid p \in \delta_{\text{push}}(q, a)\} \quad \forall q \in Q, a \in \Sigma$
  - $\tilde{\delta}_{\text{push}}(\langle q, R \rangle, a) = \begin{cases} \{\langle p, R \cup \{p\} \rangle \mid p \in \delta_{\text{push}}(q, a)\} & \text{if } R \neq T \\ \{\langle p, \{p\} \rangle \mid p \in \delta_{\text{push}}(q, a)\} & \text{if } R = T \end{cases}$

    $\forall q \in Q, R \subseteq Q, a \in \Sigma$.

  The shift function is defined analogously.

  The pop function $\tilde{\delta}_{\text{pop}} : \tilde{Q} \times \tilde{Q} \to \wp(\tilde{Q})$ is defined by:
  - $\tilde{\delta}_{\text{pop}}(q_1, q_2) = \delta_{\text{pop}}(q_1, q_2) \cup \{\langle p, \{p\} \rangle \mid p \in \delta_{\text{pop}}(q_1, q_2)\}$,
    $\forall q_1, q_2 \in Q$
  - $\tilde{\delta}_{\text{pop}}(\langle q_1, R \rangle, q_2) = \begin{cases} \{\langle p, R \cup \{p\} \rangle \mid p \in \delta_{\text{pop}}(q_1, q_2)\} & \text{if } R \neq T \\ \{\langle p, \{p\} \rangle \mid p \in \delta_{\text{pop}}(q_1, q_2)\} & \text{if } R = T \end{cases}$
  - $\tilde{\delta}_{\text{pop}}(\langle q_1, R_1 \rangle, \langle q_2, R_2 \rangle) = \begin{cases} \{\langle p, R_1 \cup \{p\} \rangle \mid p \in \delta_{\text{pop}}(q_1, q_2)\} & \text{if } R_1 \neq T \\ \{\langle p, \{p\} \rangle \mid p \in \delta_{\text{pop}}(q_1, q_2)\} & \text{if } R_1 = T \end{cases}$

    $\forall q_1, q_2 \in Q, R, R_1, R_2 \subseteq Q$.

First, we show that $L(\mathcal{A}) \subseteq L(\mathcal{B})$. Let $x \in L(\mathcal{A})$, and let ρ be a successful run on x. There exists a finite prefix $v \in \Sigma^*$ of $x = vu_1u_2\ldots$ such that the infinite path followed by $\mathcal{A}$ after reading v (i.e., on the infinite word $u_1u_2\ldots$) visits all and only states in T infinitely often. Thus, the run ρ can be written as:

$$\rho = \langle \alpha_0 = \bot,\ q_0,\ x = vu_1u_2\ldots\rangle \overset{*}{\vdash} \langle \alpha_{|v|},\ q_{|v|},\ u_1u_2\ldots\rangle \overset{+}{\vdash} \ldots \overset{+}{\vdash} \langle \alpha_i,\ q_i,\ u_i\ldots\rangle \overset{+}{\vdash} \ldots$$

where $\{q_i \mid i > |v|\} = T$ and $q_0 \in I$. Then, there is a successful run $\tilde{\rho}$ of $\mathcal{B}$ on the same word, which follows singleton states of $\mathcal{A}$ while it reads v

$$\tilde{\rho} = \langle \beta_0 = \alpha_0 = \bot,\ q_0,\ x = vu_1u_2\ldots\rangle \overset{*}{\vdash} \langle \beta_{|v|} = \alpha_{|v|},\ q_{|v|},\ u_1u_2\ldots\rangle$$

and then switches to states augmented with a box: $\langle \beta_{|v|} = \alpha_{|v|},\ q_{|v|},\ u_1u_2\ldots\rangle \vdash \langle \beta_{|v|+1} = \alpha_{|v|+1},\ \langle p, \{p\}\rangle,\ \tilde{u}_1u_2\ldots\rangle$, where $\langle \alpha_{|v|},\ q_{|v|},\ u_1u_2\ldots\rangle \vdash \langle \alpha_{|v|+1},\ p,\ \tilde{u}_1u_2\ldots\rangle$ and $u_1 = a\tilde{u}_1$.

Since after this point $\mathcal{A}$ visits each state in T and only these states infinitely often, $\mathcal{B}$ will reach infinitely often final states $(q, T) \in F$, emptying infinitely often its box as soon as it gets full, and resuming the collection of states therein with the subsequent state in the run.

Conversely, $L(\mathcal{B}) \subseteq L(\mathcal{A})$.
Let $x \in \Sigma^\omega$ be an infinite word in $L(\mathcal{B})$. Define the projection $\pi : Q \cup (Q \times \wp(Q)) \to Q$ as $\pi(q) = q$ and $\pi(\langle q, R\rangle) = q, \forall q \in Q, R \subseteq Q$. Given a run $\tilde{\rho}$ of the automaton $\mathcal{B}$, let $\pi(\tilde{\rho})$ be the natural extension of π on a run.

By construction, if $\tilde{\rho}$ is a run of $\mathcal{B}$ on an ω-word, then $\pi(\tilde{\rho}) = \rho$ is a run for $\mathcal{A}$ on the same word.
Now, let $\tilde{\rho}$ be a successful run for $\mathcal{B}$ on x; $\rho = \pi(\tilde{\rho})$ is a run for $\mathcal{A}$ on x. Since only the states augmented with a box are final states, then after a sequence (possibly empty) of singleton states initially traversed by $\mathcal{B}$, the automaton will definitively visit only states of the form (q, R) (in fact, no singleton state is reachable again from these states).

By induction on the number of final states reached by $\mathcal{B}$ along its run, it can be proved that, for each pair of final states consecutively reached by $\mathcal{B}$, say (q_{F_i}, R_i) and $(q_{F_{i+1}}, R_{i+1})$, the portion of the run visited between them, say $\tilde{\rho}_i$, is such that the set of states reached along $\pi(\tilde{\rho}_i)$ equals exactly T. Finally, since final states in $\tilde{\rho}$ are visited infinitely often, the run $\pi(\tilde{\rho})$ is successful for $\mathcal{A}$. ☐

II.2.2. Strict inclusion and incomparability results. To prove the strict inclusion and incomparability relations summarized in Figure II.2.1, we introduce some simple examples of ω-languages, whose membership properties are summarized in Table II.2.1.

1. For $\Sigma = \{a, b\}$, $L_{a\infty} = \{x \in \Sigma^\omega : x$ contains an infinite number of occurrences of letter $a\}$ is recognized by the ωDOPBEA depicted in Figure II.2.2.

	a	b
a	$\gtrdot$	$\gtrdot$
b	$\gtrdot$	$\gtrdot$
$\#$	$\lessdot$	$\lessdot$

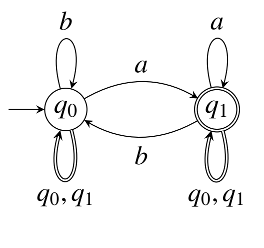


Figure II.2.2: ωDOPBEA, with its OPM, for $L_{a\infty}$.

	$\mathcal{L}(\omega\text{DOPBEA})$	$\mathcal{L}(\omega\text{OPBEA})$	$\mathcal{L}(\omega\text{DOPBA})$	$\mathcal{L}(\omega\text{DOPMA})$	$\mathcal{L}(\omega\text{OPBA})$	$\mathcal{L}(\omega\text{DBVPA})$	$\mathcal{L}(\omega\text{BVPA})$
$L_{a-\text{finite}}$	$\notin$	$\in$	$\notin$	$\in$		$\notin$	$\in$
$L_{a\infty}$	$\in$		$\in$				
$L_{\omega\text{Dyck-pr}(c,r)}$	$\in$	$\in$		$\in$			
L_{repbsd}	$\notin$	$\notin$		$\notin$	$\in$		$\in$
$L_{a2abseq}$	$\notin$	$\notin$	$\in$	$\in$	$\in$		
L^{ω}_{abseq}		$\in$					
$L_{\text{interrupt}}$	$\in$	$\in$	$\in$	$\in$	$\in$	$\notin$	$\notin$

Table II.2.1: Membership properties of some ω-languages, proved in Section II.2.2 or consequences of inclusion relations proved in previous sections. The table displays only the relations needed to prove the results in this and the following section.

2. $L_{a-\text{finite}} = \{x \in \Sigma^\omega : x$ contains a finite number of occurrences of $a\}$, i.e., the complement of $L_{a\infty}$, is clearly recognized by an ωDOPMA and by an ωOPBEA, but cannot be recognized by any ωDOPBA. The proof of this latter fact resembles the classical proof (see [40]) that deterministic Büchi automata are strictly weaker than nondeterministic Büchi ones.
3. For $\Sigma = \{c, r\}$, let $L_{\omega\text{Dyck-pr}(c,r)}$ be the language of ω-words composed by an infinite sequence of finite-length words belonging to the Dyck language with pair c, r with possibly *pending returns*, i.e. letters r not matched by any previous corresponding letter c. $L_{\omega\text{Dyck-pr}(c,r)}$ is recognized by the ωDOPMA and the ωDOPBEA whose state graph is depicted in Figure II.2.3 and with acceptance component defined, respectively, by the table $\mathcal{T} = \{\{q_0\}, \{q_0, q_1\}\}$ and the set of final states $F = \{q_0\}$.

	c	r
c	$\lessdot$	$\doteq$
r	$\gtrdot$	$\gtrdot$
#	$\lessdot$	$\lessdot$

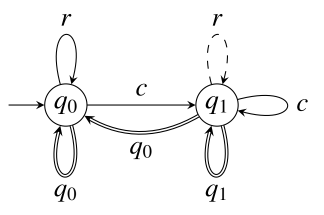


Figure II.2.3: ωDOPMA and ωDOPBEA recognizing $L_{\omega\text{Dyck-pr}(c,r)}$.

4. For $\Sigma = \{c, r\}$, let L_{repbsd} be the language (studied in [4]) consisting of ω-words x on Σ such that x has only finitely many *pending calls*, i.e. occurrences of letter c not matched by any subsequent corresponding letter r (*repbsd* stands for repeatedly bounded stack depth). L_{repbsd} is accepted by an ωOPBA, but cannot be accepted by any ωOPBEA.
Intuitively, an ωOPBEA accepts a word iff it reaches infinitely often a final configuration with empty stack reading the input string; however, the automaton is never able to remove all the input symbols piled on the stack since it cannot pop the pend-

ing calls interspersed among the correctly nested letters c, otherwise it would either introduce conflicts in the OPM or it would not be able to verify that they are in finite number.
More formally, assume by contradiction that there is an ωOPBEA $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ recognizing L_{repbsd}. M must satisfy the following constraints: since

- $r^\omega \in L_{repbsd}$, then $M_{\#r} = \{\lessdot\}$ and $M_{rr} = \{\gtrdot\}$,
- $cr^\omega \in L_{repbsd}$, then $M_{\#c} = \{\lessdot\}$, and either $M_{cr} = \{\gtrdot\}$ or $M_{cr} = \{\doteq\}$,
- $r(cr)^\omega \in L_{repbsd}$, thus if $c \doteq r$, $M_{rc} = \{\gtrdot\}$,
- $c(cr)^\omega \in L_{repbsd}$, thus if $c \doteq r$, $M_{cc} \neq \{\lessdot\}$

Hence, M must comply with one of the matrices M_1 or M_2 shown in Figure II.2.4.

M_1	c	r
c	$\gtrdot$	$\doteq$
r	$\gtrdot$	$\gtrdot$
$\#$	$\lessdot$	$\lessdot$

M_2	c	r
c	$\circ$	$\gtrdot$
r	$\circ$	$\gtrdot$
$\#$	$\lessdot$	$\lessdot$

Figure II.2.4: Matrices for L_{repbsd}, where $\circ \in \{\lessdot, \gtrdot, \doteq\}$.

Let $w = crc^2r^2c^3r^3 \dots c^nr^n \dots \in L_{repbsd}$ and let ρ be an accepting run of $\mathcal{A}$ on w starting from a state $q_0 \in I$. The proof that $L_{repbsd} \notin \mathcal{L}(\omega\text{OPBEA})$ is based on the two straightforward remarks:

- If, along a run, an ωOPA (or also an OPA) reaches a state with an empty stack, the subsequent suffix of the run does not depend on the transitions performed until that state.
- Since Q is finite, there exist $p, q \in Q$, and an infinite set of indexes $E \subseteq \mathbb{N} \setminus \{0,1,2\}$ such that, for each $i \in E$, ρ has a prefix: $q_0 \overset{v_i}{\rightsquigarrow} p \overset{w_i}{\rightsquigarrow} q$, where $v_i = c^1r^1 \dots c^{i-2}r^{i-2}c^{i-1}r^{i-2}$ and $w_i = rc^ir$ and, given the precedence relations in M_1 and M_2, both p and q are reached with an empty stack, just before performing a push move while reading the letter r in w that follows, respectively, v_i and w_i.
  For each $i \in E$, let ρ_i be the finite factor of ρ given by $p \overset{w_i}{\rightsquigarrow} q$.

Let $J \subseteq E$ be the set of indexes in E such that, $\forall i \in J$, ρ_i visits a final state with empty stack. We can build a run ρ', which differs from ρ in that

- for every $i \in E \setminus J$, the factor ρ_i is replaced by a ρ_j for some $j \in E$, with $j > i$,
- for every $i \in J$, the factor ρ_i is replaced by a ρ_j with $i < j \in J$ if $|J| = \infty$, or $i < j \in E$ if $|J| < \infty$.

ρ' is an accepting run in $\mathcal{A}$, along which the automaton reads a word with infinitely many pending calls, which does not belong to L_{repbsd}, and this is a contradiction.
Furthermore, L_{repbsd} is not recognizable by any ωDOPMA. The proof of this fact resembles the analogous proof in [4]; indeed, that proof is essentially based on topological properties of the state-graph of the automata and it is general enough to adapt to both ωVPAs and ωOPAs.

5. For $\Sigma = \{a, b\}$, let $L_{abseq} = \{a^kb^k \mid k \geq 1\}$ and $L_{a2abseq} = \{x \in \Sigma^\omega \mid x = a^2L_{abseq}^\omega\}$. Language $L_{a2abseq}$ is recognized by an ωDOPBA, but it is not recognized by any ωOPBEA (nor a fortiori by any ωDOPBEA).
Indeed, words in L_{abseq} can be recognized only with the OPM M depicted in Figure II.2.5: any other OPM will prevent verifying that the number of as equals that of bs in subwords belonging to L_{abseq}. Since $a \lessdot a$, an ωOPBEA piles up on the stack the first sequence a^2 of a word and cannot remove it afterwards; hence it cannot empty the stack infinitely often to accept a string in $L_{a2abseq}$. There is, however, an

ωDOPBA (and thus an ωDOPMA) that recognizes such a language: it is shown in Figure II.2.5. Notice also that L^{ω}_{abseq} can be recognized by an ωOPBEA, with OPM M and with state graph depicted in Figure II.2.5 but with state q_2 instead of q_0 as initial state.

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

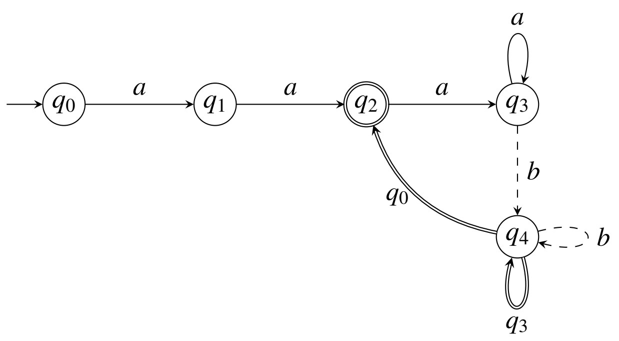


Figure II.2.5: An ωDOPBA recognizing language $L_{a2abseq}$.

II.3. Closure properties. Table II.3.1 displays the closure properties of the various families of ω-languages. In order to prove them, we first introduce some preliminary constructions in Section II.3.1. Then in Section II.3.2 we present the proofs for $\mathcal{L}(\omega\text{OPBA})$; in particular closure under complement and concatenation are the cases that require novel investigation techniques w.r.t. previous literature. In Section II.3.3 we prove the closure properties for other classes of ωOPA.

	$\mathcal{L}(\omega\text{DOPBEA})$	$\mathcal{L}(\omega\text{OPBEA})$	$\mathcal{L}(\omega\text{DOPBA})$	$\mathcal{L}(\omega\text{DOPMA})$	$\mathcal{L}(\omega\text{OPBA})$	$\mathcal{L}(\omega\text{BVPA})$
Intersection	Yes	Yes	Yes	Yes	Yes	Yes
Union	Yes	Yes	Yes	Yes	Yes	Yes
Complement	No	No	No	Yes	Yes	Yes
$L_1 \cdot L_2$	No	No	No	No	Yes	Yes

Table II.3.1: Closure properties of families of ω-languages. ($L_1 \cdot L_2$ denotes the concatenation of a language of finite-length words L_1 with an ω-language L_2).

II.3.1. Preliminary properties and constructions. The following constructions will be exploited to prove several closure properties. Indeed, they would be useful even to prove the same properties in the case of finite length languages; however, since such properties have already been proved in previous literature [18, 19] by referring to OPGs rather than OPAs, we present them in this part, which is where they are exploited in this paper.

We begin by introducing the deterministic product of transition functions, defined by extending the usual construction for finite state automata. Such a definition is meaningful when applied to automata that share the same precedence matrix, because they perform the same type of move (push/shift/pop) while reading the input word.

Definition II.3.1. *Let Q_1 and Q_2 be two disjoint sets of states of two deterministic automata sharing the same OP alphabet and let δ_1 and δ_2 be their transition functions. Their product state Q is defined as $Q = Q_1 \times Q_2$ and their product transition function $\delta : Q \times (\Sigma \cup Q) \to Q$ is defined as follows, where $q_1, q_2, p_1, p_2 \in Q, a \in \Sigma$:*

$$\begin{aligned} \delta_{push}((q_1, q_2), a) &= (\delta_{1push}(q_1, a), \delta_{2push}(q_2, a)) \\ \delta_{shift}((q_1, q_2), a) &= (\delta_{1shift}(q_1, a), \delta_{2shift}(q_2, a)) \\ \delta_{pop}((q_1, q_2), (p_1, p_2)) &= (\delta_{1pop}(q_1, p_1), \delta_{2pop}(q_2, p_2)) \end{aligned}$$

Clearly $|Q| = |Q_1| \cdot |Q_2|$.

Although this paper is not concerned with translations, we are going to need the following definition of OP Büchi ω-transducers during some technical steps; other types of ω-transducers could be defined similarly but are not necessary in this paper.

Definition II.3.2 (Operator precedence (Büchi) ω-transducer). *An* operator precedence ω-transducer *is defined in the usual way as a tuple $T = \langle \Sigma, M, Q, I, F, O, \delta, \eta \rangle$ where Σ, M, Q, I, F are defined as in Definition I.2.1, O is a finite set of output symbols, the transition function δ and the output function η are defined by $\langle \delta, \eta \rangle : Q \times (\Sigma \cup Q) \to \wp_F(Q \times O^*)$, where $\wp_F(Q \times O^*)$ denotes the set of finite subsets of $Q \times O^*$, and $\langle \delta, \eta \rangle$ can be seen as the union of three functions, $\langle \delta_{shift}, \eta_{shift} \rangle : Q \times \Sigma \to \wp_F(Q \times O^*)$, $\langle \delta_{push}, \eta_{push} \rangle : Q \times \Sigma \to \wp_F(Q \times O^*)$ and $\langle \delta_{pop}, \eta_{pop} \rangle : Q \times Q \to \wp_F(Q \times O^*)$.*

A configuration of the ω-transducer is denoted $\langle \beta,\ q,\ w \rangle \downarrow z$, where $C = \langle \beta,\ q,\ w \rangle$ is the configuration of the underlying ωOPBA and the string after $\downarrow$ represents the output of the automaton in the configuration. The transition relation $\vdash$ is naturally extended from ωOPBAs, by concatenating the output symbols produced at each move with those generated in the previous moves. Runs and acceptance by the transducer are defined as in the corresponding ωOPBA.

The transduction $\tau(x)$, $x \in \Sigma^\omega$, generated by T is the set of ω-strings produced during its nondeterministic successful runs over x.

The next statement is propaedeutic to many constructive proofs of closure properties, where the operands are in general OPAs with compatible but not identical matrices, and the result's matrix must often be the union of the two original ones. If $\mathcal{A}$ is an OPA with precedence matrix M and $M' \supseteq M$, then clearly $\mathcal{A}$ works also over M' but the language recognized by $\mathcal{A}$ over M' is not necessarily the same, since the presence of precedence relations in M' that are not included in M may allow for successful runs on some words that are, instead, not successful in the original OPA. The next statement proves, however, that the precedence matrix of an OPA can always be extended (up to completion), provided that conflict-freedom is preserved, without affecting the recognized language.

Statement II.3.1 (Extended matrix normal form). *Let $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ be an OPA (over finite-length or omega words) with $|Q| = s$. For any conflict-free OPM $M' \supseteq M$, there exists an OPA with OPM M' that recognizes the same language as $\mathcal{A}$ and has $O(|\Sigma|^2 s)$ states.*

Proof. First consider finite-length words. The new OPA $\mathcal{A}' = \langle \Sigma, M', Q', I', F', \delta' \rangle$ is derived from $\mathcal{A}$ in the following way:

- $Q' = \hat{\Sigma} \times Q \times \hat{\Sigma}$, where $\hat{\Sigma} = (\Sigma \cup \{\#\})$, i.e. the first component of a state is the lookback symbol, the second component is a state of $\mathcal{A}$ and the third component is the lookahead symbol,
- $I' = \{\#\} \times I \times \{a \in \hat{\Sigma} \mid M_{\#a} \neq \varnothing\}$,
- $F' = \{\#\} \times F \times \{\#\}$,

- $\delta' : Q' \times (\Sigma \cup Q') \to \wp(Q')$ is the transition function defined as follows.
  Let $a \in \hat{\Sigma}, b \in \Sigma, q \in Q$. The push transition $\delta'_{\text{push}} : Q' \times \Sigma \to \wp(Q')$ is defined by:

$$\delta'_{\text{push}}(\langle a, q, b\rangle, b) = \{\langle b, p, c\rangle \mid p \in \delta_{\text{push}}(q, b) \wedge M_{ab} = \{\lessdot\} \wedge M_{bc} \neq \varnothing\},$$

  The shift transition $\delta'_{\text{shift}} : Q' \times \Sigma \to \wp(Q')$ is defined analogously:

$$\delta'_{\text{shift}}(\langle a, q, b\rangle, b) = \{\langle b, p, c\rangle \mid p \in \delta_{\text{shift}}(q, b) \wedge M_{ab} = \{\doteq\} \wedge M_{bc} \neq \varnothing\},$$

  The pop transition $\delta'_{\text{pop}} : Q' \times Q' \to \wp(Q')$ is defined by:

$$\delta'_{\text{pop}}(\langle a_1, q_1, a_2\rangle, \langle b_1, q_2, b_2\rangle) = \left\{ \langle b_1, q_3, a_2\rangle \;\middle|\; \begin{array}{c} q_3 \in \delta_{\text{pop}}(q_1, q_2) \wedge \\ M_{a_1 a_2} = \{\gtrdot\} \wedge M_{b_1 a_2} \neq \varnothing \end{array} \right\},$$

  where $a_1, b_2 \in \Sigma, a_2, b_1 \in \hat{\Sigma}, q_1, q_2 \in Q$.

Clearly, the OPA $\mathcal{A}'$ has OPM M' and accepts the same language as $\mathcal{A}$.

This construction can be naturally extended to ωOPAs: in particular, for ωOPBA the set of final states of $\mathcal{A}'$ is $F' = \hat{\Sigma} \times F \times \Sigma$, i.e. a run of $\mathcal{A}'$ is accepting iff it visits infinitely often final states of $\mathcal{A}$, independently of the lookback and the lookahead symbols considered for these states. For ωOPBEA this acceptance component may be further refined as $F' = \{\#\} \times F \times \Sigma$. For ωOPMA, $\mathcal{T}' = \{t \mid t = A_1 \times S \times A_2, S \in \mathcal{T}, A_1 \subseteq \hat{\Sigma}, A_2 \subseteq \Sigma\}$ where $\mathcal{T} \subseteq \wp(Q)$ is the table of $\mathcal{A}$. Furthermore, the transformation preserves determinism. ☐

II.3.1.1. OPA's version without # as lookahead. In this section we illustrate a new version of OPAs that do not rely on the end-marker # for the recognition of a finite length word.

The new model is defined by slightly modifying the semantics of the transition relation and of the acceptance condition of original OPAs, in such a way that a string is accepted by an automaton if it reaches a final state right at the end of the parsing of the whole word, and does not perform any pop move determined by the ending delimiter # to empty the stack; thus the automaton stops just after having pushed on the stack (or updated the top of the stack symbol with) the last symbol of the string.

In this alternative characterization of OPAs, the semantics of the transition relation differs from the classical definition in that, once a configuration with the end-marker as lookahead is reached, the computation cannot evolve in any subsequent configuration, i.e., a pop move $C_1 \vdash_{\not\#} C_2$ with $C_1 = \langle \Pi[a,\ p],\ q,\ x\#\rangle$ is performed only if $x \neq \varepsilon$ (where symbol $\vdash_{\not\#}$ denotes a move according to this variation of the semantics of the transition relation). The language accepted by the automaton according to this new semantics (denoted as $L_{\not\#}$) is the set of words:

$$L_{\not\#}(\mathcal{A}) = \{x \mid \langle \bot,\ q_I,\ x\#\rangle \overset{*}{\vdash}_{\not\#} \langle \bot\gamma,\ q_F,\ \#\rangle, q_I \in I, q_F \in F, \gamma \in \Gamma^*\}$$

This new version of the automaton, called *no-#-look-aheadOPA ($\not\#$OPA)* is closer to the traditional acceptance criterion of general pushdown automata; we emphasize, however, that, unlike normal acceptance by final state of a pushdown automaton, which can perform a number of ε-moves after reaching the end of a string and accepts it if just one of the visited states is final, this type of automaton cannot perform any (pop, i.e., ε-) move when it reaches the end of the input string. The following lemmata (Lemma II.3.3 and Lemma II.3.4) prove the equivalence between the original version of OPAs and the new one.[10]

[10]Only Lemma II.3.3 will be used in Part 2 of this paper but we include both for completeness and possible further exploitation.

LEMMA II.3.3. *Let $\mathcal{A}_1$ be a nondeterministic OPA defined on an OP alphabet (Σ, M) with s states. Then there exists a nondeterministic $\not\#$OPA $\mathcal{A}_2$ on (Σ, M) and $O(s^2)$ states such that $L(\mathcal{A}_1) = L_{\not\#}(\mathcal{A}_2)$.*

We first explain informally the rationale of the simulation of $\mathcal{A}_1$ by $\mathcal{A}_2$, with the aid of an example; then we formally define its construction and prove their equivalence.

Consider a word of finite length w compatible with M: the string $\# w$ can be factored in a unique way as a sequence of bodies of chains and pending letters as

$$\# \, w = \# \, w_1 a_1 w_2 a_2 \ldots w_n a_n$$

where ${}^{a_{i-1}}[w_i]^{a_i}$ are maximal chains and each w_i can be possibly missing, with $a_0 = \#$ and $\forall i : 1 \le i \le n-1 \; a_i \lessdot a_{i+1}$ or $a_i \doteq a_{i+1}$. Let $i_j \in \{1, 2, \ldots, n\}$, $1 \le j \le k$, $k \ge 1$ be indexes such that

$$\# \lessdot a_{i_1} = a_1 \doteq \ldots \doteq a_{i_2-1} \lessdot a_{i_2} \doteq \ldots \doteq a_{i_3-1} \lessdot a_{i_3} \doteq \ldots \doteq a_{i_k-1} \lessdot a_{i_k} \doteq a_{i_k+1} \ldots \doteq a_n \qquad \text{(II.3.1)}$$

When reading w, the symbols of the string are progressively put on the stack, either by a push move or by a shift move, and, whenever a chain w_i is recognized, the symbol on the top of the stack is popped. Hence, after reading w the stack contains only the symbols $\# \; a_{i_2-1} \; a_{i_3-1} \; \ldots \; a_n$ that are the ending symbols of the open chains in the sequence (II.3.1).

When w is read by a standard OPA, the automaton performs a series of pop moves at the end of the string due to the presence of the end delimiter #. These moves progressively empty the stack. The run is accepting if it leads to a final state after all pop moves.

A nondeterministic automaton that, unlike standard OPAs, does not resort to the end delimiter # for the recognition of a string must guess nondeterministically the ending point of each open chain and guess how, in an accepting run, the states in these points would be updated if the final pop moves were progressively performed. The automaton must behave as if, at the same time, it simulates two snapshots of the accepting run of a standard OPA: a move during the reading of the input, and a step during the final pop transitions which will later on empty the stack, leading to a final state. To this aim, the states of a standard OPA are augmented with an additional component.

A $\not\#$OPA $\mathcal{A}_2$ equivalent to a given OPA $\mathcal{A}_1$ thus may be defined so that, after reading each prefix of a word, it reaches a final state whenever, if the word were completed in that point with #, $\mathcal{A}_1$ could reach an accepting state with a sequence of pop moves. In this way, $\mathcal{A}_2$ can guess in advance which words may eventually lead to an accepting state of $\mathcal{A}_1$, without having to wait until reading the delimiter # and to perform final pop moves. In other words, it simulates the possible look-ahead of the # delimiter. Before going into the details of the construction, the following example illustrates the above intuitive description.

EXAMPLE 9. *We refer to the computation of the OPA in Example 3. Consider the input word of this computation without the end-marker #. The sequence of pending letters in the input word corresponds to three open chains, according to* (II.3.1)*, with starting symbols* $+$, $\times$, $\llparenthesis$*, respectively.*

Figure II.3.1 shows the configuration just before looking ahead at the symbol #. The states depicted within a box are those placeholders that an equivalent $\not\#$OPA should fill up to guess in advance the last pop moves $q_3 = \boxed{q_3} \overset{q_0}{\Longrightarrow} \boxed{q_3} \overset{q_1}{\Longrightarrow} \boxed{q_3} \overset{q_1}{\Longrightarrow} \boxed{q_3 \in F_1}$ *of the accepting run.*
The corresponding configuration of the $\not\#$OPA is depicted in Figure II.3.2.

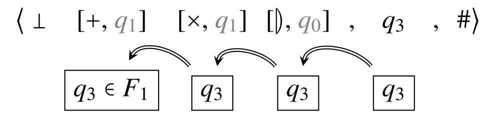


Figure II.3.1: Configuration of the OPA of Example 3 just before looking ahead at #.

$$\langle\ \bot\ \ [+, \langle q_1, \boxed{q_3}\rangle]\ \ [\times, \langle q_1, \boxed{q_3}\rangle]\ \ [⦈, \langle q_0, \boxed{q_3}\rangle]\ \ ,\ \ \langle q_3, \boxed{q_3}\rangle\ \ ,\ \ \#\rangle$$

Figure II.3.2: Configuration of the $\not\#$OPA described in Example 9.

We now proceed with the construction of $\mathcal{A}_2$ and the proof of its equivalence with $\mathcal{A}_1$.

Proof. of Lemma II.3.3

Let $\mathcal{A}_1$ be $\langle \Sigma, M, Q_1, I_1, F_1, \delta_1 \rangle$ and define $\mathcal{A}_2 = \langle \Sigma, M, Q_2, I_2, F_2, \delta_2 \rangle$ as follows.

- $Q_2 = \{B, Z, U\} \times Q_1 \times Q_1$.
  Hence, a state $\langle x, q, p\rangle$ of $\mathcal{A}_2$ is a tuple whose first component denotes a nondeterministic guess for the next input symbol to be read, i.e., whether it is a pending letter which is the initial symbol of an open chain (Z), or a pending letter within an open chain other than the first one (U), or a symbol within a maximal chain (B). The second component of a state represents the current state q in $\mathcal{A}_1$. To illustrate the meaning of the last component, consider an accepting run of $\mathcal{A}_1$ and let q be its current state just before a push move to be performed when reading the first symbol of an open chain; also, let r be the state reached by such push move and s be the state of the automaton when the stack element pushed by this move (possibly updated by subsequent shifts) is going to be popped leading to a state p. Then, in the same position of the corresponding run of $\mathcal{A}_2$, the current state would be $\langle Z, q, p\rangle \in Q_2$ and state $\langle x, r, s\rangle \in Q_2$ will be reached by $\mathcal{A}_2$ (x being nondeterministically anyone of B, Z, U); in other words, the last component p represents a guess about the state that will be reached in $\mathcal{A}_1$ when the stack element pushed by this move will be popped. Hence we can consider only states $\langle Z, q, p\rangle \in Q_2$ such that $s \overset{q}{\Longrightarrow} p$ in $\mathcal{A}_1$ for some $s \in Q_1$. In all the other positions the last component is simply propagated.
  For instance, Figure II.3.3 shows an accepting run on the word $n + n \times ⦇n + n⦈$ of a $\not\#$OPA that is equivalent to the OPA of Example 3. Note that before reading the ⦇, which is the beginning of an open chain, the automaton is in the state $\langle Z, q_0, q_3\rangle$ and then moves to $\langle B, q_2, q_3\rangle$ guessing the state that is reached by the pop move that occurs in the corresponding run of the OPA after reading the ⦈. Before reading the second n, which is the body of a maximal chain, instead, the automaton is in state $\langle B, q_0, q_3\rangle$ and, after popping n from the stack, moves to $\langle Z, q_1, q_3\rangle$ since the following $\times$ is the beginning of an open chain.
- $I_2 = \{\langle x, q, q_F\rangle \mid x \in \{Z, B\}, q \in I_1, q_F \in F_1\}$
- $F_2 = \{\langle Z, q, q\rangle \mid q \in Q_1\}$
- The transition function is defined as the union of three functions.
  The push transition function $\delta_{2\text{push}} : Q_2 \times \Sigma \to \wp(Q_2)$ is defined as follows, where $p, q, r, s \in Q_1$, $a \in \Sigma$.

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

Figure II.3.3: Example of an accepting computation for the word $n + n \times \llparenthesis n + n\rrparenthesis$ of a $\cancel{\#}$OPA that is equivalent to the OPA of Example 3.

– *Pending letter at the beginning of an open chain.*

$$\delta_{2\text{push}}\left(\langle Z, q, p\rangle, a\right) = \left\{\langle x, r, s\rangle \mid x \in \{B, Z, U\}, r \in \delta_{1\text{push}}(q, a), s \overset{q}{\Longrightarrow} p \text{ in } \mathcal{A}_1\right\}$$

– *Symbol of a maximal chain.*

$$\delta_{2\text{push}}\left(\langle B, q, p\rangle, a\right) = \left\{\langle B, r, p\rangle \mid r \in \delta_{1\text{push}}(q, a)\right\}$$

The shift transition function $\delta_{2\text{shift}} : Q_2 \times \Sigma \to \wp(Q_2)$ is defined as follows:

– *Pending letter within an open chain.*

$$\delta_{2\text{shift}}\left(\langle U, q, p\rangle, a\right) = \{\langle x, r, p\rangle \mid x \in \{B, Z, U\}, r \in \delta_{1\text{shift}}(q, a)\}$$

– *Symbol of a maximal chain.*

$$\delta_{2\text{shift}}\left(\langle B, q, p\rangle, a\right) = \{\langle B, r, p\rangle \mid r \in \delta_{1\text{shift}}(q, a)\}$$

Notice that the second component of the states computed by $\delta_{2\text{push}}$ and $\delta_{2\text{shift}}$ is independent of the first component of the starting state.
The pop transition function $\delta_{2\text{pop}} : Q_2 \times Q_2 \to \wp(Q_2)$ can be executed only within a maximal chain since there is no pop determined by the ending delimiter:

$$\delta_{2\text{pop}}\left(\langle B, q, s\rangle, \langle B, p, s\rangle\right) = \left\{\langle x, r, s\rangle \mid x \in \{B, Z, U\}, q \overset{p}{\Longrightarrow} r \text{ in } \mathcal{A}_1\right\}$$

All other moves lead to an error state.

Let us prove first $L(\mathcal{A}_1) \subseteq L_{\cancel{\#}}(\mathcal{A}_2)$. Consider a word $w \in L(\mathcal{A}_1)$. Then there exists a support $q \overset{w}{\rightsquigarrow} q'$ in $\mathcal{A}_1$ with $q \in I_1$ and $q' \in F_1$. If $w = w_1 a_1 w_2 a_2 \ldots w_n a_n$ where a_i are pending letters and w_i are maximal chains, let k be the number of open chains determined by the sequence of pending letters in w according to the structure (II.3.1), and let $a_{i_1} = a_1, a_{i_2}, \ldots, a_{i_k}$ be their initial symbols. Also, for every $i = 2, \ldots, n$, let $t(i)$ be the greatest index t such that $i_t < i$, i.e., a_i is within the $t(i)$-th open chain beginning with $a_{i_{t(i)}}$. In particular, for $i = n$, if

$a_{n-1} \lessdot a_n$ then $i_k = n$, otherwise $t(n) = k$. As a notational convention, denote by $\longmapsto$ a move that can be either a push or a shift.

Then the above support for w can be decomposed as

$$q = \widetilde{q}_0 \overset{w_1}{\rightsquigarrow} q_1 \xrightarrow{a_1} \widetilde{q}_1 \overset{w_2}{\rightsquigarrow} q_2 \overset{a_2}{\longmapsto} \ldots \overset{w_n}{\rightsquigarrow} q_n \overset{a_n}{\longmapsto} \widetilde{q}_n = p_k \tag{II.3.2}$$

$$\widetilde{q}_n = p_k \overset{q_{i_k}}{\Longrightarrow} p_{k-1} \overset{q_{i_{k-1}}}{\Longrightarrow} p_{k-2} \Longrightarrow \ldots \Longrightarrow p_2 \overset{q_{i_2}}{\Longrightarrow} p_1 \overset{q_{i_1}=q_1}{\Longrightarrow} p_0 = q'$$

where $q_i = \widetilde{q}_{i-1}$ if $w_i = \varepsilon$ for $i = 1, 2, \ldots, n$. Notice that, for every t, q_{i_t} is the state reached in this path before the push move that pushes symbol a_{i_t} on the stack; moreover, when the last symbol in the open chain beginning with a_{i_t} is to be popped, the current state is p_t and then the symbol on the top of the stack (whose state component is q_{i_t}) is removed and $\mathcal{A}_1$ moves to state p_{t-1}.

Starting with state $\langle Z, q_1, p_0\rangle$ if $w_1 = \varepsilon$ or with $\langle B, \widetilde{q}_0, p_0\rangle \overset{w_1}{\rightsquigarrow} \langle Z, q_1, p_0\rangle$ if $w_1 \neq \varepsilon$, an accepting computation of $\mathcal{A}_2$ can be built on the basis of the following facts:

- Since $\mathcal{A}_1$ performs $q_1 \xrightarrow{a_1} \widetilde{q}_1$ and $p_1 \overset{q_1}{\Longrightarrow} p_0$, then $\delta_{2\text{push}}(\langle Z, q_1, p_0\rangle, a_1) \ni \langle x, \widetilde{q}_1, p_1\rangle$ in $\mathcal{A}_2$ for $x \in \{B, Z, U\}$. This is a push move that can be applied at the beginning of the first open chain, a_1, where p_1 is the guess about the state that will be reached before the stack symbol pushed on the stack by this move will be popped.
- In general, for every t, since $\mathcal{A}_1$ executes $q_{i_t} \xrightarrow{a_{i_t}} \widetilde{q}_{i_t}$ and $p_t \overset{q_{i_t}}{\Longrightarrow} p_{t-1}$, then $\delta_2(\langle Z, q_{i_t}, p_{t-1}\rangle, a_{i_t}) \ni \langle x, \widetilde{q}_{i_t}, p_t\rangle$ for $x \in \{B, Z, U\}$. This is a push move that can be applied at the beginning of the t-th open chain, i.e. when reading a_{i_t}, where p_t is the guess about the state that will be reached before the stack symbol with the last letter of the chain will be popped. In particular, if $i_k = n$, we can reach state $\langle Z, \widetilde{q}_n, p_k\rangle$ which is final in $\mathcal{A}_2$ since $q_n = p_k$.
- For every maximal chain w_i of w (with $i \geq 2$) consider its support $\widetilde{q}_{i-1} \overset{w_i}{\rightsquigarrow} q_i$ in the sequence (II.3.2). Then in $\mathcal{A}_2$ we have a sequence of moves starting from a state $\langle B, \widetilde{q}_{i-1}, p_{t(i)}\rangle$ and reading w_i, that ends in $\langle x, q_i, p_{t(i)}\rangle$, where $x \in \{U, Z\}$. Notice that the last component of the states does not change because we are within a maximal chain. During the reading of w_i, the last component is equal to $p_{t(i)}$, as guessed by the push move at the beginning of the current open chain.
- For every $i \notin \{i_1, i_2, \ldots, i_k\}$, since $\delta_{1\text{shift}}(q_i, a_i) \ni \widetilde{q}_i$, then $\delta_{2\text{shift}}(\langle U, q_i, p_{t(i)}\rangle, a_i)$ contains $\langle x, \widetilde{q}_i, p_{t(i)}\rangle$, for $x \in \{B, Z, U\}$. In particular, if $n \neq i_k$, then $t(n) = k$ and for $i = n$ we can reach state $\langle Z, \widetilde{q}_n, p_k\rangle$ which is final in $\mathcal{A}_2$, since $\widetilde{q}_n = p_k$.

Thus, by composing in the right order the previous moves, one can obtain an accepting computation for w in $\mathcal{A}_2$.

Conversely, to prove that $L_{\#}(\mathcal{A}_2) \subseteq L(\mathcal{A}_1)$, consider a word $w \in L_{\#}(\mathcal{A}_2)$. This means that there exists a successful run of $\mathcal{A}_2$ on w. Let w be factorized as above; then the accepting run for w can be decomposed as

$$\pi_0 \overset{w_1}{\rightsquigarrow} \rho_1 \xrightarrow{a_1} \pi_1 \overset{w_2}{\rightsquigarrow} \rho_2 \ldots \rho_i \overset{a_i}{\longmapsto} \pi_i \overset{w_{i+1}}{\rightsquigarrow} \ldots \overset{w_n}{\rightsquigarrow} \rho_n \overset{a_n}{\longmapsto} \pi_n$$

where $\pi_i, \rho_i \in Q_2, \rho_i = \pi_{i-1}$ if $w_i = \varepsilon$, $\pi_0 \in I_2$ and $\pi_n \in F_2$. By projecting this path on the second component of states π_i and ρ_i (let them respectively be p_i and $r_i \in Q_1$), we obtain a path in $\mathcal{A}_1$ labelled by w. This path is not accepting because there are symbols left on the stack that need to be popped, but we can complete this path arguing by induction on the structure of maximal chains according to the definition of δ_2. Precisely, one can verify that Q_1 contains suitable states p_i (for $0 \leq i \leq n$), r_i (for $1 \leq i \leq n$), s_t (for $1 \leq t \leq k$), with $r_i = p_{i-1}$ whenever $w_i = \varepsilon$, such that the following facts hold.

- $\pi_0 \in I_2$, hence $\pi_0 = \langle x_0, p_0, s_0\rangle$, with $p_0 \in I_1$ and $s_0 \in F_1$; x_0 is B if $w_1 \neq \varepsilon$, otherwise $x_0 = Z$.
- $\pi_0 \overset{w_1}{\leadsto} \rho_1$ in $\mathcal{A}_2$ implies that the last component of state π_0 is propagated through chain w_1 without change; hence $\rho_1 = \langle Z, r_1, s_0\rangle$ with $p_0 \overset{w_1}{\leadsto} r_1$ in $\mathcal{A}_1$.
- $\rho_1 \xrightarrow{a_1} \pi_1$ is a push move of $\mathcal{A}_2$ at the beginning of an open chain, and this implies that the last component of π_1 is a guess on the state from which $\mathcal{A}_1$ would perform the corresponding pop, so that $\pi_1 = \langle x_1, p_1, s_1\rangle$ with $r_1 \xrightarrow{a_1} p_1$ and $s_1 \overset{r_1}{\Longrightarrow} s_0$ in $\mathcal{A}_1$; the first component is $x_1 = B$ if $w_2 \neq \varepsilon$ otherwise x_1 equals Z or U according to whether a_2 starts an open chains or not, respectively,
- The pop moves within $\pi_i \overset{w_{i+1}}{\leadsto} \rho_{i+1}$ for $1 \leq i < i_2$, and the shift moves within an open chain $\rho_i \overset{a_i}{\dashrightarrow} \pi_i$ for $1 < i < i_2$ propagate with no change the last component. Hence $\rho_i = \langle U, r_i, s_1\rangle$ and $\pi_i = \langle x_i, p_i, s_1\rangle$ with $p_{i-1} \overset{w_i}{\leadsto} r_i \overset{a_i}{\dashrightarrow} p_i$ in $\mathcal{A}_1$. The first component is $x_i = B$ if $w_i \neq \varepsilon$, otherwise $x_i = Z$ for $i = i_2 - 1$, and $x_i = U$ in the other cases.
- $\rho_{i_2} \xrightarrow{a_{i_2}} \pi_{i_2}$ is a push move of $\mathcal{A}_2$ at the beginning of an open chain, and this implies that the last component of π_{i_2} is a guess on the state from which $\mathcal{A}_1$ would perform the corresponding pop, so that $\pi_{i_2} = \langle x_{i_2}, p_{i_2}, s_2\rangle$ with $r_{i_2} \xrightarrow{a_{i_2}} p_{i_2}$ and $s_2 \overset{r_{i_2}}{\Longrightarrow} s_1$ in $\mathcal{A}_1$. The first component is $x_{i_2} = B$ if $w_{i_2} \neq \varepsilon$ otherwise x_1 equals Z or U according to whether $a_{i_2} + 1$ begins an open chains or not, respectively.
- Similarly for the following moves in the run.

In general, we get

$$\begin{aligned}
\rho_i &= \langle y_i, r_i, s_{t(i)}\rangle && \text{for every } i = 1, 2, \ldots, n,\\
\pi_i &= \langle x_i, p_i, s_{t(i)}\rangle && \text{for every } i \notin \{i_1, i_2, \ldots, i_k\},\\
\pi_{i_t} &= \langle x_{i_t}, p_{i_t}, s_t\rangle && \text{for every } t = 1, 2, \ldots, k,\\
\text{with } r_i \overset{a_i}{\longmapsto} p_i,\ s_t &\overset{r_{i_t}}{\Longrightarrow} s_{t-1},\ p_{i-1} \overset{w_i}{\leadsto} r_i \text{ in } \mathcal{A}_1 &&\\
\text{and } y_i \in \{Z, U\}, x_i &\in \{B, Z, U\} && \text{for every } i \text{ and } t.
\end{aligned}$$

For $i = n$ we have $n = i_k$ or $t(n) = k$, hence $\pi_n = \langle x_n, p_n, s_k\rangle$, and $p_n = s_k$ and $x_n = Z$ since $\pi_n \in F_2$. Thus, in $\mathcal{A}_1$ there is an accepting run

$$I_1 \ni p_0 \overset{w_1}{\leadsto} r_1 \xrightarrow{a_1} p_1 \overset{w_2}{\leadsto} r_2 \ldots r_i \overset{a_i}{\longmapsto} p_i \overset{w_{i+1}}{\leadsto} \ldots \overset{w_n}{\leadsto} r_n \overset{a_n}{\longmapsto} p_n = s_k$$

$$p_n = s_k \overset{r_{i_k}}{\Longrightarrow} s_{k-1} \overset{r_{i_{k-1}}}{\Longrightarrow} s_{k-2} \Longrightarrow \ldots \Longrightarrow s_2 \overset{r_{i_2}}{\Longrightarrow} s_1 \overset{r_{i_1} = r_1}{\Longrightarrow} s_0 \in F_1.$$

□

The next lemma completes the proof of equivalence between OPAs and $\not\#$OPAs.

LEMMA II.3.4. *Let $\mathcal{A}_2$ be a nondeterministic $\not\#$OPA defined on an OP alphabet (Σ, M) with s states. Then there exists a nondeterministic OPA $\mathcal{A}_1$ on (Σ, M) and $O(|\Sigma| s)$ states, such that $L(\mathcal{A}_1) = L_{\not\#}(\mathcal{A}_2)$.*

Proof. Let $\mathcal{A}_2$ be $\langle \Sigma, M, Q, I, F, \delta\rangle$ and consider, first, an equivalent form of $\mathcal{A}_2$, where all states are enriched with a lookahead symbol and no final state is reached by a pop edge: $\tilde{\mathcal{A}}_2 = \langle \Sigma, M, Q_2, I_2, F_2, \delta_2\rangle$, where

- $Q_2 = Q \times \hat{\Sigma}$, where $\hat{\Sigma} = (\Sigma \cup \{\#\})$, i.e. the first component of a state is a state of $\mathcal{A}_2$ and the second component of the state is the lookahead symbol,
- $I_2 = I \times \{a \in \hat{\Sigma} \mid M_{\#a} \neq \varnothing\}$ is the set of initial states of $\tilde{\mathcal{A}}_2$,
- $F_2 = F \times \{\#\}$

- the transition function $\delta_2 : Q_2 \times (\Sigma \cup Q_2) \to \wp(Q_2)$ is defined in the following natural way, where $a, b \in \Sigma, p, q, r \in Q$:
  - $\delta_{2\text{push}}(\langle p, a\rangle, a) = \{\langle q, b\rangle \mid q \in \delta_{\text{push}}(p, a) \wedge M_{ab} \neq \emptyset\}$,
  - $\delta_{2\text{shift}}(\langle p, a\rangle, a) = \{\langle q, b\rangle \mid q \in \delta_{\text{shift}}(p, a) \wedge M_{ab} \neq \emptyset\}$,
  - $\delta_{2\text{pop}}(\langle p, a\rangle, \langle q, b\rangle) = \{\langle r, a\rangle \mid r \in \delta_{\text{pop}}(p, q)\} \setminus F_2$.

It is easy too see that $L_{\not\#}(\mathcal{A}_2) = L_{\not\#}(\tilde{\mathcal{A}}_2)$. Furthermore, the final states of $\tilde{\mathcal{A}}_2$ cannot be reached by pop edges: in fact, these pop transitions cannot be performed by a $\not\#$OPA according to the semantics of the transition relation $\vdash_{\not\#}$, since it stops a computation right before reading the delimiter #, when the parsing of the word ends.

Thus, we build, without loss of generality, an OPA $\mathcal{A}_1$ equivalent to the $\not\#$OPA $\tilde{\mathcal{A}}_2$. $\mathcal{A}_1 = \langle \Sigma, M, Q_1, I_1, F_1, \delta_1\rangle$ has only one final state, reachable through a pop edge by all final states of $\tilde{\mathcal{A}}_2$. Its role is to let $\mathcal{A}_1$ empty the stack after reading a word that is accepted by $\tilde{\mathcal{A}}_2$.

- $Q_1 = Q_2 \cup \{q_{\text{accept}}\}$
- $I_1 = I_2 \cup \{q_{\text{accept}}\}$ if $I_2 \cap F_2 \neq \emptyset$; $I_1 = I_2$ otherwise
- $F_1 = \{q_{\text{accept}}\}$
- The transition function δ_1 equals δ_2 on all states in Q_2; in addition $\mathcal{A}_1$ has departing pop edges from the final states in F_2 to q_{accept} and q_{accept} has no outgoing push/shift edge but only self-loops pop edges.
  The push transition function $\delta_{1\text{push}} : Q_1 \times \Sigma \to \wp(Q_1)$ is defined as $\delta_{1\text{push}}(q, c) = \delta_{2\text{push}}(q, c), \forall q \in Q_2, c \in \Sigma$. The shift function is defined analogously.
  The pop transition $\delta_{1\text{pop}} : Q_1 \times Q_1 \to \wp(Q_1)$ is defined by:

  $\delta_{1\text{pop}}(q, p) = \delta_{2\text{pop}}(q, p), \forall q, p \in Q_2$

  $\delta_{1\text{pop}}(q, p) = q_{\text{accept}}, \forall q \in (F_2 \cup \{q_{\text{accept}}\}), p \in Q_2,$

We now show that $L(\mathcal{A}_1) = L_{\not\#}(\tilde{\mathcal{A}}_2)$.

$L(\mathcal{A}_1) \subseteq L_{\not\#}(\tilde{\mathcal{A}}_2)$: in fact, if the OPA $\mathcal{A}_1$ recognizes a word, then it is either the empty word and thus $q_{\text{accept}} \in I_1$ and also $\tilde{\mathcal{A}}_2$ has a successful run on it, or $\mathcal{A}_1$ recognizes a word $w \neq \varepsilon$ and there exists a run σ of $\mathcal{A}_1$ which ends in the final state q_{accept} with empty stack. Notice that q_{accept} is reached by a pop move from a state in F_2, say $q_f \in F_2$:

$$\sigma : q_0 \in I_2 \overset{w}{\rightsquigarrow} q_f \Longrightarrow q_{\text{accept}} (\overset{p \in Q_1}{\Longrightarrow} q_{\text{accept}})^*$$

and q_f itself is reached exactly when the reading of w is finished, since, as said before, a state in F_2 cannot be reached by pop moves. This condition is necessary to avoid the presence of sequences of pop moves from non-accepting states toward final states. Then the path from q_0 to q_f, which traverses the same states and edges as σ, represents a run of $\tilde{\mathcal{A}}_2$ which ends in a final state q_f right after the reading of the whole word, thus accepting w. Conversely, the relation $L(\mathcal{A}_1) \supseteq L_{\not\#}(\tilde{\mathcal{A}}_2)$ derives easily from the fact that, if $\tilde{\mathcal{A}}_2$ accepts a word along a successful run, then $\mathcal{A}_1$ recognizes the word along the same run, possibly emptying the stack in the final state q_{accept}. ∎

Remark 2. *With some further effort –and a further exponential leap in the automaton's size– a deterministic version of this $\not\#$OPA could also be built. We did not include it here, however, since the $\not\#$OPA construction will be applied only in this part to prove the closure w.r.t. concatenation with finite length languages of ωOPLs: we will see that such a closure holds only for nondeterministic automata.*

II.3.2. Closure properties and emptiness problem for class $\mathcal{L}(\omega$OPBA). $\mathcal{L}(\omega$OPBA) enjoys all closure and decidability properties suitable for model checking. Precisely, the emptiness problem is decidable for OPAs in polynomial time because they can be interpreted as pushdown automata on infinite-length words: e.g., [13] shows an algorithm that decides the alternation-free modal μ-calculus for context-free processes, with linear complexity in the size of the system's representation.

The following theorems state that $\mathcal{L}(\omega$OPBA) is a Boolean algebra closed under concatenation.

Theorem II.3.5 ($\mathcal{L}(\omega$OPBA) is closed under intersection). *Let L_1 and L_2 be ω-languages recognized by two ωOPBA defined over the same alphabet Σ, with compatible precedence matrices M_1 and M_2 and with s_1 and s_2 states respectively. Then $L = L_1 \cap L_2$ is recognizable by an ωOPBA with OPM $M = M_1 \cap M_2$ and $O(s_1 s_2)$ states.*

Proof. Let $\mathcal{A}_1 = \langle \Sigma, M_1, Q_1, I_1, F_1, \delta_1 \rangle$ and $\mathcal{A}_2 = \langle \Sigma, M_2, Q_2, I_2, F_2, \delta_2 \rangle$ be two ωOPBA with $L(\mathcal{A}_1) = L_1$ and $L(\mathcal{A}_2) = L_2$ and with compatible precedence matrices M_1 and M_2. Suppose, without loss of generality, that Q_1 and Q_2 are disjoint and do not contain $\{0,1,2\}$.

First, observe that, the two OPMs being compatible, at each move either the two automata perform the same type of move (push/shift/pop), or at least one of them stops without accepting since its transition function is not defined.

An ωOPBA that recognizes $L_1 \cap L_2$ is defined in a similar way as for classical finite-state Büchi automata; precisely, $\mathcal{A} = \langle \Sigma, M = M_1 \cap M_2, Q, I, F, \delta \rangle$ where:

- $Q = Q_1 \times Q_2 \times \{0,1,2\}$,
- $I = I_1 \times I_2 \times \{0\}$,
- $F = Q_1 \times Q_2 \times \{2\}$
- the transition function $\delta : Q \times (\Sigma \cup Q) \rightarrow \wp(Q)$ is defined as follows, where $p_1, q_1, p_2, q_2 \in Q, a \in \Sigma$:
  - $\delta_{\text{push}}(\langle p_1, p_2, x \rangle, a) = \{\langle r_1, r_2, y \rangle \mid r_1 \in \delta_{1\text{push}}(p_1, a) \wedge r_2 \in \delta_{2\text{push}}(p_2, a)\}$
  - $\delta_{\text{shift}}(\langle p_1, p_2, x \rangle, a) = \{\langle r_1, r_2, y \rangle \mid r_1 \in \delta_{1\text{shift}}(p_1, a) \wedge r_2 \in \delta_{2\text{shift}}(p_2, a)\}$
  - $\delta_{\text{pop}}(\langle p_1, p_2, x \rangle, \langle q_1, q_2, z \rangle) = \{\langle r_1, r_2, y \rangle \mid r_1 \in \delta_{1\text{pop}}(p_1, q_1) \wedge r_2 \in \delta_{2\text{pop}}(p_2, q_2)\}$

  and the third component of the states is computed as follows:
  - if $x = 0$ and $r_1 \in F_1$ then $y = 1$
  - if $x = 1$ and $r_2 \in F_2$ then $y = 2$
  - if $x = 2$ then $y = 0$
  - $y = x$ otherwise.

Reading an input string, the automaton $\mathcal{A}$ simulates $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively on the first two components of the states, whereas the third component keeps track of the succession of visits of the two automata to their final states: in particular its value is 0 at the beginning, then switches from 0 to 1, from 1 to 2 and then back to 0, whenever the first automaton reaches a final state and the other one visits a final state afterwards. This cycle is repeated infinitely often whenever both the automata reach their final states infinitely many times along their run.

Conversely, if an ω-word x does not belong to $L_1 \cap L_2$, then at least one of the runs of $\mathcal{A}_1$ and $\mathcal{A}_2$ must either stop because the transition function of the automaton is undefined for the given input or it does not visit infinitely often final states. Hence, $\mathcal{A}$ cannot have a successful run on x and the word is rejected by $\mathcal{A}$ too. $\square$

Theorem II.3.6 ($\mathcal{L}(\omega$OPBA) is closed under union). *Let L_1 and L_2 be ω-languages recognized by two ωOPBA defined over the same alphabet Σ, with compatible precedence matrices M_1 and M_2 and with s_1 and s_2 states respectively. Then $L = L_1 \cup L_2$ is recognizable by an ωOPBA with OPM $M = M_1 \cup M_2$ and $O(|\Sigma|^2(s_1 + s_2))$ states.*

Proof. Let A_1 and $\mathcal{A}_2$ be ωOPBAs accepting L_1 and L_2 over OPMs M_1 and M_2, respectively. Without loss of generality we may assume $M = M_1 = M_2$ (otherwise one can apply Statement II.3.1 increasing the number of states by a factor $|\Sigma|^2$). For $i = 1, 2$, let $\mathcal{A}_i = \langle \Sigma, M, Q_i, I_i, F_i, \delta_i \rangle$. Then the ω-language $L = L_1 \cup L_2$ is recognized by the ωOPBA $\mathcal{A} = \langle \Sigma, M, Q = Q_1 \cup Q_2, I = I_1 \cup I_2, F = F_1 \cup F_2, \delta \rangle$ whose transition function $\delta : Q \times (\Sigma \cup Q) \to \wp(Q)$ is the nondeterministic union of δ_1 and δ_2, defined by setting $\forall p, q \in Q, a \in \Sigma$:

$$\delta_{\text{push}}(q,a) = \begin{cases} \delta_{1\text{push}}(q,a) & \text{if } q \in Q_1 \\ \delta_{2\text{push}}(q,a) & \text{if } q \in Q_2 \end{cases}, \qquad \delta_{\text{shift}}(q,a) = \begin{cases} \delta_{1\text{shift}}(q,a) & \text{if } q \in Q_1 \\ \delta_{2\text{shift}}(q,a) & \text{if } q \in Q_2 \end{cases},$$

$$\delta_{\text{pop}}(p,q) = \begin{cases} \delta_{1\text{pop}}(p,q) & \text{if } p, q \in Q_1 \\ \delta_{2\text{pop}}(p,q) & \text{if } p, q \in Q_2 \end{cases}.$$

The above definition is well-posed since it applies to automata that share the same precedence matrix, because they perform the same type of move (push/shift/pop) while reading the input word.

Since the sets of states of the two automata are disjoint and Q is their union, then for every $x \in \Sigma^\omega$ there exists a successful run in $\mathcal{A}$ iff there exists a successful run of $\mathcal{A}_1$ on x or a successful run of $\mathcal{A}_2$ on x.

Clearly, the number of states of A is $|Q| = |Q_1| + |Q_2|$ and this concludes the proof, recalling the possible factor $|\Sigma|^2$ implied by Statement II.3.1. ☐

Theorem II.3.7 (Closure of $\mathcal{L}(\omega$OPBA) under complementation). *Let M be a conflict-free precedence matrix on an alphabet Σ. Let L be an ω-language on Σ that is recognized by a nondeterministic ωOPBA with precedence matrix M and s states. Then the complement of L w.r.t. L_M (the language of all the words $x \in \Sigma^\omega$ compatible with M) is recognized by an ωOPBA with the same precedence matrix M and $2^{O(s^2 + |\Sigma| s \, log |\Sigma| s)}$ states.*

Proof. The proof follows to some extent the structure of the corresponding proof for $\mathcal{L}(\omega$BVPA) [4], but it exhibits some relevant technical aspects which distinctly characterize it; in particular, we need to introduce an ad-hoc factorization of ω-words due to the more complex management of the stack performed by OPAs.

Let $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ be a nondeterministic ωOPBA with $|Q| = s$. Without loss of generality $\mathcal{A}$ can be considered complete with respect to the transition function δ, i.e. there is a run of $\mathcal{A}$ on every ω-word on Σ compatible with M.

An ω-word $w \in \Sigma^\omega$ compatible with M can be factored as a sequence of chains and pending letters $w = w_1 w_2 w_3 \ldots$ where either $w_i = a_i \in \Sigma$ is a pending letter or $w_i = a_{i1} a_{i2} \ldots a_{in}$ is the body of the chain ${}^{l_i}[w_i]^{first_{i+1}}$, where l_i denotes the last pending letter preceding w_i in the word and $first_{i+1}$ denotes the first letter of word w_{i+1}. Let also, by convention, $a_0 = \#$ be the first pending letter.

Such factorization is not unique, since a string w_i can be nested into a larger chain having the same preceding pending letter. The factorization is unique, however, if we additionally require that the body w_i has no prefix (including itself) $u_i b$ such that ${}^{l_i}[u_i]^b$ is a chain; in fact, in this case, as soon as a chain body with its context is identified after a pending letter, it becomes part of the factorization and what follows is either the beginning of a new body or a new pending letter.

For instance, for the word $w = \underbrace{\lessdot a \lessdot c \gtrdot}\, b \underbrace{\lessdot a \gtrdot}\, \underbrace{d \gtrdot}\, b \ldots$, with precedence relations in the OPM $a \gtrdot b$ and $b \lessdot d$, two possible factorizations are $w = w_1 b w_3 b \ldots$ and $w = \overline{w}_1 b \overline{w}_3 \overline{w}_4 b \ldots$, where b is a pending letter and ${}^{\#}[w_1]^b = {}^{\#}[\overline{w}_1]^b = {}^{\#}[ac]^b$, ${}^b[w_3]^b = {}^b[\overline{w}_3 d]^b$, ${}^b[\overline{w}_3]^d = {}^b[a]^d$ and ${}^b[\overline{w}_4]^b = {}^b[d]^b$ are chains. The second factorization is the unique one where each word $\overline{w}_i$ has no prefix $u_i b$ such that ${}^{l_i}[u_i]^b$ is a chain.

Let $x \in \Sigma^*$ be the body of some chain ${}^a[x]^b$ and let $T(x)$ be the set of all triples $(q,p,f) \in Q \times Q \times \{0,1\}$ such that there exists a support $q \overset{x}{\rightsquigarrow} p$ in $\mathcal{A}$, and $f = 1$ iff the support contains a state in F. Also let $\mathcal{T}$ be the set of all such $T(x)$, i.e., $\mathcal{T}$ contains sets of triples identifying all supports for some chain, and set PR to be the finite alphabet $\Sigma \cup \mathcal{T}$. A *pseudorun* for the word w in $\mathcal{A}$'s uniquely factorized as $w_1w_2w_3\ldots$ as stated above, is the ω-word $w' = y_1y_2y_3\ldots \in PR^\omega$ where $y_i = a_i$ if w_i is a pending letter, otherwise $y_i = T(w_i)$.

For the unique factorization in the example above, then, $w' = T(ac)\ b\ T(a)\ T(d)\ b\ldots$.

The automaton recognizing the complement of $L = L(\mathcal{A})$ w.r.t. L_M can be built as an "online composition" of a transducer ωOPBA $\mathcal{B}$ that computes the pseudorun corresponding to an input word w, and a Büchi finite-state automaton $\mathcal{B}_R$ that recognizes all the pseudoruns of ω-words not accepted by $\mathcal{A}$: while reading w, $\mathcal{B}$ outputs the pseudorun w' of w online, and the states of $\mathcal{B}_R$ are updated accordingly. The automaton accepts if both $\mathcal{B}$ and $\mathcal{B}_R$ reach infinitely often final states.

In order to define $\mathcal{B}_R$ we first define a nondeterministic Büchi finite-state automaton $\mathcal{A}_R = \langle PR, Q_{\mathcal{A}_R}, I_{\mathcal{A}_R}, F_{\mathcal{A}_R}, \delta_{\mathcal{A}_R}\rangle$ over the alphabet PR whose language includes all pseudoruns w' of any words $w \in L(\mathcal{A})$.

The states of $\mathcal{A}_R$ correspond to the states of $\mathcal{A}$, but are extended with a lookback symbol that, in a correct pseudorun, represents the last pending letter of the input word read so far. $\mathcal{A}_R$ has all transitions corresponding to $\mathcal{A}$'s push and shift transitions but is devoid of pop edges (in fact it is a finite state automaton). In addition, for every $S \in \mathcal{T}$ it is endowed with arcs labeled S which link, for each triple (q,p,f) in S and $a \in \hat{\Sigma} = \Sigma \cup \{\#\}$, either the pair of states $\langle a,q\rangle$, $\langle a,p\rangle$ if $f = 0$, or $\langle a,q\rangle$, $\langle a,p'\rangle$ if $f = 1$, where $\langle a,p'\rangle$ is a new final state which takes into account the states in F met along the support $q \rightsquigarrow p$ and which has the same outgoing edges as $\langle a,p\rangle$.

Formally, $Q_{\mathcal{A}_R} = \hat{\Sigma} \times (Q \cup Q')$, where $Q' = \{q' \mid q \in Q\}$, $I_{\mathcal{A}_R} = \{\#\} \times I$, $F_{\mathcal{A}_R} = \hat{\Sigma} \times (F \cup Q')$. The transition function of $\mathcal{A}_R$ is defined as follows, where $a \in \hat{\Sigma}, q \in Q, q' \in Q', S \in \mathcal{T}$ (δ_{push} and δ_{shift} are the transition functions of $\mathcal{A}$):

- $\delta(\langle a,q\rangle, b) = \begin{cases} \langle b, \delta_{\text{push}}(q,b)\rangle & \text{if } a \lessdot b \\ \langle b, \delta_{\text{shift}}(q,b)\rangle & \text{if } a \doteq b \end{cases}$
- $\delta(\langle a,q\rangle, S) = \{\langle a,p\rangle \mid \langle q,p,0\rangle \in S\} \cup \{\langle a,p'\rangle \mid \langle q,p,1\rangle \in S\}$
- $\delta(\langle a,q'\rangle, X) = \delta(\langle a,q\rangle, X),\ \forall X \in PR$.

Notice that, given a set $S \in \mathcal{T}$, the existence of an edge S between the pairs of states q,p in the triples in S can be decided in an effective way.

The automaton $\mathcal{A}_R$ built so far is able to parse all pseudoruns and recognizes all pseudoruns of ω-words recognized by $\mathcal{A}$. However, since its moves are no longer completely determined by the OPM M, it can also accept input words along the edges of the graph of $\mathcal{A}$ that are not pseudoruns since they do not correspond to a correct factorization on PR. This is irrelevant, however, since the aim of the proof is to devise an automaton recognizing the complement of $L(\mathcal{A})$, and all the words in $L_M \setminus L(\mathcal{A})$ are parsed along pseudoruns, which are not accepted by $\mathcal{A}_R$. If one gives as input words only pseudoruns (and not generic words on PR), then they will be accepted by $\mathcal{A}_R$ if the corresponding words on Σ belong to $L(\mathcal{A})$, and they will be rejected if the corresponding words do not belong to $L(\mathcal{A})$ (see Figure II.3.4). Then we can construct a deterministic Büchi automaton $\mathcal{B}_R$ that accepts the complement of $L(\mathcal{A}_R)$, on the alphabet PR [36]. If $\mathcal{B}_R$ receives only input words on PR that are pseudoruns, then it will accept only words in $L_M \setminus L(\mathcal{A})$.

Now we define a nondeterministic transducer ωOPBA $\mathcal{B}$ which on reading w generates online the pseudorun w'. The transducer $\mathcal{B}$ nondeterministically guesses whether the next input symbol is a pending letter, the beginning of a chain appearing in the factorization of w,

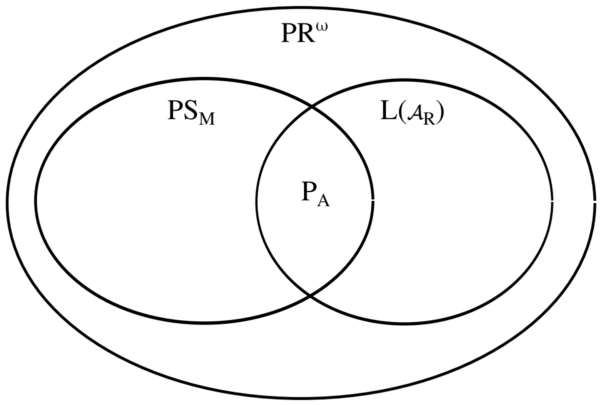


Figure II.3.4: Containment relations for languages, where $PS_M = \{w' \in PR^\omega \mid w' \text{ is the pseudorun in } \mathcal{A} \text{ for } w \in L_M\}$ and $P_A = \{w' \in PR^\omega \mid w' \text{ is the pseudorun in } \mathcal{A} \text{ for } w \in L(\mathcal{A})\}$.

or a symbol within such a chain, and uses stack symbols Z, B, or elements in $\mathcal{T}$, respectively, to distinguish these three cases.

Whenever the automaton reads a pending letter it outputs the same letter, whereas when it completes the recognition of a chain of the factorization, performing a pop move that removes from the stack an element with state B, it outputs the set of all the pairs of states which define a support for the chain. Thus, the output w' produced by $\mathcal{B}$ is unique, despite the nondeterminism of the translator.

Formally, the transducer ωOPBA $\mathcal{B} = \langle \Sigma, M, Q_B, I_B, F_B, PR, \delta_B, \eta_B \rangle$ is defined as follows:

- $Q_B = \{Z, B\} \cup \mathcal{T}$, i.e., a state in Q_B represents the guess whether the next symbol to be read is a pending letter (Z), the beginning of a chain (B), or a letter within such a chain w_i ($T \in \mathcal{T}$). In the third case, T contains all information necessary to correctly simulate the moves of $\mathcal{A}$ during the parsing of the chain w_i of w, and compute the corresponding symbol y_i of w'. In particular, T is a set comprising all triples (r, q, v) where r represents the state reached before the last push move, q represents the current state reached by $\mathcal{A}$, and v is a bit that reminds whether, while reading the chain, a state in F has been encountered (as in the construction of a deterministic OPA on words of finite length, it is necessary to keep track of the state from which the parsing of a chain started, to avoid erroneous merges of runs on pop moves).
- $I_B = F_B = \{B, Z\}$.
- The transition function and the output function are defined as the union of three pairs of functions. Let $a \in \Sigma, T, S \in \mathcal{T}$.
  The push pair $\langle \delta_{\text{Bpush}}, \eta_{\text{Bpush}} \rangle : Q_B \times \Sigma \to \wp_F(Q_B \times PR^*)$ is defined as follows, where the symbols after $\downarrow$ denote the output.
  - *Push of a pending letter.*

$$\langle \delta_{\text{Bpush}}, \eta_{\text{Bpush}} \rangle (Z, a) = \{B \downarrow a,\ Z \downarrow a\}$$

  - *Push at the beginning of a chain of the factorization.*

$$\langle \delta_{\text{Bpush}}, \eta_{\text{Bpush}} \rangle (B, a) = \{T \downarrow \varepsilon\}$$

$$\text{where } T = \left\{ \langle q, p, v \rangle \mid q \in Q, p \in \delta_{\text{push}}(q, a), v = 1 \text{ iff } p \in F \right\}$$

– *Push within a chain of the factorization.*

$$\langle \delta_{\text{Bpush}}, \eta_{\text{Bpush}} \rangle (T, a) = \{S \downarrow \varepsilon\} \quad \text{where}$$

$$S = \left\{ \langle q, p, \nu \rangle \mid \exists \langle r, q, \xi \rangle \in T \text{ s.t. } \nu = \left[\begin{array}{ll} \xi & \text{if } p \notin F \\ 1 & \text{if } p \in F \end{array} \right., \; p \in \delta_{\text{push}}(q, a) \right\}$$

The shift pair $\langle \delta_{\text{Bshift}}, \eta_{\text{Bshift}} \rangle : Q_B \times \Sigma \to \wp_F(Q_B \times PR^*)$ is defined as follows:

– *Pending letter.*

$$\langle \delta_{\text{Bshift}}, \eta_{\text{Bshift}} \rangle (Z, a) = \{B \downarrow a, \; Z \downarrow a\}$$

– *Shift move within a chain of the factorization.*

$$\langle \delta_{\text{Bshift}}, \eta_{\text{Bshift}} \rangle (T, a) = \{S \downarrow \varepsilon\} \quad \text{where}$$

$$S = \left\{ \langle r, p, \nu \rangle \mid \exists \langle r, q, \xi \rangle \in T \text{ s.t. } \nu = \left[\begin{array}{ll} \xi & \text{if } p \notin F \\ 1 & \text{if } p \in F \end{array} \right., \; p \in \delta_{\text{shift}}(q, a) \right\}$$

The pop pair $\langle \delta_{\text{Bpop}}, \eta_{\text{Bpop}} \rangle : Q_B \times Q_B \to \wp_F(Q_B \times PR^*)$ is defined as follows.

– *Pop at the end of a chain of the factorization.*

$$\langle \delta_{\text{Bpop}}, \eta_{\text{Bpop}} \rangle (T, B)) = \{B \downarrow R, \; Z \downarrow R\} \quad \text{where}$$

$$R = \left\{ \langle r, p, \nu \rangle \mid \exists \langle r, q, \xi \rangle \in T \text{ s.t. } p \in \delta_{\text{pop}}(q, r), \nu = \left[\begin{array}{ll} \xi & \text{if } p \notin F \\ 1 & \text{if } p \in F \end{array} \right. \right\}$$

– *Pop within a chain of the factorization*[11].

$$\langle \delta_{\text{Bpop}}, \eta_{\text{Bpop}} \rangle (T, S)) = \{R \downarrow \varepsilon\} \quad \text{where}$$

$$R = \left\{ \langle t, p, \nu \rangle \mid \exists \langle r, q, \xi \rangle \in T, \exists \langle t, r, \zeta \rangle \in S \text{ s.t. } p \in \delta_{\text{pop}}(q, r), \right.$$
$$\left. \nu = \left[\begin{array}{ll} \xi & \text{if } p \notin F \\ 1 & \text{if } p \in F \end{array} \right. \right\}$$

An error state is reached in any other case.

We conclude the construction by computing the size of the resulting automaton, which is an "online composition" of $\mathcal{B}$ and $\mathcal{B}_R$. The Büchi finite-state automaton $\mathcal{A}_R$ has $O(|\Sigma|s)$ states and hence the automaton $\mathcal{B}_R$ has $2^{O(|\Sigma|s \log |\Sigma| s)}$ states [40, 36]; whereas the transducer $\mathcal{B}$ has $|Q_B| = 2^{O(s^2)}$ states. Thus the ωOPBA has $2^{O(s^2 + |\Sigma| s \log |\Sigma| s)}$ states.

To prove that $\mathcal{B}$ produces all $\mathcal{A}$'s pseudoruns –whether accepting or not– observe, first, that its guess about reading a pending letter or the beginning of a chain belonging to the unique factorization defined above, or reading a symbol within such a chain, is essentially the same as the one described in the proof of Lemma II.3.3, where the recognition of a maximal chain is replaced by the recognition of a chain with no prefixes that are chains; thus, wrong guesses are resolved at the time of a pop move (e.g., a pop move is not defined on a first state

[11] Remember that we consider only chains having no prefixes that are chains.

of type Z). Furthermore, pending letters, when correctly guessed as such, are output as soon as they are read (the incorrectly guessed ones belong to runs that will be aborted); elements of $\mathcal{T}$ are output only when a chain of the factorization is recognized, i.e., the transition is defined on a pair of states whose second component is B, which separates these moves from the pop ones occurring within a chain of the factorization; the set T output during the move records all pairs of states that can be the beginning and the end of a support of the recognized chain. Finally the input string is accepted iff infinitely many times either pending letters are read or chains of the factorization are recognized, or both facts occur, i.e., the string is compatible with the OPM, and the produced output is the pseudorun associated with the input by definition, independently on whether the original $\mathcal{A}$'s run was accepting, i.e., infinitely many times sets of triples with $v = 1$ have been output, or not. ∎

Let us finally consider the case of concatenation between a finite length OPL and a language in $\mathcal{L}(\omega\text{OPBA})$. For classical families of automata (on finite or infinite length words) the closure with respect to concatenation is traditionally proved by building an automaton which simulates the moves of the first automaton while reading the first word of the concatenation and –whether deterministically or not– once it reaches some final state, it switches to the initial states of the second one. This natural approach has already been proved ineffective for OPLs in the case of finite-length words since the structure of two concatenated strings is not necessarily the concatenation of the two structures, so that the actions of the second automaton cannot be independent from those of the previous one ([18] provides a constructive proof of the closure of finite-length OPLs w.r.t. concatenation in terms of generating grammars); in fact the lack of the # delimiter between the two strings prevents the typical look-ahead mechanisms which drives the operator-based parsing; thus, the stack cannot be emptied by the normal sequence of pop moves before beginning the parse of the new string. In the case of ω-languages the difficulty is further exacerbated by the fact the automaton might never be able to empty the stack, as e.g., in the case of a language $L_1 \subseteq \{a,b\}^*$ with $a \lessdot a$, $b \lessdot a$, concatenated with $L_2 = \{a^\omega\}$. Notice also that, after reading the first finite word in the concatenation, it would not be possible to determine whether this word *might* be accepted by –possibly nondeterministically– guessing the position of a potential delimiter #, since this check would require to know the states already reached and piled on the stack, which are not visible without emptying the stack itself.

To overcome the above difficulties we follow this approach:

- We give up deterministic parsing. In fact the different computational power between deterministic and nondeterministic automata is a distinguishing property when moving from finite to infinite length languages. Thus, we nondeterministically guess the point of separation between the first finite word and the second infinite one.
- To afford the second major problem, i.e., the lack of enough knowledge to decide whether the guessed first word would be accepted by the corresponding automaton, we use $\not\#$OPAs introduced in Section II.3.1.

The following theorem exploits the above approach. Its proof differs significantly from the non-trivial proof of closure under concatenation of OPLs of finite-length words [18], which, instead, can be recognized deterministically.

Theorem II.3.8 ($\mathcal{L}(\omega\text{OPBA})$ is closed under concatenation). *Let $L_1 \subseteq \Sigma^*$ be a language of finite words recognized by an OPA with OPM M_1 and s_1 states. Let $L_2 \subseteq \Sigma^\omega$ be an ω-language recognized by a nondeterministic ωOPBA with OPM M_2 compatible with M_1 and s_2 states. Then the concatenation $L_1 \cdot L_2$ is also recognized by an ωOPBA with OPM $M \supseteq M_1 \cup M_2$ and $O(s_1^2 + s_2^2)$ states.*

Proof. Let $\mathcal{A}_1$ be a nondeterministic OPA on (Σ, M_1) that recognizes L_1 and let $\mathcal{A}_2 =$

$\langle \Sigma, M_2, Q_2, I_2, F_2, \delta_2 \rangle$ be a nondeterministic ωOPBA with OPM M_2 compatible with M_1 that accepts L_2. Suppose, without loss of generality, that the sets of states of $\mathcal{A}_1$ and $\mathcal{A}_2$ are disjoint.

To define an automaton ωOPBA $\mathcal{A}$ that accepts $L_1 \cdot L_2$, we first build a $\not\#$OPA $\mathcal{A}'_1 = \langle \Sigma, M_1, Q_1, I_1, F_1, \delta_1 \rangle$ such that $L_{\not\#}(\mathcal{A}'_1) = L(\mathcal{A}_1)$.

The automaton $\mathcal{A}$ can recognize the first finite words in the concatenation $L_1 \cdot L_2$ by simulating $\mathcal{A}'_1$: reading the input string, if $\mathcal{A}'_1$ reaches a final state at the end of a finite-length prefix, then it belongs to L_1 and $\mathcal{A}$ immediately starts the recognition of the second infinite string without the need to perform any pop move to empty the stack. From this point onwards, then, $\mathcal{A}$ checks that the remaining infinite portion of the input belongs to L_2, behaving as the ωOPBA $\mathcal{A}_2$.

The strings belonging to the concatenation of two OPLs, however, may contain new chains that span over the two concatenated words. Consider, for instance, the concatenation of $L_1 = \{a^m b^n \mid m \geq n \geq 1\}$ with $L_2 = \{c^+ b^\omega\}$; notice that any OPA recognizing L_1 must be defined on an OPM such that $a \lessdot a$, $a \doteq b$, $b \gtrdot b$ to be able to compare the occurrences of a with those of b; assume also the further precedence relations $a \lessdot c$, $c \lessdot c$, $c \gtrdot b$ (such relations could be mandated, e.g., by other components of either language not included here for simplicity). An automaton recognizing $L_1 \cdot L_2$ can deterministically find the borderline between words $x \in L_1$, and $y \in L_2$; after finishing reading x it will have on its stack $m - n$ remaining as; however, since $a \lessdot c$ it cannot empty the stack and must push all cs on top of the as. Only when receiving the first b, it will pop all cs until the top of the stack will store an a. Since $a \doteq b$, and $b \gtrdot b$ the next action must consist in shifting the b by replacing the topmost a and then popping it, thus consuming part of the stack left by the analysis of x; in other words, it must produce the support of a chain ${}^a[ab]^b$, whose left part belongs to L_1 and whose right part belongs to L_2.

Therefore, $\mathcal{A}$ cannot merely read the second infinite word performing the same transitions as $\mathcal{A}_2$, but it can still simulate this ωOPBA by keeping in the states some summary information about its runs. In this way, while reading the second word in the concatenation, whenever $\mathcal{A}$ has to reduce a chain that extends to the previous word in L_1 and, therefore, must perform a pop move of a symbol in the portion of the stack piled up during the parsing of the first word, it can restore the state that $\mathcal{A}_2$ would instead have reached, resuming therefrom as in a run of $\mathcal{A}_2$.

Precisely, $\mathcal{A}$ is defined as the tuple $\langle \Sigma, M, Q, I, F, \delta \rangle$ where:

- $M \supseteq M_1 \cup M_2$ and may be supposed to be a complete matrix, for instance assigning arbitrary precedence relations to the empty entries, so that the strings in the concatenation of languages L_1 and L_2 are compatible with M.
- $Q = Q_1 \cup Q_2 \cup Q_2 \times (Q_2 \cup \{-\})$, i.e. the set of states of $\mathcal{A}$ includes the states of $\mathcal{A}'_1$ and $\mathcal{A}_2$, along with the states of $\mathcal{A}_2$ extended with a second component. The first component is the state of Q_2 that $\mathcal{A}_2$ would reach in its corresponding computation on the second word of the concatenation, and the second one represents the state of the symbol that is on the top of the stack when the current input letter is read in this run of $\mathcal{A}_2$. Storing this component is necessary to guarantee that, whenever the automaton $\mathcal{A}$ has to perform a pop move that removes symbols that have been piled on the stack during the recognition of the first word in the concatenation, it is still possible to compute the state that $\mathcal{A}_2$ would have reached instead.
  This second component is denoted $'-'$ if all the preceding symbols in the stack have been piled up during the parsing of the first word of the concatenation (thus the stack of $\mathcal{A}_2$ is empty).
- $I = I_1 \cup \{\langle q_0, - \rangle \mid q_0 \in I_2\}$ if $\varepsilon \in L_1$; $I = I_1$ otherwise

- $F = F_2 \cup F_2 \times (Q_2 \cup \{-\})$
- The transition function $\delta : Q \times (\Sigma \cup Q) \to \wp(Q)$ is $\delta = \delta_1 \cup \delta_2 \cup \delta^{\text{join}}$ and is defined as the union of three functions: the transition functions of $\mathcal{A}'_1$ and $\mathcal{A}_2$ by which it simulates the first automaton on the first word of the concatenation and the second automaton on the second one, and a function δ^{join} that handles the nondeterministic transition from the simulation of the first automaton to the second one and the parsing of the suffix (within the second word of the concatenation) of the chains that span over the two words.
  Function δ^{join} is defined as follows: let $c \in \Sigma$, $p \in Q_1, q, q_1, q_2, q_3 \in Q_2$, $r \in (Q_2 \cup \{-\})$.
  The push transition function $\delta^{\text{join}}_{\text{push}} : Q \times \Sigma \to \wp(Q)$ is defined by:
  - $\delta^{\text{join}}_{\text{push}}(p, c) = \{\langle q_0, - \rangle \mid q_0 \in I_2, \text{ if } \exists p_f \in F_1 \text{ s.t. } \delta_{1\text{push}}(p, c) \ni p_f\}$,
    i.e., $\mathcal{A}$ nondeterministically enters the initial states of $\mathcal{A}_2$ after the recognition of a word in L_1
  - $\delta^{\text{join}}_{\text{push}}(\langle q, r \rangle, c) = \delta_{2\text{push}}(q, c)$,
    i.e., $\mathcal{A}$ simulates a push move of $\mathcal{A}_2$, reaching a state in Q_2, whenever it starts to recognize a chain in the second word of the concatenation (which thus does not extend to the first word).

  The shift transition function $\delta^{\text{join}}_{\text{shift}} : Q \times \Sigma \to \wp(Q)$ is defined by:
  - $\delta^{\text{join}}_{\text{shift}}(p, c) = \{\langle q_0, - \rangle \mid q_0 \in I_2, \text{ if } \exists p_f \in F_1 \text{ s.t. } \delta_{1\text{shift}}(p, c) \ni p_f\}$,
    i.e., $\mathcal{A}$ nondeterministically enters the initial states of $\mathcal{A}_2$ after the recognition of a word in L_1
  - $\delta^{\text{join}}_{\text{shift}}(\langle q_1, - \rangle, c) = \{\langle q_2, q_1 \rangle \mid q_2 \in \delta_{2\text{push}}(q_1, c)\}$,
    i.e., $\mathcal{A}$ simulates the push move induced by the precedence relation $\# \lessdot c$ that, in the corresponding run of $\mathcal{A}_2$, starts the recognition of a chain that is a prefix of the second word of the concatenation
  - $\delta^{\text{join}}_{\text{shift}}(\langle q_1, q_2 \rangle, c) = \{\langle q_3, q_2 \rangle \mid q_3 \in \delta_{2\text{shift}}(q_1, c)\}$,
    i.e., $\mathcal{A}$ performs a shift move within a chain that spans over the two words of the concatenation.

  The pop transition function $\delta^{\text{join}}_{\text{pop}} : Q \times Q \to \wp(Q)$ is defined by:
  - $\delta^{\text{join}}_{\text{pop}}(\langle q, - \rangle, p) = \langle q, - \rangle$,
    i.e, $\mathcal{A}$ concludes to recognize a chain, at the end of the first word of the concatenation, induced by the precedence relations with the letters of the second string, and consumes the corresponding stack symbols piled while reading the first word
  - $\delta^{\text{join}}_{\text{pop}}(\langle q_1, q_2 \rangle, p) = \{\langle q_3, - \rangle \mid q_3 \in \delta_{2\text{pop}}(q_1, q_2)\}$,
    i.e., whenever the precedence relations induce a merge of the chains of the words of the concatenation, $\mathcal{A}$ restores the state q_3 of $\mathcal{A}_2$ from which a run of $\mathcal{A}_2$ will continue
  - $\delta^{\text{join}}_{\text{pop}}(q_1, \langle q_2, r \rangle) = \{\langle q_3, r \rangle \mid q_3 \in \delta_{2\text{pop}}(q_1, q_2)\}$,
    i.e., $\mathcal{A}$ completes the recognition of a chain that belongs to a composed chain spanning over the two words of the concatenation.

One can verify that, after having simulated $\mathcal{A}'_1$ and nondeterministically guessed the end of a word in L_1, $\mathcal{A}$ proceeds with the simulation of $\mathcal{A}_2$ and accepts the remaining ω-string iff it belongs to L_2. In fact, the projection on the first component of the states visited along $\mathcal{A}$'s run on the second word of the concatenation identifies a successful run of $\mathcal{A}_2$ on the same word. ∎

To summarize, Table II.3.2 displays the complexities of the various constructions to obtain the closure w.r.t. Boolean operations and concatenation; it also compares them with the corresponding complexities for VPLs showing that the only main difference occurs in the case of concatenation.

	$\mathcal{L}(\omega\text{OPBA})$	$\mathcal{L}(\omega\text{BVPA})$
$L_1 \cap L_2$	$O(s_1 s_2)$	$O(s_1 s_2)$
$L_1 \cup L_2$	$O(\|\Sigma\|^2(s_1 + s_2))$	$s_1 + s_2$
$\neg L_1$	$2^{O(s_1^2+\|\Sigma\|s_1 \log\|\Sigma\|s_1)}$	$2^{O(s_1^2)}$
$L_3 \cdot L_1$	$O(s_1^2 + s_3^2)$	$s_1 + s_3$

Table II.3.2: Size of state sets of languages recognizing $L_1 \cap L_2$, $L_1 \cup L_2$, $\neg L_1$ and $L_3 \cdot L_1$. The results on ωOPBAs have been proved, respectively, in Theorem II.3.5, Theorem II.3.6, Theorem II.3.7 and Theorem II.3.8. The complexity results on ωBVPAs derive from the constructions and proofs of their closure properties shown in [4].

II.3.3. Closure properties of the other classes of ωOPLs. The class of languages recognized by ωDOPMAs is a Boolean algebra. The other classes are closed only under union and intersection.

Theorem II.3.9 ($\mathcal{L}(\omega$DOPMA) is a Boolean algebra). *Let L_1 and L_2 be ω-languages that are recognized by two ωDOPMAs defined over the same alphabet Σ, with compatible precedence matrices M_1 and M_2 and s_1 and s_2 states respectively. Then $L_1 \cap L_2$ (resp. the complement of L_1 w.r.t. L_M, or $L_1 \cup L_2$) is recognized by an ωDOPMA with OPM $M = M_1 \cap M_2$ and $s_1 s_2$ (resp. s_1, or $|\Sigma|^4 s_1 s_2$) states.*

Proof. Let $\mathcal{A}_1 = \langle \Sigma, M_1, Q_1, q_{01}, \mathcal{T}_1, \delta_1 \rangle$ and $\mathcal{A}_2 = \langle \Sigma, M_2, Q_2, q_{02}, \mathcal{T}_2, \delta_2 \rangle$ be ωDOPMAs recognizing languages L_1 and L_2. Assume without loss of generality that their transition function is total (otherwise, it can be naturally completed once the set of states is extended with an "error" state).

An ωDOPMA $\mathcal{A}$ with OPM $M = M_1 \cap M_2$ recognizing $L = L_1 \cap L_2$ may be defined adopting the usual product construction for ω-regular automata, requiring that a successful path in $\mathcal{A}$ corresponds to paths that visit infinitely often sets in the table $\mathcal{T}_1$ and $\mathcal{T}_2$. More precisely let $\mathcal{A} = \langle \Sigma, M, Q, q_0, \mathcal{T}, \delta \rangle$ where

- $Q = Q_1 \times Q_2$,
- $q_0 = (q_{01}, q_{02})$,
- Define π_i (i = 1, 2) as the projection from $Q_1 \times Q_2$ on Q_i, that can also be naturally extended to define projections on paths of the automata, and let
  $\mathcal{T} = \{P \subseteq Q_1 \times Q_2 \mid \pi_1(P) \in \mathcal{T}_1 \wedge \pi_2(P) \in \mathcal{T}_2\}$,
- The transition function δ is the product of δ_1 and δ_2 (see Definition II.3.1).

Let ρ be a successful path of $\mathcal{A}$, starting in the initial state $q_0 = (q_{01}, q_{02})$: since it is accepting, the set $Inf(\rho) = P \in \mathcal{T}$. By definition of $\mathcal{T}$, the paths ρ_1 and ρ_2 that are the projection of ρ on the set of states of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, have $Inf(\rho_1) = \pi_1(P) \in \mathcal{T}_1$ and $Inf(\rho_2) = \pi_2(P) \in \mathcal{T}_2$: hence ρ_1 and ρ_2 are successful paths for the two automata, and x belongs to $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.

Let now $x \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$; thus, x labels two successful paths ρ_1 and ρ_2 of the two automata, i.e., $Inf(\rho_1) \in \mathcal{T}_1$ and $Inf(\rho_2) \in \mathcal{T}_2$. The path ρ of $\mathcal{A}$ which visits the pairs of states of the two automata, performing the same type of move they perform for each input symbol, is defined so as $\pi_1(Inf(\rho)) = Inf(\rho_1) \in \mathcal{T}_1$ and $\pi_2(Inf(\rho)) = Inf(\rho_2) \in \mathcal{T}_2$. Therefore, by definition of $\mathcal{T}$, ρ is a successful path for $\mathcal{A}$.

To recognize the complement of L_1, given that $\mathcal{A}$ is deterministic and its transition function is total, it is clearly sufficient to build the ωDOPMA $\mathcal{A}' = \langle \Sigma, M_1, Q_1, q_{01}, \wp(Q_1) \setminus \mathcal{T}_1, \delta \rangle$ whose table is the complement of $\mathcal{T}_1$ w.rt. $\wp(Q_1)$.

To obtain the closure w.r.t. union, we can assume that $M_1 = M_2$ w.l.o.g. (otherwise one can apply Statement II.3.1, increasing the number of states of each automaton of a factor $|\Sigma|^2$) and apply De Morgan's law. The number of states of the resulting automaton is $|Q_1| \cdot |Q_2|$ and this concludes the proof, recalling the factor $|\Sigma|^2 \cdot |\Sigma|^2$ implied by the possible aplication of Statement II.3.1. Notice that, if one considers automata with compatible but not equal matrices, De Morgan's law could not be applied: in fact, the equality

$$L_1 \cup L_2 = L_{M_1 \cap M_2} \setminus [(L_{M_1} \setminus L_1) \cap (L_{M_2} \setminus L_2)]$$

does not hold, unless $M_1 = M_2$. ☐

Proposition II.3.10. *Let L_1 and L_2 be ω-languages recognized by two ωOPBEA (resp. ωDOPBA, ωDOPBEA) defined over the same alphabet Σ, with compatible precedence matrices M_1 and M_2 and with s_1 and s_2 states respectively. Then $L = L_1 \cap L_2$ is recognized by an ωOPBEA (resp. ωDOPBA, ωDOPBEA) with OPM $M = M_1 \cap M_2$ and $O(s_1 s_2)$ states.*

Proof. For ωOPBEA, we can assume without loss of generality that the the automaton is in normal form with partitioned sets of states, (see Definition II.1.5), and apply the same construction as for ωOPBA (see Theorem II.3.5). The use of automata with partitioned sets of states guarantees that a run of $\mathcal{A}$ on an ω-word reaches infinitely often a final state with empty stack iff both $\mathcal{A}_1$ and $\mathcal{A}_2$ have a run for the word which traverses infinitely often a final state with empty stack.

For ωDOPBA and ωDOPBEA, the proof derives from the fact that, if $\mathcal{A}_1$ and $\mathcal{A}_2$ are deterministic, then the resulting intersection automaton is deterministic too. ☐

Proposition II.3.11. *Let L_1 and L_2 be ω-languages recognized by two ωOPBEA (resp. ωDOPBA, ωDOPBEA) defined over the same alphabet Σ, with compatible precedence matrices M_1 and M_2 and s_1 and s_2 states respectively. Then $L = L_1 \cup L_2$ is recognized by an ωOPBEA (resp. ωDOPBA, ωDOPBEA) with OPM $M = M_1 \cup M_2$ and $O(|\Sigma|^2 s_1 s_2))$ (resp. $O(|\Sigma|^4 s_1 s_2))$ states.*

Proof. The proof for ωOPBEA is analogous to the proof of closure under union for ωOPBA(see Theorem II.3.5).

For the determistic models, the construction must be refined. Let A_1 and A_2 be ωDOPBA accepting L_1 and L_2 over OPMs M_1 and M_2, respectively. As usual, we assume that that both transition functions are complete and $M_1 = M_2$ (otherwise one can apply Statement II.3.1, increasing the number of states of a factor $|\Sigma|^2$). Let $A_i = \langle \Sigma, M, Q_i, q_{0i}, F_i, \delta_i \rangle$, for $i = 1, 2$. An ωDOPBA (resp. ωDOPBEA) $\mathcal{A}_3$ which recognizes $L_1 \cup L_2$ is then defined by adopting the usual product construction for regular automata: $\mathcal{A}_3 = \langle \Sigma, M, Q_3, q_{03}, F_3, \delta_3 \rangle$ where:

- $Q_3 = Q_1 \times Q_2$,
- $q_{03} = (q_{01}, q_{02})$,
- $F_3 = F_1 \times Q_2 \cup Q_1 \times F_2$
- and the transition function is the product of δ_1 and δ_2.

The number of states of $\mathcal{A}_3$ is given by the product $|Q_1| \cdot |Q_2|$ and this concludes the proof, recalling the factor $|\Sigma|^2 \cdot |\Sigma|^2$ implied by Statement II.3.1. ☐

Theorem II.3.12 (ωDOPBA, ωOPBEA, ωDOPBEA are not closed under complement). *Let L be an ω-language accepted by an ωDOPBA (resp. ωOPBEA, or ωDOPBEA) with*

OPM M on alphabet Σ. There does not necessarily exist an ωOPBEA (resp. ωOPBEA, or ωDOPBEA) recognizing the complement of L w.r.t. L_M.

Proof. Language $L_{a\infty}$ can be recognized by an ωDOPBA with an OPM M (shown, for instance, in Figure II.2.2), but there's no ωDOPBA that can recognize the complement of this language w.r.t. L_M, i.e. the language $L_{a-\text{finite}}$, as mentioned in Section II.2.2. The same argument on $L_{a\infty}$ holds also for ωDOPBEAs.

Finally, as regards ωOPBEAs, L^{ω}_{abseq} is recognized by the ωOPBEA with OPM M and state graph presented in Section II.2.2. However, no ωOPBEA can recognize the complement of this language w.r.t. L_M. Such an ωOPBEA, in fact, should have OPM M so that no word in L^{ω}_{abseq} can be accepted. The precedence relation $M_{aa} = \{\lessdot\}$ (which is necessary to verify that in a sequence of type $(a^k b^h)^{\omega}$ there is at least one substring with $k \neq h$), however, prevents an ωOPBEA from accepting the word a^{ω}, which belongs to the complement of L^{ω}_{abseq} w.r.t. L_M, since it implies that, while reading the word, the ωOPBEA can never reach a state with empty stack. ☐

Theorem II.3.13 (ωDOPBA, ωOPBEA, ωDOPBEA, and ωDOPMA are not closed under concatenation). *Let L_2 be an ω-language accepted by an ωOPBEA (resp. ωDOPBA, ωDOPBEA, or ωDOPMA) with OPM M on alphabet Σ and let $L_1 \subseteq \Sigma^*$ be a language (of finite words) recognized by an OPA with a compatible precedence matrix. The ω-language defined by the concatenation $L_1 \cdot L_2$ is not necessarily recognizable by an ωOPBEA (resp. ωDOPBA, ωDOPBEA, or ωDOPMA).*

Proof. For ωDOPBAs, let $\Sigma = \{a, b\}$ and consider the language $L_{a-\text{finite}}$, which can be seen as the concatenation $L_{a-\text{finite}} = L_1 \cdot L_2$ of a language of finite words $L_1 = \{a, b\}^*$, which can be clearly recognized by an OPA, and an ω-language $L_2 = \{b^{\omega}\}$, which can be recognized by an ωDOPBA, with compatible precedence matrices. Since language $L_{a-\text{finite}}$ cannot be recognized by an ωDOPBA, then the class of languages $\mathcal{L}(\omega\text{DOPBA})$ is not closed w.r.t. concatenation.

Given $\Sigma = \{c, r\}$, the language L_{repbsd} cannot be recognized by an ωOPBEA (respectively ωDOPBEA, or ωDOPMA), as shown in Section II.2.2. Consider the OPA that accepts the language $L_1 = \Sigma^*$ of words of finite length whose OPM is the same as the precedence matrix depicted in Figure II.2.3. These words have necessarily a finite number of pending calls, since they have finite length. Moreover, let $\mathcal{A}_2$ be an ωOPBEA (respectively ωDOPBEA, or ωDOPMA) that recognizes the ω-language $L_{\omega\text{Dyck-pr}(c,r)}$ and which is depicted in Figure II.2.3. The concatenation ω-language $L_1 \cdot L_{\omega\text{Dyck-pr}(c,r)}$ is exactly the set of ω-words with a finite number of pending calls, i.e. L_{repbsd}. Hence, the class of languages $\mathcal{L}(\omega\text{OPBEA})$ (respectively $\mathcal{L}(\omega\text{DOPBEA})$, or $\mathcal{L}(\omega\text{DOPMA})$) is not closed w.r.t. concatenation. ☐

II.4. Monadic Second-order Logic characterization of ωOPLs. We finally provide a characterization of ωOPLs in terms of a MSO logic which is interpreted over infinite words. As usual, we focus our attention on $\mathcal{L}(\omega\text{OPBA})$, the most general class of ωOPLs.

We adopt the same conventions and notations as in Section I.4, and extend the formula evaluation over ω-strings in the natural way. To distinguish the infinite case from the finite one, we will use symbol $\models_{\omega}$ instead of $\models$. Given an OP alphabet (Σ, M) and a MSO formula φ, we denote the language of all strings $w \in \Sigma^{\omega}$ such that $\#w \models_{\omega} \varphi$ by $L^{\omega}(\varphi) = \{w \in \Sigma^{\omega} \mid \#w \models_{\omega} \varphi\}$.

Example 10 (Managing interrupts). *Consider again the system that manages interrupts described in Example 8. The same rules enforced by the automaton of Figure II.1.2 are also formalized by the following sentences.*

- *All int_2 are eventually served by a corresponding $serve_2$:*

$$\forall \boldsymbol{x}\ (int_2(\boldsymbol{x}) \Rightarrow \exists \boldsymbol{y}(serve_2(\boldsymbol{y}) \wedge (\boldsymbol{y} = \boldsymbol{x} + 1 \vee \boldsymbol{x} \curvearrowright \boldsymbol{y}))).$$

- *Lower priority interrupts are not accepted when a higher priority one is pending:*

$$\forall \boldsymbol{x},\boldsymbol{y}\ (int_2(\boldsymbol{x}) \wedge serve_2(\boldsymbol{y}) \wedge \boldsymbol{x} \curvearrowright \boldsymbol{y} \Rightarrow \forall \boldsymbol{k}(\boldsymbol{x} < \boldsymbol{k} < \boldsymbol{y} \Rightarrow \neg int_1(\boldsymbol{k}))).$$

As another example consider the "weak fairness requirement" also mentioned in Example 8 which states that after a first $call_a$ not matched by ret_a but interrupted by a int_1 or int_2, a second $call_a$ cannot be interrupted by a new lower priority interrupt int_1 (but can still be interrupted at any time by higher priority ones): the sentence below formalizes such a constraint.

$$\begin{aligned}\neg\exists \boldsymbol{x}_1,\boldsymbol{x}_2(&\boldsymbol{x}_1 < \boldsymbol{x}_2 \wedge call_a(\boldsymbol{x}_1) \wedge call_a(\boldsymbol{x}_2)\wedge\\ &\forall \boldsymbol{x}_3(\boldsymbol{x}_1 \le \boldsymbol{x}_3 \le \boldsymbol{x}_2 \wedge call_a(\boldsymbol{x}_3) \Rightarrow \neg\exists \boldsymbol{y}_3(ret_a(\boldsymbol{y}_3) \wedge (\boldsymbol{y}_3 = \boldsymbol{x}_3 + 1 \vee \boldsymbol{x}_3 \curvearrowright \boldsymbol{y}_3)))\wedge\\ &\exists z_1,z_2((int_1(z_1) \vee int_2(z_1)) \wedge int_1(z_2) \wedge \bigwedge_{i=1}^{2}(z_i = \boldsymbol{x}_i + 1 \vee \boldsymbol{x}_i \curvearrowright z_i)))\end{aligned}$$

Theorem II.4.1. *Let (Σ, M) be an OP alphabet. L is accepted by a nondeterministic ωOPBA $\mathcal{A}$ over (Σ, M) if and only if there exists an MSO sentence φ such that $L = L^\omega(\varphi)$.*

The construction of a nondeterministic ωOPBA equivalent to an MSO formula is identical to the one given for finite strings.

The converse construction also follows essentially the same path as in the case of finite-length languages; thus, we only point out the few relevant differences w.r.t. the construction of Section I.4. Formula φ is defined as

$$\varphi := \begin{matrix}\exists \boldsymbol{A}_0,\boldsymbol{A}_1,\dots,\boldsymbol{A}_N\\ \exists \boldsymbol{B}_0,\boldsymbol{B}_1,\dots,\boldsymbol{B}_N\\ \exists \boldsymbol{C}_0,\boldsymbol{C}_1,\dots,\boldsymbol{C}_N\end{matrix}\left(\bigvee_{q_i\in I}\mathrm{Start}_i \ \wedge\ \varphi_\delta \ \wedge\ \varphi_{unique} \ \wedge\ \bigvee_{q_f\in F}\mathrm{Accept}_f\right), \tag{II.4.1}$$

where Start_i si defined as in Section I.4, and Accept_f is a shortcut representing the Büchi acceptance condition (a final state is reached infinitely often):

$$\mathrm{Accept}_f := \forall \boldsymbol{x}\exists \boldsymbol{y}(\boldsymbol{x} < \boldsymbol{y} \wedge \boldsymbol{y} \in Q_f).$$

Formula φ_δ encodes the nondeterministic transition functions of the automaton and is obtained from formula $\varphi_{\delta_{\text{push}}} \wedge \varphi_{\delta_{\text{shift}}} \wedge \varphi_{\delta_{\text{pop}}}$ defined in Section I.4, by replacing expressions as $q_k = \delta(\dots)$ by expressions as $q_k \in \delta(\dots)$. Finally, formula φ_{unique} is defined as the conjunction of the following formulae:

$$\varphi_{uniqueA} := \forall \boldsymbol{x}\bigwedge_{i=0}^{N}\left(\boldsymbol{x} \in \boldsymbol{A}_i \Rightarrow \neg\bigvee_{j=0}^{N}(j \neq i \wedge \boldsymbol{x} \in \boldsymbol{A}_j)\right)$$

$$\varphi_{unique\_next} := \forall \boldsymbol{x},\boldsymbol{y}\bigwedge_{k=0}^{N}\left(\mathrm{Next}_k(\boldsymbol{x},\boldsymbol{y}) \Rightarrow \neg\bigvee_{j\neq k}\mathrm{Next}_j(\boldsymbol{x},\boldsymbol{y})\right)$$

Such formula was not necessary in the finite case because it was implied by the determinism of the automaton.

The proof that formula φ is satisfied by all and only the words accepted by $\mathcal{A}$ is again based on Lemmata I.4.3 and I.4.4, but we need some more properties to cope with infinite words.

Any ω-word $w \in \Sigma^\omega$ compatible with M can be factored, as in the proof of Theorem II.3.7, as a sequence $w = w_1w_2w_3\dots$ where either $w_i \in \Sigma$ is a pending letter, or w_i is the body of the chain ${}^{a_i}[w_i]^{b_i}$, where a_i is the last pending letter before w_i and b_i is the first symbol of w_{i+1}. A similar factorization holds for a finite word $\#w$ without end delimiter. We denote by $\boldsymbol{P}$ the set of positions in a (finite or infinite) string w that correspond to pending letters and by $\boldsymbol{E}$ the set of positions of the right delimiter of the chains of the factorization. These two sets are not necessarily disjoint, and $\boldsymbol{EP}$ is their union.

$$\begin{aligned} z \in \boldsymbol{P} &:= \forall \boldsymbol{x},\boldsymbol{y}\ (\boldsymbol{x} < z < \boldsymbol{y} \wedge \boldsymbol{x} \curvearrowright \boldsymbol{y} \Rightarrow \#(\boldsymbol{y})) \\ z \in \boldsymbol{E} &:= \exists \boldsymbol{x}\ (\boldsymbol{x} \in \boldsymbol{P} \wedge \boldsymbol{x} \curvearrowright z) \\ z \in \boldsymbol{EP} &:= z \in \boldsymbol{P} \vee z \in \boldsymbol{E} \end{aligned}$$

Any prefix of an infinite string w which ends in an EP position of w is called *EP-prefix* of w.

Let us define

$$\psi_{i,k}(\boldsymbol{A}_0,\dots,\boldsymbol{A}_N,\boldsymbol{B}_0,\dots\boldsymbol{B}_N,\boldsymbol{C}_0,\dots,\boldsymbol{C}_N) := \mathrm{Start}_i \wedge \varphi'_\delta \wedge \mathrm{Final}_k$$

where

$$\mathrm{Final}_k := \exists \boldsymbol{y} \exists \boldsymbol{e}\ (\ \boldsymbol{y} \in Q_k\ \wedge\ \boldsymbol{y} \le \boldsymbol{e}\ \wedge\ \boldsymbol{e} \in \boldsymbol{EP}\ \wedge\ \forall z(\boldsymbol{y} \le z\ \wedge\ z \in \boldsymbol{EP} \Rightarrow z = \boldsymbol{e}))$$

and φ'_δ is as φ_δ except for the formula $\varphi_{\delta_{pop}}$, where the constraint $\neg\#(\boldsymbol{y})$ is conjuncted to the antecedent of $\varphi_{\delta_{pop\_fw}}$, and $\varphi_{\delta_{pop\_bwB}}$ and $\varphi_{\delta_{pop\_bwC}}$ are replaced by the unique formula

$$\varphi_{pop\_bw} := \forall \boldsymbol{x},z,\boldsymbol{v},\boldsymbol{y} \bigwedge_{k=0}^{N} \left(\begin{array}{c} \boldsymbol{x} \in \boldsymbol{B}_k \wedge \boldsymbol{v} \in \boldsymbol{C}_k \\ \wedge \\ \neg\#(\boldsymbol{y}) \wedge \mathrm{Tree}(\boldsymbol{x},z,\boldsymbol{v},\boldsymbol{y}) \end{array} \Rightarrow \bigvee_{i=0}^{N} \bigvee_{j=0}^{N} \left(\begin{array}{c} \mathrm{Tree}_{i,j}(\boldsymbol{x},z,\boldsymbol{v},\boldsymbol{y}) \\ \wedge \\ \delta_{\mathrm{pop}}(q_i,q_j) \ni q_k \end{array} \right) \right)$$

We will interpret formula $\psi_{i,k}$ over finite strings. More precisely, let w' be an EP-prefix of a string $w \in \Sigma^\omega$. It is $w \vDash_\omega \varphi$ if and only if there exist an initial state q_i, a final state q_f, and an assignment $\boldsymbol{A}_0,\dots,\boldsymbol{C}_N$ such that $w' \vDash \psi_{i,f}(\boldsymbol{A}_0,\dots,\boldsymbol{C}_N)$ for an infinite number of EP-prefixes w' of w. In this case, a position $\boldsymbol{x}$ in a prefix w' may start a chain that goes beyond the end of w', hence in such cases $\boldsymbol{x}$ is in $\boldsymbol{B}_k$ in the assigment satisfying $w \vDash_\omega \varphi$ but $w' \nvDash_\omega \varphi_{pop\_bwB}$. This is the reason why we replace the backward formulae of $\varphi_{\delta_{\mathrm{pop}}}$ in φ'_δ.

For any assignment for $\boldsymbol{A}_0,\dots,\boldsymbol{C}_N$, it is $w' \vDash \psi_{i,k}(\boldsymbol{A}_0,\dots,\boldsymbol{C}_N)$ if and only if there exists a run of $\mathcal{A}$ for w' beginning from state q_i that visits state q_k somewhere after the last EP position before $|w'|$. The run can be built reasoning as in Lemmata I.4.3 and I.4.4 within the chains of the factorization, and using formulae $\varphi_{\delta_{\mathrm{push}}}$ and $\varphi_{\delta_{\mathrm{shift}}}$ for the positions of pending letters. The properties corresponding to states q_i and q_k are provided by formulae Start_i and Final_k. If w' and w'' are EP-prefixes of w and both satisfy $\psi_{i,k}$ *with the same assignment* to $\boldsymbol{A}_0,\dots,\boldsymbol{C}_N$, then the corresponding runs built with such a construction are one the prefix of the other.

Hence $w \vDash_\omega \varphi$ if and only if there exist infinitely many (finite) runs of $\mathcal{A}$ on EP-prefixes of w, each of them beginning from q_i and visiting the same final state q_f somewhere after its last EP position; such runs are all prefixes of the same infinite run ρ.

Furthermore, since there is a move in ρ that reaches q_f while reading the suffix of each of those EP-prefixes after its last EP position, then ρ traverses infinitely often q_f, and hence ρ is accepting for $\mathcal{A}$.

Symmetrically, one can prove that if there exists an accepting run ρ for an ω-string w in $\mathcal{A}$, then $w \vDash_\omega \varphi$.

II.5. Concluding remarks. In this paper we have supplied a number of results about OPLs which, together with previous recent and less recent ones, qualify OPLs as the largest class of deterministic context-free languages that enjoy all of the following basic properties which have a strong impact on various types of practical applications, spanning from parsing to model checking:

- Local parsability: this property, not pursued in this paper, allows for realizing simple and efficient parallel and/or incremental algorithms [7, 6];
- Closure under all main language operations –Boolean ones, concatenation, Kleene * and others [18];
- Automata-theoretic and Monadic Second Order logic characterization;
- Extension of all above properties to the case of ω-languages, i.e., languages consisting of infinite-length strings, with the noticeable and typical exception of the lack of equivalence between deterministic and nondeterministic automata –under the Büchi acceptance condition.

As for the complexity of the constructions used to prove our results we have shown that they are in general of the same order as those of corresponding constructions for less powerful language families –typically, VPLs; the few cases of different complexity have been pointed out in Table II.3.2.

This fairly complete foundational characterization of OPLs can now ignite –and partially already did– further research along several directions. On the one side we are developing practical tools exploiting the above properties both in parsing and in automatic verification; on the other side we envisage many interesting special cases of OPLs motivated by different possible applications.

For instance, we are investigating the use of logic formalisms simpler than MSO logic to characterize suitable subclasses of general OPLs, in the same vein as it has been done for regular languages [15], VPLs [2] and for various cases of tree-languages [1, 10]; a first result on this respect is that free languages, a subclass of OPLs originally motivated by grammar inference [19, 20] can be defined in terms of a first-order logic rather than a second order one [27].

We are also investigating new, less usual application fields for OPLs, or suitable subclasses thereof, beyond the traditional field of programming languages, e.g., in the direction suggested by Examples 5, 8 and others not reported here which are in the same vein as the application indicated for VPLs but considerably extend its scope.

Acknowledgments. We are very grateful to the anonymous reviewers for their constructive criticism and their thoughtful, precise, and very detailed suggestions to improve the original manuscript.

REFERENCES

[1] L. Afanasiev, P. Blackburn, I. Dimitriou, B. Gaiffe, E. Goris, M.J. Marx, and M. de Rijke, *PDL for ordered trees*, Journal of Applied Non-Classical Logic, 15 (2005), pp. 115–135.

[2] R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, and L. Libkin, *First-order and temporal logics for nested words*, Logical Methods in Computer Science, 4 (2008).

[3] R. Alur and P. Madhusudan, *Visibly Pushdown Languages*, in STOC: ACM Symposium on Theory of Computing (STOC), 2004.

[4] ———, *Adding nesting structure to words*, Journ. ACM, 56 (2009).

[5] B. B. von Braunmuhl and R. Verbeek, *Input-driven languages are recognized in log n space*, in Proceedings of the Symposium on Fundamentals of Computation Theory, Lect. Notes Comput. Sci. 158, Springer, 1983, pp. 40–51.

[6] A. Barenghi, S. Crespi Reghizzi, D. Mandrioli, F. Panella, and M. Pradella, *The PAPAGENO parallel-parser generator*, in 23rd International Conference on Compiler Construction (CC), April 2014.

[7] A. Barenghi, S. Crespi Reghizzi, D. Mandrioli, and M. Pradella, *Parallel parsing of operator precedence grammars*, Information Processing Letters, (2013). DOI:10.1016/j.ipl.2013.01.008.

[8] J. Berstel and L. Boasson, *Balanced Grammars and Their Languages*, in Formal and Natural Computing, W. Brauer et al., ed., vol. 2300 of LNCS, Springer, 2002, pp. 3–25.

[9] L. Boasson and M. Nivat, *Adherences of languages*, Journal of Computer and System Sciences, 20 (1980), pp. 285 – 309.

[10] A. Boral and S. Schmitz, *Model-checking parse trees*, in LICS, 2013.

[11] W. S. Brainerd, *The minimalization of tree automata*, Information and Control, 13 (1968), pp. 484–491.

[12] J. R. Büchi, *On a decision method in restricted second order arithmetic*, in Proceedings of the 1960 International Congress on Logic, Methodology and Philosophy of Science (LMPS'60), Ernest Nagel, Patrick Suppes, and Alfred Tarski, eds., Stanford University Press, 1962, pp. 1–11.

[13] O. Burkart and B. Steffen, *Model checking for context-free processes*, in CONCUR '92, vol. 630 of LNCS, 1992, pp. 123–137.

[14] D. Caucal and S. Hassen, *Synchronization of Grammars*, in CSR, Edward A. Hirsch, Alexander A. Razborov, Alexei L. Semenov, and Anatol Slissenko, eds., vol. 5010 of LNCS, Springer, 2008, pp. 110–121.

[15] C. Choffrut, A. Malcher, C. Mereghetti, and B. Palano, *On the Expressive Power of FO[+]*, in LATA, 2010, pp. 190–201.

[16] ———, *First-order logics: some characterizations and closure properties*, Acta Inf., 49 (2012), pp. 225–248.

[17] E. M. Clarke, E. A. Emerson, and A. P. Sistla, *Automatic verification of finite-state concurrent systems using temporal logic specifications*, ACM Trans. Program. Lang. Syst., 8 (1986), pp. 244–263.

[18] S. Crespi Reghizzi and D. Mandrioli, *Operator Precedence and the Visibly Pushdown Property*, Journal of Computer and System Science, 78 (2012), pp. 1837–1867.

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

[20] S. Crespi Reghizzi, M. A. Melkanoff, and L. Lichten, *The Use of Grammatical Inference for Designing Programming Languages*, Communications of the ACM, 16 (1973), pp. 83–90.

[21] K. De Bosschere, *An Operator Precedence Parser for Standard Prolog Text*, Softw., Pract. Exper., 26 (1996), pp. 763–779.

[22] M. J. Fischer, *Some properties of precedence languages*, in STOC '69: Proc. first annual ACM Symp. on Theory of Computing, New York, NY, USA, 1969, ACM, pp. 181–190.

[23] R. W. Floyd, *Syntactic Analysis and Operator Precedence*, Journ. ACM, 10 (1963), pp. 316–333.

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

[25] M. A. Harrison, *Introduction to Formal Language Theory*, Addison Wesley, Reading, MA, 1978.

[26] Donald E. Knuth, *On the translation of languages from left to rigth*, Information and Control, 8 (1965), pp. 607–639.

[27] V. Lonati, D. Mandrioli, F. Panella, and M. Pradella, *First-order Logic Definability of Free Languages*, in 10th Int. Computer Science Symposium in Russia (CSR), LNCS, 2015, p. To appear.

[28] V. Lonati, D. Mandrioli, and M. Pradella, *Precedence Automata and Languages*, in 6th Int. Computer Science Symposium in Russia (CSR), vol. 6651 of LNCS, 2011, pp. 291–304.

[29] ———, *Logic Characterization of Invisibly Structured Languages: the Case of Floyd Languages*, in 39th Int. Conf. on Current Trends in Theory and Practice of Computer Science (SOFSEM), vol. 7741 of LNCS, Springer, 2013, pp. 307–318.

[30] R. McNaughton, *Testing and generating infinite sequences by a finite automaton*, Information and Control, 9 (1966), pp. 521–530.

[31] ———, *Parenthesis Grammars*, Journ. ACM, 14 (1967), pp. 490–500.

[32] D. E. Muller, *Infinite sequences and finite machines*, in Proceedings of the Fourth Annual Symposium on Switching Circuit Theory and Logical Design, SWCT '63, Washington, DC, USA, 1963, IEEE Computer Society, pp. 3–16.

[33] D. Nowotka and J. Srba, *Height-Deterministic Pushdown Automata*, in MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings, L. Kucera and A. Kucera, eds., vol. 4708 of LNCS, Springer, 2007, pp. 125–134.

[34] F. Panella, M. Pradella, V. Lonati, and D. Mandrioli, *Operator precedence ω-languages*, in 17th International Conference on Developments in Language Theory (DLT), vol. 7907 of LNCS, 2013, pp. 396–408.

[35] ———, *Operator precedence ω-languages*, CoRR, abs/1301.2476 (2013). http://arxiv.org/abs/1301.2476.

[36] D. Perrin and J.-E. Pin, *Infinite words*, vol. 141 of Pure and Applied Mathematics, Elsevier, Amsterdam, 2004.

[37] M.O. Rabin, *Automata on infinite objects and Church's problem*, Regional conference series in mathematics, Published for the Conference Board of the Mathematical Sciences by the American Mathematical Society, 1972.

[38] A. K. Salomaa, *Formal Languages*, Academic Press, New York, NY, 1973.

[39] J. Thatcher, *Characterizing derivation trees of context-free grammars through a generalization of finite automata theory*, Journ. of Comp. and Syst.Sc., 1 (1967), pp. 317–322.

[40] W. Thomas, *Handbook of theoretical computer science (vol. B)*, MIT Press, Cambridge, MA, USA, 1990, ch. Automata on infinite objects, pp. 133–191.