Operator Precedence ω-languages

Federica Panella[1], Matteo Pradella[1], Violetta Lonati[2], Dino Mandrioli[1]

[1] DEIB - Politecnico di Milano, via Ponzio 34/5, Milano, Italy
{federica.panella, matteo.pradella, dino.mandrioli}@polimi.it
[2] DI - Università degli Studi di Milano, via Comelico 39/41, Milano, Italy
lonati@di.unimi.it

Abstract. Recent literature extended the analysis of ω-languages from the regular ones to various classes of languages with "visible syntax structure", such as visibly pushdown languages (VPLs). Operator precedence languages (OPLs), instead, were originally defined to support deterministic parsing and exhibit interesting relations with these classes of languages: OPLs strictly include VPLs, enjoy all relevant closure properties and have been characterized by a suitable automata family and a logic notation. We introduce here operator precedence ω-languages (ωOPLs), investigating various acceptance criteria and their closure properties. Whereas some properties are natural extensions of those holding for regular languages, others require novel investigation techniques. Application-oriented examples show the gain in expressiveness and verifiability offered by ωOPLs w.r.t. smaller classes.

Keywords: ω-languages, Operator precedence languages, Push-down automata, Closure properties, Infinite-state model checking.

1 Introduction

Languages of infinite strings, i.e. 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. The pioneering work by Büchi and others investigated their main algebraic properties in the context of finite state machines, pointing out commonalities and differences w.r.t. the finite length counterpart [4,17].

More recent literature, mainly under the motivation of widening the application of model checking techniques to larger language families, extended this analysis to various classes of languages with "visible structure", i.e., languages whose syntax structure is immediately visible in their strings: parenthesis languages, tree languages, visibly pushdown languages (VPLs) [1] are examples of such classes.

Operator precedence languages, instead, were defined by Floyd in the 1960s, and still are in use [9], with the original motivation of supporting deterministic parsing, which is trivial for visible structure languages but is crucial for general context-free languages such as programming languages [8], where structure is often left implicit (e.g. in arithmetic expressions). Recently, these seemingly unrelated classes of languages have been shown to share most major features; precisely OPLs strictly include VPLs and enjoy all the same closure properties [7]. This observation motivated characterizing OPLs in terms of a suitable automata family [10] and in terms of a logic notation [11], which was missing in previous literature.

In this paper we further the investigation of OPLs properties to the case of infinite strings, i.e., we introduce and study operator precedence ω-languages (ωOPLs). We prove closure and decidability properties that are a fundamental condition enabling infinite-state model checking. Also, we present a few simple application-oriented examples that show the considerable gain in expressiveness and verifiability offered by ω-OPLs w.r.t. previous classes.

We follow traditional lines of research in theory on ω-languages considering various acceptance criteria, their mutual expressiveness relations, and their closure properties, yet departing from the classical path for a number of critical and new issues. Not surprisingly, some properties are natural extensions of those holding for, say, regular languages or VPLs, whereas others required different and novel investigation techniques essentially due to the more general managing of the stack.

Due to space limitations, herein we focus on the newest and most interesting aspects. Also, we limit the technicalities of formal arguments to a minimum, relying instead on intuition and examples. The reader can find more results and all details in the technical report [14]. The paper is organized as follows. The next section provides basic concepts on operator precedence languages of finite-length words and on operator precedence automata able to recognize them. Section 3 defines operator precedence automata which can deal with infinite strings, analyzing various classical acceptance conditions for ω-abstract machines. Section 4 proves the closure properties they enjoy w.r.t typical operations on ω-languages and shows also that the emptiness problem is decidable for these formalisms. Finally, Section 5 draws some conclusions.

2 Preliminaries

Operator precedence languages [7,8] have been characterized in terms of both a generative formalism (operator precedence grammars, OPGs) and an equivalent operational one (operator precedence automata, OPAs, named Floyd automata or FAs in [10]), but in this paper we consider the latter, as it is better suited to model and verify nonterminating computations.

Let Σ be an alphabet. The empty string is denoted ε. Between the symbols of the alphabet three types of operator precedence (OP) binary relations can hold: *yields* precedence, *equal* in precedence and *takes* precedence, denoted $\lessdot$, $\doteq$ and $\gtrdot$ respectively. Notice that $\doteq$ is not necessarily an equivalence relation, and $\lessdot$ and $\gtrdot$ are not necessarily strict partial orders. 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 that requires the lookback and lookahead of one character to determine the next action to perform [9]. The initial # can only yield precedence, and other symbols can only take precedence on the ending #.

Definition 1. *An* operator precedence matrix *(OPM) M over an alphabet Σ is a* $|\Sigma \cup \{\#\}| \times |\Sigma \cup \{\#\}|$ *array that with each ordered pair* (a, b) *associates the set* M_{ab} *of OP* *relations holding between a and b. M is conflict-free iff* $\forall a, b \in \Sigma, |M_{ab}| \leq 1$. *We call* (Σ, M) *an operator precedence alphabet if M is a conflict-free OPM on* Σ.

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

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

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 $a \circ b$. Two matrices are *compatible* if their union is conflict-free. A matrix is *complete* if it contains no empty case.

In the following we assume that M is $\doteq$*-acyclic*, which means that $c_1 \doteq c_2 \doteq \cdots \doteq c_k \doteq c_1$ does not hold for any $c_1, c_2, \ldots, c_k \in \Sigma, k \geq 1$. See [14] for a discussion on this hypothesis. Let also (Σ, M) be an OP alphabet.

Definition 2. *A* nondeterministic operator precedence automaton *(OPA) is 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,*
- $\delta : Q \times (\Sigma \cup Q) \to 2^Q$ *is the transition function.*

The transition function can be seen as the union of two disjoint functions:

$$\delta_{\text{push}} : Q \times \Sigma \to 2^Q \qquad \delta_{\text{flush}} : Q \times Q \to 2^Q$$

An OPA can be represented by a graph with Q as the set of vertices and $\Sigma \cup Q$ as the set of edge labels: there is an edge from state q to state p labeled by $a \in \Sigma$ if and only if $p \in \delta_{push}(q, a)$, and there is an edge from state q to state p labeled by $r \in Q$ if and only if $p \in \delta_{flush}(q, r)$. To distinguish flush transitions from push transitions we denote the former ones by a double arrow.

To define the semantics of the automaton, we introduce some notation. We use letters $p, q, p_i, q_i, \ldots$ for states in Q and we set $\Sigma' = \{a' \mid a \in \Sigma\}$; symbols in Σ' are called *marked* symbols.

Let Γ be $(\Sigma \cup \Sigma' \cup \{\#\}) \times Q$; we denote symbols in Γ as $[a\ q]$, $[a'\ q]$, or $[\#\ q]$, respectively. We set $symbol([a\ q]) = symbol([a'\ q]) = a$, $symbol([\#\ q]) = \#$, and $state([a\ q]) = state([a'\ q]) = state([\#\ q]) = q$. Given a string $\beta = B_1 B_2 \ldots B_n$ with $B_i \in \Gamma$, we set $state(\beta) = state(B_n)$.

A *configuration* is any pair $C = \langle \beta\ ,\ w \rangle$, where $\beta = B_1 B_2 \ldots B_n \in \Gamma^*$, $symbol(B_1) = \#$, and $w = a_1 a_2 \ldots a_m \in \Sigma^* \#$. A configuration represents both the contents β of the stack and the part of input w still to process.

A computation (run) of the automaton is a finite sequence of moves $C \vdash C_1$; there are three kinds of moves, depending on the precedence relation between $symbol(B_n)$ and a_1:

push move: if $symbol(B_n) \doteq a_1$ then $C_1 = \langle \beta[a_1\ q]\ ,\ a_2 \ldots a_m \rangle$, with $q \in \delta_{push}(state(\beta), a_1)$;

mark move: if $symbol(B_n) \lessdot a_1$ then $C_1 = \langle \beta[a_1{}'\ q]\ ,\ a_2 \ldots a_m \rangle$, with $q \in \delta_{push}(state(\beta), a_1)$;

flush move: if $symbol(B_n) \gtrdot a_1$ then let i the greatest index such that $symbol(B_i) \in \Sigma'$ (such index always exists). Then $C_1 = \langle B_1 B_2 \ldots B_{i-2}[symbol(B_{i-1})\ q]\ ,\ a_1 a_2 \ldots a_m \rangle$, with $q \in \delta_{flush}(state(B_n), state(B_{i-1}))$.

Push and mark moves both push the input symbol on the top of the stack, together with the new state computed by δ_{push}; such moves differ only in the marking of the symbol on top of the stack. The flush move is more complex: the symbols on the top of the stack are removed until the first marked symbol (*included*), and the state of the next symbol below them in the stack is updated by δ_{flush} according to the pair of states that delimit the portion of the stack to be removed; notice that in this move the input symbol is not consumed and it remains available for the following move.

Finally, we say that a configuration $[\# \; q_I]$ is *starting* if $q_I \in I$ and a configuration $[\# \; q_F]$ is *accepting* if $q_F \in F$. The language accepted by the automaton is defined as:

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

An OPA is *deterministic* when I is a singleton and $\delta_{\text{push}}(q,a)$ and $\delta_{\text{flush}}(q,p)$ have at most one element, for every $q, p \in Q$ and $a \in \Sigma$.

An *operator precedence transducer* is defined in the natural way.

Example 1. As an introductory example, consider a language of queries on a database expressed in relational algebra. We consider a subset of classical operators (union, intersection, selection σ, projection π and natural join $\bowtie$). Just like mathematical operators, the relational operators have precedences between them: unary operators σ and π have highest priority, next highest is the "*multiplicative*" operator $\bowtie$, lowest are the "*additive*" operators $\cup$ and $\cap$. Denote as T the set of tables of the database and, for the sake of simplicity, let E be a set of conditions for the unary operators. The OPA depicted in Figure 1 accepts the language of queries without parentheses on the alphabet $\Sigma = T \cup \{\bowtie, \cup, \cap\} \cup \{\sigma, \pi\} \times E$, where we use letters $A, B, R \ldots$ for elements in T and we write σ_{expr} for a pair $(\sigma, expr)$ of selection with condition $expr$ (similarly for projection π_{expr}). The same figure also shows an accepting computation on input $A \cup B \bowtie C \bowtie \pi_{\text{expr}} D$.

Notice that the sentences of this language show the same structure as arithmetic expressions with prioritized operators and without parentheses, which cannot be represented by VPAs due to the particular shape of their OPM [7].

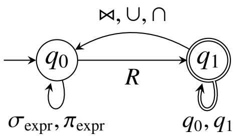
Definition 3. *A* simple chain *is a word $a_0 a_1 a_2 \ldots a_n a_{n+1}$, written as $\langle^{a_0} a_1 a_2 \ldots a_n{}^{a_{n+1}} \rangle$, such that: $a_0 \in \Sigma \cup \{\#\}$, $a_i \in \Sigma$ for every $i : 1 \le i \le n+1$, $M_{a_0 a_{n+1}} \neq \emptyset$, and $a_0 \lessdot a_1 \doteq a_2 \ldots a_{n-1} \doteq a_n \gtrdot a_{n+1}$.*

A composed chain *is a word $a_0 x_0 a_1 x_1 a_2 \ldots a_n x_n a_{n+1}$, where $\langle^{a_0} a_1 a_2 \ldots a_n{}^{a_{n+1}} \rangle$ is a simple chain, and $x_i \in \Sigma^*$ is the empty word or is such that $\langle^{a_i} x_i{}^{a_{i+1}} \rangle$ is a chain (simple or composed), for every $i : 0 \le i \le n$. Such a composed chain will be written as $\langle^{a_0} x_0 a_1 x_1 a_2 \ldots a_n x_n{}^{a_{n+1}} \rangle$.*

A word w over (Σ, M) is compatible *with M iff for each pair of consecutive letters c, d in w $M_{cd} \neq \emptyset$, and for each factor x of $\#w\#$ such that $x = a_0 x_0 a_1 x_1 a_2 \ldots a_n x_n a_{n+1}$ where $a_0 \lessdot a_1 \doteq a_2 \ldots a_{n-1} \doteq a_n \gtrdot a_{n+1}$ and $x_i \in \Sigma^*$ is the empty word or is such that $\langle^{a_i} x_i{}^{a_{i+1}} \rangle$ is a chain (simple or composed) for every $0 \le i \le n$, $M_{a_0 a_{n+1}} \neq \emptyset$.*

Definition 4. *Let $\mathcal{A}$ be an operator precedence automaton. A* support *for the simple chain $\langle^{a_0} a_1 a_2 \ldots a_n{}^{a_{n+1}} \rangle$ is any path in $\mathcal{A}$ of the form*

$$\xrightarrow{a_0} q_0 \xrightarrow{a_1} q_1 \longrightarrow \ldots \longrightarrow q_{n-1} \xrightarrow{a_n} q_n \overset{q_0}{\Longrightarrow} q_{n+1} \tag{1}$$



	R	σ_{expr}	π_{expr}	$\bowtie$	$\cup$	$\cap$	$\#$
R				$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
σ_{expr}	$\lessdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
π_{expr}	$\lessdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
$\bowtie$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
$\cup$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
$\cap$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
$\#$	$\lessdot$	$\lessdot$	$\lessdot$				$\doteq$

$$
\begin{array}{lr}
\langle [\# \; q_0] & , A \cup B \bowtie C \bowtie \pi_{\text{expr}} D \# \rangle \\
\langle [\# \; q_0][A' \; q_1] & , \cup B \bowtie C \bowtie \pi_{\text{expr}} D \# \rangle \\
\langle [\# \; q_1] & , \cup B \bowtie C \bowtie \pi_{\text{expr}} D \# \rangle \\
\langle [\# \; q_1][\cup' \; q_0] & , B \bowtie C \bowtie \pi_{\text{expr}} D \# \rangle \\
\langle [\# \; q_1][\cup' \; q_0][B' \; q_1] & , \bowtie C \bowtie \pi_{\text{expr}} D \# \rangle \\
\langle [\# \; q_1][\cup' \; q_1] & , \bowtie C \bowtie \pi_{\text{expr}} D \# \rangle \\
\langle [\# \; q_1][\cup' \; q_1][\bowtie' \; q_0] & , C \bowtie \pi_{\text{expr}} D \# \rangle \\
\langle [\# \; q_1][\cup' \; q_1][\bowtie' \; q_0][C' \; q_1] & , \bowtie \pi_{\text{expr}} D \# \rangle \\
\langle [\# \; q_1][\cup' \; q_1][\bowtie' \; q_1] & , \bowtie \pi_{\text{expr}} D \# \rangle \\
\langle [\# \; q_1][\cup' \; q_1][\bowtie' \; q_1][\bowtie' \; q_0] & , \pi_{\text{expr}} D \# \rangle \\
\langle [\# \; q_1][\cup' \; q_1][\bowtie' \; q_1][\bowtie' \; q_0][\pi_{\text{expr}}' \; q_0] & , D \# \rangle \\
\langle [\# \; q_1][\cup' \; q_1][\bowtie' \; q_1][\bowtie' \; q_0][\pi_{\text{expr}}' \; q_0][D' \; q_1] & , \# \rangle \\
\langle [\# \; q_1][\cup' \; q_1][\bowtie' \; q_1][\bowtie' \; q_0][\pi_{\text{expr}}' \; q_1] & , \# \rangle \\
\langle [\# \; q_1][\cup' \; q_1][\bowtie' \; q_1][\bowtie' \; q_1] & , \# \rangle \\
\langle [\# \; q_1][\cup' \; q_1][\bowtie' \; q_1] & , \# \rangle \\
\langle [\# \; q_1][\cup' \; q_1] & , \# \rangle \\
\langle [\# \; q_1] & , \# \rangle
\end{array}
$$

Fig. 1: Automaton, precedence matrix and example of computation for language of Example 1.

The label of the last (and only) flush is exactly q_0, i.e. the first state of the path; this flush is executed because of relation $a_n \gtrdot a_{n+1}$.
A support for the composed chain $\langle^{a_0} x_0 a_1 x_1 a_2 \ldots a_n x_n{}^{a_{n+1}} \rangle$ *is any path in $\mathcal{A}$ of the form*

$$\xrightarrow{a_0} q_0 \overset{x_0}{\rightsquigarrow} q_0' \xrightarrow{a_1} q_1 \overset{x_1}{\rightsquigarrow} q_1' \xrightarrow{a_2} \ldots \xrightarrow{a_n} q_n \overset{x_n}{\rightsquigarrow} q_n' \overset{q_0'}{\Longrightarrow} q_{n+1} \qquad (2)$$

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

- *if $x_i \neq \varepsilon$, then $\xrightarrow{a_i} q_i \overset{x_i}{\rightsquigarrow} q_i'$ is a support for the chain $\langle^{a_i} x_i{}^{a_{i+1}} \rangle$, i.e., it can be decomposed as $\xrightarrow{a_i} q_i \overset{x_i}{\rightsquigarrow} q_i'' \overset{q_i}{\Longrightarrow} q_i'$.*
- *if $x_i = \varepsilon$, then $q_i' = q_i$.*

Notice that the label of the last flush is q_0'.

The chains fully determine the structure of the parsing of any automaton on a word compatible with M, and hence the structure of the syntax tree of the word. Indeed, if the automaton performs the computation $\langle [a \; q_0] \; , \; xb \rangle \overset{*}{\vdash} \langle [a \; q] \; , \; b \rangle$ on a factor axb, then $\langle^a x^b \rangle$ is necessarily a chain over (Σ, M) and there exists a support like (2) with $x = x_0 a_1 \ldots a_n x_n$ and $q_{n+1} = q$.

3 Operator precedence ω-languages and automata

Traditionally, ω-automata have been classified on the basis of the acceptance condition they are equipped with. All acceptance conditions refer to the occurrence of states which are visited in a computation of the automaton, and they generally impose constraints on those states that are encountered infinitely (or also finitely) often during a run. Classical notions of acceptance (introduced by Büchi [4], Muller [12], Rabin [15], Streett [16]) can be naturally adapted to ω-automata for operator precedence languages and can be characterized according to a peculiar acceptance component of the automaton on ω-words. Here we focus mainly on nondeterministic Büchi-operator precedence ω-automata with acceptance by final state; other models are briefly presented and compared in Section 3.1.

As usual, we denote by Σ^ω the set of infinite-length words over Σ. Thus, the symbol # occurs only at the beginning of an ω-word. Given a precedence alphabet (Σ, M), the definition of an ω-word compatible with the OPM M and the notion of syntax tree of an infinite-length word are the natural extension of these concepts for finite strings. We also use the notation "$\exists^\omega i$" as a shorthand for "there exist infinitely many i".

Definition 5. *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 #.* Configurations *and* (infinite) runs *are defined as for OP automata on finite-length words.*
Let $\mathcal{S}$ be a run of the automaton on a given word $x \in \Sigma^\omega$. Define $In(\mathcal{S}) = \{q \in Q \mid \exists^\omega i \ \langle \beta_i\ ,\ x_i \rangle \in \mathcal{S}$ with $state(\beta_i) = q\}$ as the set of states that occur infinitely often at the top of the stack of configurations in $\mathcal{S}$. A run $\mathcal{S}$ of an ωOPBA *on an infinite word $x \in \Sigma^\omega$ is* successful *iff there exists a state $q_f \in F$ such that $q_f \in In(\mathcal{S})$. $\mathcal{A}$* accepts *$x \in \Sigma^\omega$ iff there is a successful run of $\mathcal{A}$ on x.*

As in the finite-length case, the class of languages accepted by ωBVPAs (*nondeterministic Büchi visibly pushdown ω-automata*) is a proper subset of that accepted by ωOPBAs. Indeed, classical families of automata, like Visibly Pushdown Automata [1], imply several restrictions that hinder them from being able to deal with the concept of precedence among symbols and make them unsuitable to model several interesting aspects often exhibited by real-world systems in various contexts.

To mention a few examples, a natural field of application of ωOPLs is the representation of processes or tasks which are assigned a priority to fulfill their requirements, which is a common paradigm in the area of operating systems, e.g. the processing of interrupts from peripherals with different priorities, or in the context of Web services, where servers provide services to users with different privileges. ωOPLs can also model the run-time behavior of database systems, e.g. for modeling sequences of user's transactions with possible rollbacks, and revision control systems (such as subversion or git). Examples of such systems are more extensively presented in [14].

3.1 Other automata models for operator precedence ω-languages

There are several possibilities to define other classes of OP ω-languages. We may introduce a variant of ωOPBA (called ωOPBEA) which recognizes a word if the automaton traverses final states with an empty stack infinitely often, and we may as well consider automata with acceptance conditions other than Büchi's, as e.g. Muller operator precedence ω-automata (ωOPMAs). Furthermore, *deterministic* ωOPA and OP *ω-transducers* can be specified in the natural way as for operator precedence automata on finite-length words.

In general, the relationships among languages recognized by the different classes of operator precedence ω-automata and visibly pushdown ω-languages are summarized in Figure 2, where ωDOPBA and ωDOPMA denote the classes of deterministic ωOPBAs and deterministic ωOPMAs respectively. The detailed proofs of the strict containment relations holding among the classes in Figure 2 are presented in [13, Chapter 4]. The proofs regarding the relationships between those classes which are not comparable are described in [14]. In the sequel we will consider only the most expressive class of ωOPAs, i.e. ωOPBA.

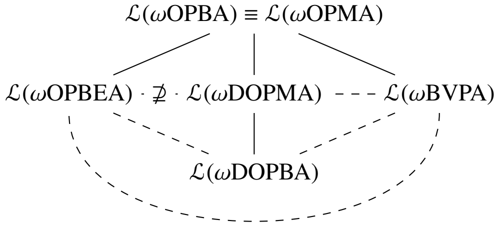

$\mathcal{L}(\omega\text{OPBA}) \equiv \mathcal{L}(\omega\text{OPMA})$
$\mathcal{L}(\omega\text{OPBEA})$
$\not\supseteq$
$\mathcal{L}(\omega\text{DOPMA})$
$\mathcal{L}(\omega\text{BVPA})$
$\mathcal{L}(\omega\text{DOPBA})$

Fig. 2: Containment relations for ωOPLs. Solid lines denote strict inclusions; dashed lines link classes which are not comparable.

4 Closure properties and emptiness problem

In this section we focus on the most interesting closure properties of ωOPAs, which are summarized in Table 1, where they are compared with the properties enjoyed by VPAs on infinite-length words. All operations are assumed to be applied to, and, when closure subsists, to produce, languages with compatible OPMs.

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

Table 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 and an ω-language L_2).

The main family ωOPBA is closed under Boolean operations and under concatenation with a language of finite words accepted by an OPA. Furthermore, the emptiness problem is decidable for ωOPAs in polynomial time because they can be interpreted as pushdown automata on infinite-length words: e.g. [5] 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; thus the emptiness problem for the intersection of the language recognized by a pushdown process and the language of a given property in this logic is decidable.

Closure under intersection and union hold for ωOPBAs as for classical ω-regular languages: these closure properties, together with those enjoyed by ωDOPBAs and ωDOPMAs, can be proved in a similar way as for classical families of ω-automata, and their proofs can be found in [13, Chapter 5] and [14, Section 4]. Closure under complementation and concatenation for ωOPBAs, instead, required novel investigation techniques.

Closure under concatenation

Unlike other families of languages, closure under concatenation has been proved for finite-length word OPLs by using their generating grammars with some difficulty [6], essentially due to the peculiar structure of their syntax trees. In the case of OPAs, and of infinite-length words, difficulties are further exacerbated by the fact that an OPA relies on the end-marker # to empty the stack before accepting a string; on the contrary, when parsing the concatenation of two OPL strings, the stack *cannot* always be emptied after reading the former one; for instance, consider a language $L_1 \subseteq \Sigma^*$ with an OPM where $a \lessdot a$ and $b \lessdot a$: a word in L_1 ending with a b concatenated with a^ω compels the OPA to let the stack indefinitely grow with no chance for any flush move after the reading of the L_1 word.

To overtake this difficulty we use a new approach which heavily exploits nondeterminism; remember in fact that, similarly to regular languages and VPLs, ωDOPBAs are strictly less powerful than ωOPBAs (see Figure 2). The basic idea consists in *guessing* the end of the first word and deciding whether it could be accepted by the original OPA recognizing L_1 *without emptying the stack*. This is a nontrivial job which requires storing suitable information in the stack at any mark move as it will be explained shortly.

To achieve our goal we first introduce a variant of the semantics of the transition relation and of the acceptance condition for OPAs: a string x is accepted if the automaton reaches a final state right at the end of the parsing of the whole word, and it does not perform any flush move determined by the ending delimiter # to empty the stack; thus it stops just after having put the last symbol of x on the stack. Precisely, the semantics of the transition relation differs from the definition of classical OPAs in that, once a configuration with the endmarker as lookahead is reached, the computation cannot evolve in any subsequent configuration, i.e. a flush move $C \vdash C_1$ with $C = \langle B_1 B_2 \dots B_n \ , \ y\# \rangle$ and $symbol(B_n) \gtrdot y\#$ is performed only if $y \neq \varepsilon$. The language accepted by this variant of the automaton (denoted as $\widetilde{L}$) is the set of words:

$$\widetilde{L}(\mathcal{A}) = \{x \mid \langle [\# \ q_I] \ , \ x\# \rangle \overset{*}{\vdash} \langle \gamma [a \ q_F] \ , \ \# \rangle, q_I \in I, q_F \in F, \gamma \in \Gamma^*, a \in \Sigma \cup \{\#\}\}$$

We emphasize 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 accept if just one of the visited states is final, this type of automaton cannot perform any (flush) move after reaching the endmarker through the last look-ahead.

Nevertheless, the variant and the classical definition of OPA are equivalent: the following lemma shows the first direction of inclusion between the two formalisms. Statement 1 in [14], although not necessary to prove closure under concatenation of $\mathcal{L}(\omega\text{OPBA})$, completes the proof of equivalence between traditional and variant OPAs.

Lemma 1. *Let $\mathcal{A}_1$ be a nondeterministic OPA defined on an OP alphabet (Σ, M) with s states. Then there exists a nondeterministic OPA $\mathcal{A}_2$ with the same precedence matrix as $\mathcal{A}_1$ and $O(|\Sigma| s^2)$ states such that $L(\mathcal{A}_1) = \widetilde{L}(\mathcal{A}_2)$.*

Sketch of the proof. Consider a word of finite length w which is preceded by a delimiter # but which is not ended with such a symbol. Define a chain in a word w as *maximal* if it does not belong to a larger composed chain. In a word of finite length preceded and ended by # only the outer chain $\langle^{\#} w^{\#}\rangle$ is maximal.

The *body* of a chain $\langle^{a} w^{b}\rangle$, simple or composed, is the word w. A word w which is preceded but not ended by a delimiter # can be factored in a unique way as a sequence of bodies of maximal chains w_i and letters a_i as $\#\ w = \#\ w_1 a_1 w_2 a_2 \ldots w_n a_n$ where $\langle^{a_{i-1}} w_i{}^{a_i}\rangle$ are maximal chains and each w_i can be possibly missing, with $a_0 = \#$ and $\forall i : 1 \leq i \leq n-1\ a_i \lessdot a_{i+1}$ or $a_i \doteq a_{i+1}$. During the parsing of word w, the symbols of the string are put on the stack and, whenever a chain is recognized, the letters of its body are flushed away. Hence, after the parsing of $\#w$ the stack contains only the symbols $\#\ a_1\ a_2 \ldots\ a_n$, which we call *pending letters*, and is structured as a sequence

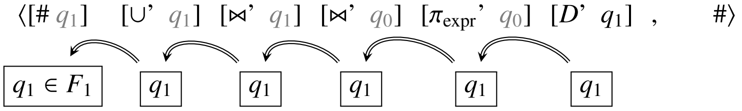
$$\# \lessdot a_{i_1} = a_1 \doteq a_2 \doteq \ldots \lessdot a_{i_2} \doteq a_{i_2+1} \doteq \ldots \lessdot a_{i_3} \doteq a_{i_3+1} \doteq \ldots \lessdot a_{i_k} \doteq a_{i_k+1} \doteq \ldots \doteq a_n$$

of k *open chains*, i.e., sequences of symbols $b_0 \lessdot b_1 \doteq b_2 \doteq \ldots \doteq b_m$, for $m \geq 1$. At the end of the computation a classical OPA performs a series of flush moves due to the presence of the final symbol #. These moves progressively empty the stack, removing one by one the open chains.

A nondeterministic automaton that, unlike classical OPAs, does not resort to the last # for the recognition, guesses nondeterministically the ending point of each open chain on the stack and guesses how, in an accepting run, the states in these points of the stack would be updated if the final flush moves were progressively performed. The automaton must behave as if, at the same time, it simulates two steps of the accepting run of a classical OPA: a move during the parsing of the string and a step during the final flush transitions which will later on empty the stack, leading to a final state. To this aim, the states of a classical OPA are augmented with an additional component to store the necessary information. If the forward path consisting of moves during the parsing of the string and the backward path of flush moves guessed by the automaton can consistently meet and be rejoined when the parsing of the input string stops, then combined they constitute an accepting run of the classical OPA.

A variant 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 flush 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 flush moves. To illustrate, we use the following example.

Example 2. Consider Figure 1. If we take the input word of this computation without the ending marker #, then the sequence of pending letters on the stack, after the automaton puts on the stack the last symbol D, is $\# \lessdot \cup \lessdot \bowtie \lessdot \bowtie \lessdot \pi_{expr} \lessdot D$. There are five open chains with starting symbols $\cup$, $\bowtie$, $\bowtie$, π_{expr}, D, hence the computation ends with five consecutive flush moves determined by the delimiter #. The following figure shows the configuration just before looking ahead at the symbol #. The states (depicted within boxes) at the end of the open chains are those placeholders that an equivalent variant OPA should guess in order to find in advance the last flush moves $q_1 = \boxed{q_1} \overset{q_0}{\Longrightarrow} \boxed{q_1} \overset{q_0}{\Longrightarrow} \boxed{q_1} \overset{q_1}{\Longrightarrow} \boxed{q_1} \overset{q_1}{\Longrightarrow} \boxed{q_1} \overset{q_1}{\Longrightarrow} \boxed{q_1 \in F_1}$ of the accepting run.


⟨[# q_1] [∪' q_1] [⋈' q_1] [⋈' q_0] [π_{expr}' q_0] [D' q_1] , #⟩
$q_1 \in F_1$
q_1
q_1
q_1
q_1
q_1

The corresponding configuration of the variant OPA, with the augmented states, would be:

$$\langle [\# \ q_1, \boxed{q_1}] \ \ [\cup\text{'} \ q_1, \boxed{q_1}] \ \ [\bowtie\text{'} \ q_1, \boxed{q_1}] \ \ [\bowtie\text{'} \ q_0, \boxed{q_1}] \ \ [\pi_{\text{expr}}\text{'} \ q_0, \boxed{q_1}] \ \ [D\text{'} \ q_1, \boxed{q_1}] \ , \ \#\rangle$$

The formal definition of the variant automaton and the proof of its equivalence with a classical OPA are presented in [14].

We are now ready for the main result.

Theorem 1. *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 a* ωOPBA *with OPM $M_3 \supseteq M_1 \cup M_2$ and $O(|\Sigma|(s_1^2 + s_2^2))$ states.*

Sketch of the proof. Intuitively, a nondeterministic ωOPBA recognizing $L_1 \cdot L_2$ first simulates the variant automaton recognizing L_1, guesses the end of the L_1 word, and leaves a suitable “marker” on top of the stack before beginning the simulation of the second ωOPBA. In this process, the only nontrivial technical aspect is the fact that the second phase cannot leave unaffected the part of the stack that is left as a “legacy” by the first phase; thus, some flush moves must “invade” the lower part of the stack and the two phases cannot be completely independent, somewhat mimicking the construction of the OP grammar generating the concatenation of two OPLs [7]. □

Closure under complementation

Theorem 2. *Let M be a conflict-free precedence matrix on an alphabet Σ. Denote by $L_M \subseteq \Sigma^\omega$ the ω-language comprising all infinite words $x \in \Sigma^\omega$ compatible with M.*
Let L be an ω-language on Σ that can be recognized by a nondeterministic ωOPBA *with precedence matrix M and s states. Then the complement of L w.r.t L_M is recognized by an* ωOPBA *with the same precedence matrix M and $2^{O(s^2)}$ states.*

Sketch of the proof. The proof follows to some extent the structure of the corresponding proof for Büchi VPAs [1], 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 ωOPAs.

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. such that there is a run of $\mathcal{A}$ on every ω-word on Σ compatible with M.

In general, a sentence on Σ^ω can be factored in a unique way so as to distinguish the subfactors of the string that can be recognized without resorting to the stack of the automaton and those subwords for which the use of the stack is necessary.
More precisely, an ω-word $w \in \Sigma^\omega$ 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 a finite sequence of letters such that $\langle^{l_i} w_i^{first_{i+1}}\rangle$ is a chain, 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.

Notice that 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 w_i has no prefix which is a chain.

As an example, 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 $b \lessdot d$, the unique factorization is $w = w_1 b w_3 w_4 b \ldots$, where b is a pending letter and $\langle^{\#} ac^{b}\rangle, \langle^{b} a^{d}\rangle, \langle^{b} d^{b}\rangle$ are chains.

Define a *semisupport* for a chain $\langle^{a_0} x^{a_{n+1}}\rangle$ (simple or composed) as any path in $\mathcal{A}$ which is a support for the chain (Equations 1 and 2), where however the initial state of the path is not restricted to be the state reached after reading symbol a_0.

Let $x \in \Sigma^*$ be such that $\langle^{a} x^{b}\rangle$ is a chain for some a, 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 semisupport $q \overset{x}{\leadsto} p$ in $\mathcal{A}$, and $f = 1$ iff the semisupport contains a state in F. Also let $\mathcal{T}$ be the set of all such $T(x)$, i.e., $\mathcal{T}$ contains set of triples identifying all semisupports for some chain, and set $PR = \Sigma \cup \mathcal{T}$. The *pseudorun* for w in $\mathcal{A}$ is the ω-word $w' = y_1 y_2 y_3 \ldots \in PR^{\omega}$ where $y_i = a_i$ if $w_i = a_i$, otherwise $y_i = T(w_i)$. For the example above, then, $w' = T(ac)\ b\ T(a)\ T(d)\ b \ldots$.

Deferring to [14] further details of our proof, which from this point on resembles [1] with the necessary adaptions, we can define a nondeterministic Büchi finite-state automaton $\mathcal{A}_R$ over alphabet PR which has $O(s)$ states and accepts a pseudorun iff the corresponding words on Σ belong to $L(\mathcal{A})$. Consider then a deterministic Streett automaton $\mathcal{B}_R$ that accepts the complement of $L(\mathcal{A}_R)$ on the alphabet PR and, receiving pseudoruns as input words, accepts only words in $L_M \backslash L(\mathcal{A})$. The automaton $\mathcal{B}_R$ has $2^{O(\mathrm{s} \log \mathrm{s})}$ states and $O(s)$ accepting constraints [17]. We can build a nondeterministic transducer ωOPBA $\mathcal{B}$ that on reading w generates online the pseudorun w', which will be given as input to $\mathcal{B}_R$. The final automaton, that recognizes the complement of $L = L(\mathcal{A})$ w.r.t L_M, is the ωOPBA representing the product of $\mathcal{B}_R$ (converted to a Büchi automaton), which has $2^{O(\mathrm{s} \log \mathrm{s})}$ states, and $\mathcal{B}$, with $2^{O(s^2)}$ states; thus it has $2^{O(s^2)}$ states. □

5 Conclusions and further research

We presented a formalism for infinite-state model checking based on operator precedence languages, continuing to explore the paths in the lode of operator precedence languages started up by Robert Floyd a long time ago. We introduced various classes of automata able to recognize operator precedence languages of infinite-length words whose expressive power outperforms classical models for infinite-state systems as Visibly Pushdown ω-languages, allowing to represent more complex systems in several practical contexts. We proved the closure properties of ωOPLs under Boolean operations that, along with the decidability of the emptiness problem, are fundamental for the application of such formalism to model checking.

Our results open further directions of research. A first interesting topic deals with the characterization of ωOPLs in terms of suitable monadic second order logical formulas, that has already been studied for operator precedence languages of finite-length strings [11]. This would further strengthen applicability of model checking techniques. The next step of investigation will regard the actual design and study of complexity issues of algorithms for model checking of expressive logics on these pushdown models. We expect that the peculiar features of Floyd languages, as their "locality principle" which makes them suitable for parallel and incremental parsing [2,3] and their expressivity, might be interestingly exploited to devise efficient and attractive software model-checking procedures and approaches.

References

1. Alur, R., Madhusudan, P.: Adding nesting structure to words. Journ. ACM 56(3) (2009)
2. Barenghi, A., Crespi Reghizzi, S., Mandrioli, D., Pradella, M.: Parallel parsing of operator precedence grammars. Information Processing Letters (2013), DOI:10.1016/j.ipl.2013.01.008
3. Barenghi, A., Viviani, E., Crespi Reghizzi, S., Mandrioli, D., Pradella, M.: PAPAGENO: a parallel parser generator for operator precedence grammars. In: 5th International Conference on Software Language Engineering (SLE) (2012)
4. Büchi, J.R.: Weak Second-Order Arithmetic and Finite Automata. Mathematical Logic Quarterly 6(1-6), 66–92 (1960)
5. Burkart, O., Steffen, B.: Model checking for context-free processes. In: CONCUR '92, LNCS, vol. 630, pp. 123–137 (1992)
6. Crespi Reghizzi, S., Mandrioli, D.: Operator Precedence and the Visibly Pushdown Property. In: LATA. pp. 214–226 (2010)
7. Crespi Reghizzi, S., Mandrioli, D.: Operator Precedence and the Visibly Pushdown Property. Journal of Computer and System Science 78(6), 1837–1867 (2012)
8. Floyd, R.W.: Syntactic Analysis and Operator Precedence. Journ. ACM 10(3), 316–333 (1963)
9. Grune, D., Jacobs, C.J.: Parsing techniques: a practical guide. Springer, New York (2008)
10. Lonati, V., Mandrioli, D., Pradella, M.: Precedence Automata and Languages. In: 6th Int. Computer Science Symposium in Russia (CSR), LNCS, vol. 6651, pp. 291–304 (2011)
11. Lonati, V., Mandrioli, D., Pradella, M.: 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), LNCS, vol. 7741, pp. 307–318. Springer (2013)
12. Muller, D.E.: Infinite sequences and finite machines. In: Proceedings of the Fourth Annual Symposium on Switching Circuit Theory and Logical Design. pp. 3–16. SWCT '63, IEEE Computer Society, Washington, DC, USA (1963)
13. Panella, F.: Floyd languages for infinite words. Master's thesis, Politecnico di Milano (2011), http://home.dei.polimi.it/panella
14. Panella, F., Pradella, M., Lonati, V., Mandrioli, D.: Operator precedence ω-languages. CoRR abs/1301.2476 (2013), http://arxiv.org/abs/1301.2476
15. Rabin, M.: 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)
16. Streett, R.S.: Propositional dynamic logic of looping and converse is elementarily decidable. Information and Control 54(1-2), 121 – 141 (1982)
17. Thomas, W.: Handbook of theoretical computer science (vol. B). chap. Automata on infinite objects, pp. 133–191. MIT Press, Cambridge, MA, USA (1990)