Abstract

The operator precedence languages (OPLs) represent the largest known subclass of the context-free languages which enjoys all desirable closure and decidability properties. This includes the decidability of language inclusion, which is the ultimate verification problem. Operator precedence grammars, automata, and logics have been investigated and used, for example, to verify programs with arithmetic expressions and exceptions (both of which are deterministic pushdown but lie outside the scope of the visibly pushdown languages). In this paper, we complete the picture and give, for the first time, an algebraic characterization of the class of OPLs in the form of a syntactic congruence that has finitely many equivalence classes exactly for the operator precedence languages. This is a generalization of the celebrated Myhill-Nerode theorem for the regular languages to OPLs. As one of the consequences, we show that universality and language inclusion for nondeterministic operator precedence automata can be solved by an antichain algorithm. Antichain algorithms avoid determinization and complementation through an explicit subset construction, by leveraging a quasi-order on words, which allows the pruning of the search space for counterexample words without sacrificing completeness. Antichain algorithms can be implemented symbolically, and these implementations are today the best-performing algorithms in practice for the inclusion of finite automata. We give a generic construction of the quasi-order needed for antichain algorithms from a finite syntactic congruence. This yields the first antichain algorithm for OPLs, an algorithm that solves the \textsc{ExpTime}-hard language inclusion problem for OPLs in exponential time.

1 Introduction

Pushdown automata are a fundamental model of computation and the preferred formalism to parse programs in a deterministic manner. In verification, they are used to encode the behaviors of both systems and specifications that involve, for example, nested procedure calls.
However, unlike for regular languages specified by finite automata, the inclusion of context-free languages given by pushdown automata is undecidable, even for deterministic machines. This is why expressive subclasses of context-free languages with decidable properties have been studied in the past decades. Prominent among those formalisms is the class of visibly pushdown languages [3], which is strictly contained in the deterministic context-free languages. A visibly pushdown language (VPL) is a context-free language where each word admits a single parse tree, which does not depend on the pushdown automaton that generates (or accepts) the word. More technically, visibly pushdown automata (VPDAs) extend finite automata with a memory stack that is restricted to “push” and “pop” operations on disjoint subsets of the input alphabet. VPDAs have become popular in verification for several reasons. First, they recognize “well-nested” words, which find applications in the analysis of HTML and XML documents. Second, their restricted stack behavior enables desirable closure and decidability properties; in particular, in contrast to deterministic context-free languages, VPDAs can be complemented and their inclusion is decidable. Third, the VPLs admit a generalization of the celebrated Myhill-Nerode theorem for the regular languages [2]: they can be characterized algebraically by a finite syntactic congruence, which not only explains the decidability results, but also leads to symbolic verification algorithms, such as antichain-based universality and inclusion checking for VPDAs [11].

There are, however, important languages that are parsable by deterministic pushdown automata, yet are not visibly pushdown. An important example are the arithmetic expressions with two binary operators, addition and multiplication, where multiplication takes precedence over addition. Most programming languages allow such expressions with implicit precedence relations between operators, instead of insisting on explicit parentheses to disambiguate. For this very purpose, Floyd introduced three elementary precedence relations between letters, namely, equals in precedence \( \doteq \), yields precedence \( \prec \), and takes precedence \( \succ \), which provide structure to words. He introduced the operator precedence languages (OPLs), a subclass of the context-free languages, where non-conflicting precedence relations between letters can be derived from the context-free grammar [33]. The ability to extract non-conflicting relations from the grammar provides a unique parse tree for each word. However, unlike for VPLs, a letter is not assigned to a unique stack operation, but will trigger “push” and “pop” operations depending on its precedence with respect to the adjacent letters. This allows OPLs to model not only arithmetic expressions, but also languages with exception handling capabilities, where a single closed parenthesis may close several open parentheses [1, 48].

The class of OPLs lies strictly between the VPLs and the deterministic context-free languages. Despite their extra expressive power, the OPLs enjoy the closure and decidability properties of the VPLs, and they even do so at the same cost in computational complexity: the class of OPLs is closed under all boolean and regular operations (union, intersection, complement, concatenation, reverse, and Kleene star) [20, 21]; their emptiness can be solved in \( \text{PTime} \) (it is \( \text{PTime} \)-hard for VPDAs), and universality and inclusion in \( \text{ExpTime} \) (they are \( \text{ExpTime} \)-hard for VPDAs) [43]. Moreover, OPLs admit a logical characterization in terms of a monadic second-order theory over words, as well as an operational characterization in terms of automata with a stack (called OPAs) [43]. In short, OPLs offer many of the benefits of the VPLs at no extra cost.

In this paper, we complete the picture by showing that OPLs also offer an algebraic characterization in form of a generalized Myhill-Nerode theorem. Specifically, we define a syntactic congruence relation \( \equiv_L \) for languages \( L \) such that \( \equiv_L \) has finitely many equivalence classes if and only if \( L \) is an OPL. Finite syntactic congruences provide a formalism-independent (i.e., grammar- and automaton-independent) definition for capturing the algebraic essence of
a class of languages. In addition to the regular languages (Myhill-Nerode) and the VPLs, such congruences have been given also for tree languages [37], for profinite languages [47], for omega-regular languages [4, 44], for sequential and rational transducers [15, 30]. Furthermore, such characterization results through syntactic congruences have been used to design determinization [2, 38], minimization [34, 41], and learning [12, 41, 46] algorithms.

Our contribution in this paper is twofold. Besides giving a finite congruence-based characterization of OPLs, we show how such a characterization can be used to obtain antichain-based verification algorithms, i.e., symbolic algorithms for checking the universality and inclusion of operator precedence automata (OPA). Checking language inclusion is the paradigmatic verification problem for any automaton-based specification formalism, but it is also computationally difficult: \textsc{PSPACE}-hard for finite automata, \textsc{ExpTime}-hard for VPDAs, undecidable for pushdown automata. This is why the verification community has devised and implemented symbolic algorithms, which avoid explicit subset constructions for determinization and complementation by manipulating symbolic representations of sets of states. For finite automata, the antichain-based algorithms have proven to be particularly efficient in practice: DWINA [29] outperforms MONA [40] for deciding WS1S formulae, ATC4VPA [11] outperforms VPAchecker [50] for deciding VPDAs inclusion, and Acacia [31] outperforms Lily [39] for LTL synthesis. They leverage a quasi-order on words to prune the search for counterexamples. Intuitively, whenever two words are candidates to contradict the inclusion between two given languages, and the words are related by the quasi-order at hand, the “greater” word can be discarded without compromising the completeness of the search. During symbolic fixpoint iteration, this “quasi-order reduction” yields a succinct representation of intermediate state sets. Based on our syntactic congruence, we show how to systematically compute a quasi-order that enables the antichain approach. Then, we provide the first antichain algorithm for checking language inclusion (and as a special case, universality) between OPAs. In fact, our antichain inclusion algorithm can take any suitable syntactic congruence over structured words (more precisely, any finite equivalence relation that is monotonic for structured words and saturates its language). The instantiation of the antichain algorithm with our syntactic congruence yields an \textsc{ExpTime} algorithm for the inclusion of OPAs, which is optimal in terms of enumeration complexity.

In summary, we generalize two of the most appealing features of the regular languages – the finite characterization by a syntactic congruence, and the antichain inclusion algorithm – to the important context-free subclass of operator precedence languages.

Overview. In Section 2, we define operator precedence alphabets and structured words. We present operator precedence grammars as originally defined by Floyd. We then define the operator precedence languages (OPLs) together with their automaton model (OPAs). Finally, we summarize the known closure and complexity results for OPLs and OPAs. In Section 3, we introduce the syntactic congruence that characterizes the class of OPLs. Subsection 3.1 proves that the syntactic congruence of every OPLs has finitely many equivalence classes, and Subsection 3.2 proves that every language whose syntactic congruence has finitely many equivalence classes is an OPL. In Section 4, we present our antichain inclusion algorithm. First, we introduce the notion of a language abstraction and prove that our syntactic congruence is a language abstraction of OPLs. We also present a quasi-order that relaxes the syntactic congruence while preserving the property of being a language abstraction. Then, we provide an antichain algorithm that decides the inclusion between automata whose languages have finite abstractions. We prove the correctness of our algorithm and establish its complexity on OPAs. In Section 5, we conclude with future directions.
Related Work. Operator precedence grammars and their languages were introduced by Floyd [33] with the motivation to construct efficient parsers. Inspired by Floyd’s work, Wirth and Weber [51] defined simple precedence grammars as the basis of an ALGOL-like language. The relation between these two models was studied in [32]. The properties of OPLs were studied in [17, 21]. Later, their relation with the class of VPLs was established in [20], their parallel parsing was explored in [5], and automata-theoretic and logical characterizations were provided in [43]. Recent contributions provide a model-checking algorithm for operator precedence automata [14], a generalization to a weighted model [27], and their application to verifying procedural programs with exceptions [48].

The OPLs form a class of structured context-free languages [45] that sits strictly between deterministic context-free languages and the VPLs [3, 19]. To the best of our knowledge, the OPLs constitute the largest known class that enjoys all desired closure and decidability properties. Several attempts have been made to move beyond this class, however, this often comes at the cost of losing some desirable property. For example, the locally chain-parsable languages are not closed under concatenation and Kleene star [18], and the higher-order OPLs with fixed order are not closed under concatenation [22]. Despite the fact that they are more powerful than the VPLs and enjoy all closure and decidability properties, the class of OPLs is not nearly as well studied. In particular, a finite syntactic congruence characterizing the VPLs was provided in [2]. An analogous result was missing for the OPLs until now.

The antichain algorithm for checking language inclusion was originally introduced for finite automata [52] and later extended to alternating finite automata [53]. The approach has been adapted to solve games with imperfect information [13], the inclusion of tree automata [8], the realizability of linear temporal logic [31], the satisfiability of quantified boolean formulas [9], the inclusion of ω-visibly pushdown automata [24], the satisfiability of weak monadic second-order logic [28], and the inclusion of Büchi automata [25, 26]. The antichain-based approach can be expressed as a complete abstract interpretation as it is captured by the framework introduced in [35, 36]. We provide the first antichain inclusion algorithm for OPLs, and the first generic method to construct an antichain algorithm from a finite syntactic congruence.

2 Operator Precedence Languages

We assume that the reader is familiar with formal language theory.

2.1 Operator Precedence Relations and Structured Words

Let $\Sigma$ be a finite alphabet. We refer by $\Sigma^*$ to the set of all words over $\Sigma$, by $\varepsilon$ to the empty word, and we let $\Sigma^* = \Sigma^* \setminus \{\varepsilon\}$. Given a word $w \in \Sigma^*$, we denote by $|w|$ its length, by $w^\circ$ its first letter, and by $w^\bullet$ its last letter. In particular $|\varepsilon| = 0$, $\varepsilon^\circ = \varepsilon$, and $\varepsilon^\bullet = \varepsilon$.

An operator precedence alphabet $\hat{\Sigma}$ is an alphabet $\Sigma$ equipped with the precedence relations $\preceq, \succeq, \cong$, given by a matrix (see Figure 1). Formally, for each ordered pair of letters $(a, b) \in \Sigma^2$, exactly one of the following holds:

- $a$ yields precedence to $b$, denoted $a \preceq b$,
- $a$ takes precedence over $b$, denoted $a \succeq b$,
- $a$ equals in precedence with $b$, denoted $a \cong b$.

1 In the literature, operator precedence matrices are defined over sets of precedence relations, leading then to notion of precedence conflict. We use the restriction to singletons because it covers the interesting part of the theory.
For $a, b \in \Sigma$, we write $a \succeq b$ iff $a \geq b$ or $a \preceq b$, and similarly $a \preceq b$ iff $a \leq b$ or $a \succeq b$. It is worth emphasizing that, despite their appearance, the operator precedence relations $\preceq, \preceq, \succeq$ and $\preceq$ are in general neither reflexive nor transitive. We extend the precedence relations with $\varepsilon$ such that $\varepsilon \prec a$, $a \succ \varepsilon$, and $\varepsilon \equiv \varepsilon$ for all $a \in \Sigma$.

Every word induces a sequence of precedences. For some words, this sequence corresponds to a chain \[^{[43]}\]. This is a building block of structured words.

\textbf{Definition 1 (chain).} Let $a_i \in \hat{\Sigma}$ and $u_i \in \hat{\Sigma}^+$ for all $i \in \mathbb{N}$, and let $n \geq 1$. A word $w = a_0a_1 \ldots a_{n+1} \in \hat{\Sigma} \cup \{\varepsilon\}$ is a simple chain when $a_0, a_{n+1} \in \hat{\Sigma}$ and $a_0 \preceq a_1 \succeq \cdots \succeq a_n \preceq a_{n+1}$. A word $w = a_0u_1a_1 \ldots a_nu_na_{n+1} \in \hat{\Sigma}^+$ is a composite chain when $a_0a_1 \ldots a_{n+1}$ is a simple chain and for all $0 \leq i \leq n$, either $a_iu_ia_{i+1}$ is a (simple or composite) chain or $u_i = \varepsilon$. A word $w$ is a chain when $w$ is a simple or a composite chain.

For all $x, y, z \in \hat{\Sigma}^+$, the predicate $x[y]z$ holds iff $(x^n)y(z^n)$ is a chain. Note that, if $x[y]z$ then $xyz \neq \varepsilon$.

\textbf{Example 2.} Let $\hat{\Sigma}$ be the operator precedence alphabet in Figure 1 that specifies the precedence relations for generating arithmetic expressions. The word $1 + 0 \times (1 + 1)$ is a simple chain because $1 \ll (1 \ll 0) \gg$. Moreover, the word $(1 + 1)$ is a composite chain because the words $(1+1)$ and $(1+1)$ are simple chains.

Next, we define a function that conservatively simplifies the structure of a given word.

\textbf{Definition 3 (collapsing function).} For a given operator precedence alphabet $\hat{\Sigma}$, its collapsing function $\lambda_{\hat{\Sigma}} : \hat{\Sigma}^* \rightarrow \hat{\Sigma}^*$ is defined inductively as follows: $\lambda_{\hat{\Sigma}}(w) = \lambda_{\hat{\Sigma}}(xz)$ if $w = xyz$ and $x[y]z$ for some $x, y, z \in \hat{\Sigma}^+$, and $\lambda_{\hat{\Sigma}}(w) = w$ if there is no such $x, y, z \in \hat{\Sigma}^+$. When $\hat{\Sigma}$ is clear from the context, we denote its collapsing function by $\lambda$.

For every $w \in \hat{\Sigma}$, observe that $\lambda(w)$ is in the following collapsed form: there exist $1 \leq i \leq j \leq n = |\lambda(w)|$ such that $a_1 \succeq \cdots \succeq a_{i-1} \succ a_i \preceq a_{i+1} \preceq \cdots \preceq a_j \succeq a_{j+1} \leq \cdots \leq a_n$.

\textbf{Example 4.} Let $\hat{\Sigma}$ be the operator precedence alphabet in Figure 1. Let $w = (1 + 0) \times (1 + 1)$ and observe that $\lambda(w) = (\varepsilon \times (\varepsilon \ast 0) \cdot (\varepsilon \ast (1 + 1)))$ since $\varepsilon \ast (1 + 0)^0$ and $\varepsilon \ast (1 + 1)^0$. Note also that $(\varepsilon \ast \varepsilon) \ll (\varepsilon \ast \varepsilon)$.

Note that the collapsed form is unique and allows us to generalize classical notions of well-nested words.
Definition 5 (structured words). Let $\hat{\Sigma}$ be an operator precedence alphabet. We define the following sets of words:

$$
\hat{\Sigma} = \{ w \in \hat{\Sigma}^* \mid \lambda(w) = 1 \}, \quad \Sigma = \{ w \in \hat{\Sigma}^* \mid \lambda(w) = 1 \}, \quad \Sigma_n = \{ w \in \hat{\Sigma}^* \mid \lambda(w) = 1 \}
$$

Looking back at the definition of collapsed form, one can verify for every word $w \in \hat{\Sigma}$ that $w \in \hat{\Sigma}^*_i$ iff $i = 1$, and $w \in \hat{\Sigma}^*_n$ iff $j = n$.

Example 6. Let $\hat{\Sigma}$ be the operator precedence alphabet in Figure 1. The word $+ \times \epsilon$ is in $\hat{\Sigma}^*_\Sigma$, the word $\epsilon \times \epsilon$ is in $\hat{\Sigma}^*_\Sigma$, and the word $\epsilon \epsilon \epsilon$ is in $\hat{\Sigma}^*_\Sigma$. Moreover, note that $+ \times \epsilon \epsilon \epsilon \epsilon \epsilon$ is non-ambiguous when for all $u \in L(\hat{\Sigma})$ admits a unique derivation tree.

2.2 Operator Precedence Grammars

A context-free grammar $G = (\Sigma, V, R, S)$ is tuple where $\Sigma$ is a finite set of terminal symbols, $V$ is a finite set of non-terminal symbols, $R \subseteq V \times (\Sigma \cup V)^*$ is a finite set of derivation rules, and $S \in V$ is the starting symbol. Given $\alpha, \beta \in (\Sigma \cup V)^*$, we write $\alpha \rightarrow \beta$ when $\beta$ can be derived from $\alpha$ with one rule, i.e., when there exists $(\alpha_2, \beta_2) \in R$, $\alpha = \alpha_1 \alpha_2 \beta_2$ and $\beta = \alpha_1 \beta_2 \beta_3$. Derivations using a sequence of rules are denoted by $\rightarrow^*$, the transitive closure of the relation $\rightarrow$. The language of $G$ is $L(G) = \{ w \in \Sigma^* \mid S \rightarrow^* w \}$. A derivation tree for $u \in L(G)$ is a tree over $\Sigma \cup V \cup \{ \epsilon \}$ such that the root is labeled by $S$, the concatenation of all leaves is $u$, and if a node is labeled by $a$ and its children labeled by $\beta_1, \ldots, \beta_k$ then $(a, \beta_1, \ldots, \beta_k) \in R$. A grammar is said to be non-ambiguous when for all $u \in L(G)$ admits a unique derivation tree.

Intuitively, an operator precedence grammar (OPG for short) is an unambiguous context-free grammar whose derivation trees comply with some operator precedence matrix. Formally, let $G = (\Sigma, V, R, S)$ be a context-free grammar and $A \in V$ be a non-terminal, and define the following sets of terminal symbols where $B \in V \cup \{ \epsilon \}$ and $\alpha \in (V \cup \Sigma)^*$:

$$
L_G(A) = \{ a \in \Sigma \mid A \rightarrow^* Baa \} \quad R_G(A) = \{ a \in \Sigma \mid A \rightarrow^* aab \}
$$

Given $a, b \in \Sigma$, we define the following operator precedence relations where $\alpha, \beta \in (V \cup \Sigma)^*$:

- $a <_G b$ if there exists a rule $A \rightarrow a\alpha\beta$ where $C \in V$ and $b \in L_G(C)$,
- $a >_G b$ if there exists a rule $A \rightarrow a\alpha\beta$ where $C \in V$ and $a \in R_G(C)$,
- $a \approx_G b$ if there exists a rule $A \rightarrow a\alpha\beta$ where $C \in V \cup \{ \epsilon \}$.

Finally, $G$ is an operator precedence grammar if and only if for all $a, b \in \Sigma$, we have that $|\{ \circ \in \{ <_G, \approx_G, >_G \} \mid a \circ b \}| \leq 1$.

Example 7. Let $G_{\text{arith}} = (\Sigma, V, R, A)$ be a context-free grammar over $\hat{\Sigma} = \{ +, \times, \{ \}, 0, 1 \}$ as in Figure 1 where $V = \{ A, B, C \}$ and $R$ contains the following rules:

$$
A \rightarrow A + B \mid B \quad B \rightarrow B \times C \mid C \quad C \rightarrow \{ A \} \mid 0 \mid 1
$$

The language $L(G_{\text{arith}})$ consists of valid arithmetic expressions with an implicit relation between terminal symbols: parentheses take precedence over multiplication, which takes precedence over addition [43]. The missing relations, replaced by $\cdot$ in the matrix of Figure 1, denote the precedence relations that cannot be encountered by the given grammar, so the chosen precedence relation does not matter. For example, $00$ and $\{ \}$ are not valid arithmetic expressions and cannot be generated by $G_{\text{arith}}$. We remark that the structures of derivation trees and chains share strong similarities as highlighted by Figure 2 and Figure 3.
2.3 Operator Precedence Automata

Intuitively, operator precedence automata are pushdown automata where stack operations are determined by the precedence relations between the next letter and the top of the stack.

Definition 8 (operator precedence automaton). An operator precedence automaton (OPA for short) over $\Sigma$ is a tuple $A = (Q, I, F, \Delta)$ where $Q$ is a finite set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is a set of accepting states, and $\Delta \subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma^+ \cup \{\perp\}))^2$ is the $\Sigma$-driven transition relation where $\Gamma = \Sigma \times Q$ is the stack alphabet and $\perp$ denotes the empty stack, meaning that, when $((s, a, \alpha), (t, b, \beta)) \in \Delta$ the following holds:

- If $\alpha = \perp$ or $\alpha = (q, a')\alpha'$ with $a' \preceq a$, then the input triggers a push stack-operation implying that $b = \varepsilon$ and $\beta = (s, a)\alpha$. We write $(s, \alpha) \xrightarrow{a} (t, \beta)$.
- If $\alpha = (q, a')\alpha'$ with $a' \lhd a$, then the input triggers a shift stack-operation implying that $b = \varepsilon$ and $\beta = (q, a)\alpha'$. We write $(s, \alpha) \xrightarrow{\alpha} (t, \beta)$.
- If $\alpha = (q, a')\alpha'$ with $a' \rhd a$, then the input triggers a pop stack-operation implying that $b = a$ and $\beta = \alpha'$. We write $(s, \alpha) \xrightarrow{a} (t, \beta)$.

Let $A$ be an OPA. A configuration of $A$ is a triplet $(q, u, \theta)$ where $q \in Q$ is the current stack state, $u \in \Sigma^*$ is the input suffix left to be read, and $\theta \in \Gamma^+ \cup \{\perp\}$ is the current stack. A run of $A$ is a finite sequence of configurations $((q_i, u_i, \theta_i))_{1 \leq i \leq n}$ for some $n \in \mathbb{N}$ such that, for all $1 \leq i \leq n$, the automaton fires (i) a push-transition $(\theta_{i-1}, q_{i-1}) \xrightarrow{a} (q_i, \theta_i)$ where $u_{i-1} = au_i$, (ii) a shift-transition $(\theta_{i-1}, q_{i-1}) \xrightarrow{\alpha} (q_i, \theta_i)$ where $u_{i-1} = au_i$, or (iii) a pop-transition $(\theta_{i-1}, q_{i-1}) \xrightarrow{a} (q_i, \theta_i)$ where $u_{i-1} = u_i \in \{au \mid u \in \Sigma^*\}$. We write $(s, u, \alpha) \xrightarrow{n} (t, v, \beta)$ when $(s, u, \alpha)(t, v, \beta)$ is a run, and let $(s, u, \alpha) \xrightarrow{\star} (t, v, \beta)$ be its reflexive transitive closure. For all $n \in \mathbb{N}$, we define the predicate $(s, u, \alpha) \xrightarrow{n} (t, v, \beta)$ inductively by $(s, u, \alpha) = (t, v, \beta)$ when $n = 0$ and by $\exists(q, w, \theta), (s, u, \alpha) \xrightarrow{n} (q, w, \theta) \xrightarrow{n-1} (t, v, \beta)$ otherwise.

The language of $A$ is defined by $L(A) = \{w \in \Sigma^* \mid q_0 \in I, q_f \in F, (q_0, w, \perp) \xrightarrow{\star} (q_f, \varepsilon, \perp)\}$. An OPA is deterministic when $|I| = 1$ and $\Delta$ is a function from $Q \times (\Gamma^+ \cup \{\perp\})$ to $Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma^+ \cup \{\perp\})$, and it is complete when from every configuration $(s, u, \theta)$ there exists a run that ends in $(t, \varepsilon, \perp)$ for some state $t \in Q$. For a given stack $\theta \in \Gamma^+ \cup \{\perp\}$, we define $\theta^+$ as the stack symbol at the top of $\theta$ if $\theta \in \Gamma^+$, and $\theta^+ = \varepsilon$ if $\theta = \perp$.

Definition 9 (operator precedence language). An operator precedence language (OPL for short) is a language recognized by some operator precedence automaton.

If $L$ is an OPL over the operator precedence alphabet $\hat{\Sigma}$, we say that $L$ is a $\hat{\Sigma}$-OPL.

Remark 10. The literature on OPLs often assumes the $\preceq$-acyclicity of operator precedence relations of the alphabet, i.e., that there is no $n \geq 1$ and $a_1, \ldots, a_n \in \Sigma$ with $a_1 \preceq \ldots \preceq a_n \preceq a_1$. This assumption is used to bound the right-hand side of OPG derivation rules, and find a key application for constructing an OPA that recognizes the language of a given OPA [43]. We omit this assumption since it is not needed for establishing the results on OPAs, including the construction of an OPA that recognizes the language of a given OPA.

Now, we present an OPA that recognizes valid arithmetic expressions.

Example 11. Recall the OPG of Example 7 generating arithmetic expressions over the operator precedence alphabet of Figure 1. In Figure 4, we show an OPA that recognizes the same language and an example of a computation.
2.4 Expressiveness and Decidability of Operator Precedence Languages

In this section, briefly summarize some known results about OPLs. First, we remark that OPLs are context-free languages as they are recognized by a subclass of pushdown automata.

► Theorem 12 (from [20]). Deterministic context-free languages strictly include OPLs.

The language $L = \{a^nba^n \mid n \geq 0\}$, which is a deterministic context-free language, separates the two classes. Indeed, it is not an OPL because while the first segment of $a^n$ must push to the stack (i.e., $a \prec a$), the last segment must pop (i.e., $a \succ a$), resulting in conflicting precedence relations. Next, we recall that OPLs enjoy the many closure properties.

► Theorem 13 (from [20, 21]). OPLs are closed under boolean operations, concatenation, Kleene star, reversal, prefixing, and suffixing.

The class of VPLs enjoy these closure as well. In fact, every VPL can be expressed as an OPL with an operator precedence alphabet designed as follows: internal characters and returns take precedence over any character; calls equal in precedence with returns, and they yield precedence to calls and internal characters.

► Theorem 14 (from [20]). OPLs strictly include visibly pushdown languages.

The language $L = \{a^nba^n \mid n \geq 1\} \cup \{c^{n}d^{n} \mid n \geq 1\} \cup \{e^{n}(bd)^{n} \mid n \geq 1\}$, which is an OPL due to their closure under union, separate the two classes. Indeed, for $L$ to be a VPL, the first set requires that $a$ is a call and $b$ is a return. Similarly, $c$ is a call and $d$ is a return due to the second set. However, the last set requires that at most one of $b$ and $d$ is a return, resulting in a contradiction. We also note that OPAs support determinization.

► Theorem 15 (from [43]). Every OPL can be recognized by a deterministic OPA.

Despite their expressive power, OPL remain decidable for the classical decision problems. In particular, OPAs enjoy the same order of complexity as VPDA for basic decision problems.

► Theorem 16 (from [42, 43]). The language emptiness is in PTIME-C for OPAs. The language inclusion, universality, and equivalence are in PTIME for deterministic OPAs and EXPTIME-C for nondeterministic OPAs.

► Remark 17. The membership problem is in PTIME for OPAs. Determining whether a given word $w$ is accepted by a given OPA $\mathcal{A}$ can be done in polynomial time by constructing an automaton $\mathcal{B}$ that accepts only $w$, constructing the intersection $\mathcal{C}$ of $\mathcal{A}$ and $\mathcal{B}$, and deciding the non-emptiness of $\mathcal{C}$.
3 A Finite Congruence for Operator Precedence Languages

This section introduces a congruence-based characterization of OPLs, similar to the Myhill-Nerode congruence for regular languages. We let $\hat{\Sigma}$ be an operator precedence alphabet throughout the section. A relation $\preceq$ over $\hat{\Sigma}^*$ is monotonic when $x \preceq y$ implies $uxv \preceq uyv$ for all $x, y, u, v \in \hat{\Sigma}^*$. Intuitively, monotonicity requires two words in relation to stay related while becoming embedded into some context that constructs a larger word. However, such a definition is not well suited for structured words as it does not follow how chains are constructed. Hence, we introduce a more restrictive notion than monotonicity.

Definition 18 (chain-monotonicity). A relation $\preceq$ over $\hat{\Sigma}^*$ is chain-monotonic when $x \preceq y$ implies $uu_v x v u \preceq uu_v y v u$ for all $x, y, u, v, u_0, v_0 \in \hat{\Sigma}^*$ such that $u_0 z v_0 \in \hat{\Sigma}^* \setminus \{\varepsilon\}$, $z \in \hat{\Sigma}^*$, and $u [u_0 z v_0]^n$ for each $z \in \{x, y\}$.

Chain-monotonicity requires two words in relation to stay related while being embedded into some context that construct larger structured words. This leads us to describe when two words agree on whether an embedding into a larger word forms a chain. For this, we introduce a relation that relates words that behave similarly with respect to the chain structure.

Definition 19 (chain equivalence). We define the chain equivalence $\approx$ over $\hat{\Sigma}^*$ as follows:

$$x \approx y \iff \bigwedge \{ x^i = y^i \land x^j = y^j \land \forall u, v, u_0, v_0 \in \hat{\Sigma}^* \setminus \{\varepsilon\}, (u_0 x^i v \in \hat{\Sigma}^* \setminus \{\varepsilon\} \land x^j v_0 \in \hat{\Sigma}^* \setminus \{\varepsilon\}) \Rightarrow ("u [u_0 x v]^n" \Leftrightarrow "u [u_0 y v]^n") \}$$

We observe that $\varepsilon$ is in relation with itself exclusively, i.e., $x = \varepsilon$ iff $\varepsilon \approx x$ iff $x \approx \varepsilon$. Consider a word $w \in \hat{\Sigma}^*$ for which $\lambda(w)$ is of the form $a_1 \ldots a_t b_1 \ldots b_m c_1 \ldots c_n$ for some $t, m, n \in \mathbb{N}$ such that $a_1 \geq \ldots \geq a_t > b_1 = \ldots = b_m < c_1 \leq \ldots \leq c_n$ where $a_1, b_j, c_k \in \Sigma$ for all $i, j, k$. We define the profile of $w$ as $P_w = (w^i, u^i, P_{u_i}, P_{w_i})$, where $P_{u_i} = \{a_1, b_1\} \cup \{a_{i+1} | a_i \geq a_{i+1} \leq 1 \leq i < t\}$ and $P_{w_i} = \{b_m, c_n\} \cup \{c_k | c_k < c_{k+1}, 1 \leq k < n\}$. There are at most $|\Sigma|^2 \times 2^{2|\Sigma| - 2} + 1$ profiles. We can show that two words with the same profile are chain equivalent, leading to the following proposition.

Proposition 20. $\approx$ is a chain-monotonic equivalence relation with finitely many classes.

Next, we introduce an equivalence relation that characterizes OPLs.

Definition 21 (syntactic congruence). Given $L \subseteq \hat{\Sigma}^*$, we define $\equiv_L$ as the following relation over $\hat{\Sigma}^*$:

$$x \equiv_L y \iff x \approx y \land \bigwedge \{ \forall u, v, u_0, v_0 \in \hat{\Sigma}^* \setminus \{\varepsilon\}, (u_0 x v \in \hat{\Sigma}^* \setminus \{\varepsilon\} \land x^j v_0 \in \hat{\Sigma}^* \setminus \{\varepsilon\} \Rightarrow ("u [u_0 x v]^n" \Leftrightarrow "u [u_0 y v]^n") \} \}$$

Let us demonstrate the syntactic congruence.

Example 22. Let $\Sigma = \{a, b\}$ and let $\hat{\Sigma}$ be the operator precedence alphabet with the relations $a \prec a$, $a \doteq b$, $b \succ a$, and $b \succ b$. Consider the language $L = \{a^n b^n | n \geq 1\}$.

There are 17 potential profiles for $\hat{\Sigma}$ in total. Although some of them cannot occur due to the precedence relations of $\hat{\Sigma}$, the remaining ones correspond to the equivalence classes of $\approx$. For example, $(a, a, \{a\}, \{a, b\})$ cannot occur since $b \succ a$, and $(a, b, \{a\}, \{b\})$ contains exactly the words in $L$ which are of the form $a^n b^n$ for some $n \geq 1$. For brevity, we only show how the syntactic congruence $\equiv_L$ refines the class of corresponding to $(a, a, \{a\}, \{a\})$ by splitting it into four subclasses. The profile $(a, a, \{a\}, \{a\})$ captures exactly the words of the form $w = a$ or $w$ is a$a$ where in each prefix of $a$ there are no more $b$’s than $a$’s. Notice that for such $w$, $\lambda(w)$ is of the form $(ab)^* a^+$, where $a^+ = \{a^n | n > 0\}$. Notice that for such $w$, $\lambda(w)$ is of the form $(ab)^* a^+$, where $a^+ = \{a^n | n > 0\}$.
We first argue that $a \not\equiv_L aa$ but $aa \equiv_L aa^n$ for all $n \geq 1$. Taking $u = v = u_0 = \varepsilon$ and $v_0 = b$, observe that the preconditions for the syntactic congruence are satisfied but $ab \notin L$ while $aab \notin L$, therefore $a \not\equiv_L aa$. Now, let $n \geq 2$, and consider the words $aa$ and $aa^n$. Intuitively, since there is no $x, y \in \hat{\Sigma}^*$ such that $xaay \in L$ and $xaa^n y \in L$, we show that whenever the preconditions for the congruence are satisfied, both longer words are out of $L$.

Given $u, v, u_0, v_0 \in \hat{\Sigma}^*$, such that $u_0 a \in \hat{\Sigma}_2^*$, $av_0 \in \hat{\Sigma}_2^*$, and $u[u_0 aa v_0]$, we assume towards contradiction that $u[a aa v_0] v \in L$. Since $u[a aa v_0] v \in L$ and $u_0 a \in \hat{\Sigma}_2^*$, we have $u_0 = \varepsilon$. Moreover, since $av_0 \in \hat{\Sigma}_2^*$, we have that $v_0$ is either of the form $a^* v b$ or $a^* b$. Consequently, $\lambda(u[a aa v_0])$ is $aa^*$ or $aa^* b$. This contradicts that $u[u_0 aa v_0]$ because $a < a$, and therefore $u[a aa v_0] v \notin L$. The same argument shows that $u[a aa] v_0 v \notin L$, implying that $aa \equiv_L aa^n$.

Similarly as above, we can show that $u \not\equiv_L v$ but $v \equiv_L w$ for all $u, v, w \in \hat{\Sigma}^*$ such that $\lambda(u) = (ab)^i a$, $\lambda(v) = (ab)^j aa$, and $\lambda(w) = (ab)^k aa^n$, where $n, i, j, k \geq 1$.

We now show that the syntactic congruence is chain-monotonic.

**Theorem 23.** For every $L \subseteq \hat{\Sigma}^*$, $\equiv_L$ is a chain-monotonic equivalence relation.

The main result of this section is the characterization theorem below. We prove each direction separately in Sections 3.1 and 3.2.

**Theorem 24.** A language $L$ is an OPL iff $\equiv_L$ admits finitely many equivalence classes.

### 3.1 Finiteness of the Syntactic Congruence

Let $\hat{\Sigma}$ be an operator precedence alphabet, $A = (Q, I, F, \Delta)$ be an OPA over $\hat{\Sigma}$, and $\ast \notin \Sigma$ be a fresh letter for which we extend the precedence relation with $a < \ast$ for all $a \in \Sigma$.

For every word $w \in \hat{\Sigma}^*$, we define the functions $f_w : Q \times (\Gamma \cup \{\bot\}) \to 2^Q$ and $\Phi_w : Q \times (\Gamma \cup \{\bot\}) \to 2^Q$ such that for all $q \in Q$ and all $\gamma \in \Gamma \cup \{\bot\}$, we have $f_w(q, \gamma) = \{q_w \in Q \mid \exists \gamma_w \in \Gamma^+ \cup \{\bot\}, (q, w, \gamma) \prec\ast (q_w, \ast, \gamma_w)\}$ and $\Phi_w(q, \gamma) = \{\gamma_w \in \Gamma^+ \cup \{\bot\} \mid \exists q_w \in Q, (q, w, \gamma) \prec\ast (q_w, \ast, \gamma_w)\}$. Intuitively, the states in $f_w(q, \gamma)$ and the stacks in $\Phi_w(q, \gamma)$ come from the configurations that $A$ can reach after reading $w$ from an initial state in $I$, but before triggering any pop-transition due to reaching the end of the word $w$.

Furthermore, for every $w \in \hat{\Sigma}^*$, we define the function $g_w : Q^2 \times (\Gamma \cup \{\bot\}) \to 2^Q$ such that for all $q_1, q_2 \in Q$ and all $\gamma \in \Gamma \cup \{\bot\}$ we have $g_w(q_1, q_2, \gamma) = \{p_w \in Q \mid \exists \gamma_w \in \Phi_w(q_1, \gamma), (q_2, e, \gamma) \prec\ast (p_w, e, \bot)\}$. Intuitively, $g_w(q_1, q_2, \gamma)$ is the set of states that $A$ can reach after triggering from $q_2$ the pop-transitions that empty the (unique) stack $\gamma_w$ in $\Phi_w(q_1, \gamma)$ that was generated by reading $w$ while moving from the state $q_1$ to some state in $f_w(q_1, \gamma)$.

Recall that for a given stack $\theta \in \Gamma^+ \cup \{\bot\}$, we denote by $\theta^\top$ the stack symbol at the top of $\theta$, which is $\varepsilon$ when $\theta = \bot$. Moreover, for a given set of stacks $\Theta \subseteq \Gamma^+ \cup \{\bot\}$, let us define $\Theta^\top = \{\theta^\top \mid \theta \in \Theta\}$. For the sequel, we define the following equivalence relation:

**Definition 25 (structural congruence).** Given an OPA $A = (Q, I, F, \Delta)$, we define the relation $\equiv_A$ over $\hat{\Sigma}^*$ as follows:

$$x \equiv_A y \iff x \approx y \land \forall x \land f_x = f_y \land g_x = g_y \land (\forall q \in Q, \forall \gamma \in \Gamma \cup \{\bot\}, (\Phi_x(q, \gamma))^\top = (\Phi_y(q, \gamma))^\top)$$

First, we show that the structural congruence of any OPA has a finite index.

**Lemma 26.** For every OPA $A$ with $n$ states and $m$ input letters, the structural congruence $\equiv_A$ has at most $O(m)^O(m \times n)^{O(1)}$ equivalence classes.

Then, we show that for any OPA the syntactic congruence of its language is coarser than its structural congruence, therefore has a finite index as well.
Lemma 27. For every OPA \( A \), the congruence \( \equiv_{L(A)} \) is coarser than the congruence \( \equiv_{A} \).

As a direct result of Lemmas 26 and 27 above, we obtain the following.

Corollary 28. For every \( L \subseteq \hat{\Sigma}^* \), if \( L \) is a \( \hat{\Sigma} \)-OPL then \( \equiv_L \) has finite index.

### 3.2 From the Syntactic Congruence to Operator Precedence Automata

Consider a language \( L \subseteq \hat{\Sigma}^* \) such that \( \equiv_L \) has finitely many equivalence classes. We construct a deterministic OPA that recognizes \( L \) and whose states are based on the equivalence classes of \( \equiv_L \). Given \( w \in \hat{\Sigma}^* \), we denote by \([w] \) its equivalence class with respect to \( \equiv_L \). We construct \( \mathcal{A} = (Q, \{q_0\}, F, \Delta) \) with the set of states \( Q = \{([u],[v]) \mid u,v \in \hat{\Sigma}^* \} \), the initial state \( q_0 = ([\varepsilon],[\varepsilon]) \), the set of accepting states \( F = \{([u],[w]) \mid w \in L \} \), and the \( \hat{\Sigma} \)-driven transition function \( \Delta : Q \times \hat{\Sigma} \times (\Gamma^+ \cup \{\bot\}) \rightarrow Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma^+ \cup \{\bot\}) \), where \( \Gamma = \Sigma \times Q \), is defined as follows: \( \Delta \) maps \(([u],[v]),a,((\Gamma^+ \cup \{\bot\}))\) to \(([a],[\varepsilon]),c,(a,([u],[v]),b,([v'],[v'])\{\theta}\) if \( b \preceq a \), it returns \(([uva],[\varepsilon]),c,(a,([u'],[v'])\{\theta}\) if \( b \succeq a \), and \(([([u'],[v']\{\theta}\),a,\theta\) if \( b \succeq a \). The soundness of our construction is given by the proof of the following lemma in Appendix.

Lemma 29. For every \( L \subseteq \hat{\Sigma}^* \), if \( \equiv_L \) has finite index then \( L \) is a \( \hat{\Sigma} \)-OPL.

### 4 Antichain-based Inclusion Checking

Considering two languages \( L_1 \) and \( L_2 \) given by some automata, the classical approach for deciding whether \( L_1 \subseteq L_2 \) holds is to first compute the complement \( T_2 \) of \( L_2 \), and then decide the emptiness of \( L_1 \cap T_2 \). The major drawback with this approach is that the complementation requires the determinization of the automaton denoting \( L_2 \). A way to avoid the determinization is to search among words of \( L_1 \) for a counterexample to \( L_1 \subseteq L_2 \). For this, a breadth-first search can be performed symbolically as a fixpoint iteration. In order to guarantee its termination, the search is equipped with a well quasi-order, and considers only words that are not subsumed, i.e., the minima of \( L_1 \) with respect to the quasi-order. It is known that well quasi-orders satisfy the finite basis property, i.e., all sets of words have finitely many minima. Our approach is inspired by [36] which, in the context of unstructured words, presents the antichain approach as a Galois connection, and observes that the upward closure of the quasi-order is a complete abstraction of concatenation according to the standard notion of completeness in abstract interpretation [16]. We identify, in the context of structured words, sufficient conditions on quasi-orders to enable the antichain approach, by defining the class of language abstraction quasi-orders (which satisfy the finite basis property). Further, we relax the syntactic congruence into a quasi-order that is a language abstraction of a given OPL. In particular, we prove that the syntactic congruence itself is a language abstraction for its language. Then, we design our inclusion algorithm based on a fixpoint characterization of OPLs, which allows us to iterate breadth-first over all words accepted by a given OPA. Once equipped with a language abstraction quasi-order, this fixpoint is guaranteed to terminate, thus to synthesize a finite set \( T \subseteq L_1 \) of membership queries for \( L_2 \) which suffices to decide whether \( L_1 \subseteq L_2 \) holds.

#### 4.1 Language Abstraction by Quasi-order

Let \( E \) be a set of elements and \( \preceq \) be a binary relation over \( E \). The relation \( \preceq \) is a quasi-order when it is reflexive and transitive. A quasi-order \( \preceq \) over \( E \) is decidable if for all \( x, y \in E \), determining whether \( x \preceq y \) holds is computable. Given a subset \( X \) of \( E \), we define its upward closure with respect to the quasi-order \( \preceq \) by \( \uparrow_X \) \( \{ e \in E \mid \exists x \in X, x \preceq e \} \). Given two
subsets \( X, Y \subseteq E \) the set \( X \) is a basis for \( Y \) with respect to \( \preceq \), denoted \( \mathcal{B}(X \preceq Y) \), whenever \( X \subseteq Y \) and \( \preceq | X = \preceq | Y \). The quasi-order \( \preceq \) is a well quasi-order if and only if for each set \( Y \subseteq E \) there exists a finite set \( X \subseteq E \) such that \( \mathcal{B}(X \preceq Y) \). This property on bases is also known as the finite basis property. Other equivalent definitions of well quasi-orders can be found in the literature [23], we will use the following two:

1. For every sequence \( \{e_i\}_{i \in \mathbb{N}} \) in \( E \), there exists \( i, j \in \mathbb{N} \) with \( i < j \) such that \( e_i \preceq e_j \).
2. There is no sequence \( \{X_i\}_{i \in \mathbb{N}} \) in \( 2^E \) such that \( \preceq | X_1 \preceq | X_2 \preceq \ldots \) holds.

Let \( L_1, L_2 \) be two languages. The main idea behind our inclusion algorithm is to compute a finite subset \( T \) of \( L_1 \), called a query-basis, such that \( T \subseteq L_2 \Leftrightarrow L_1 \subseteq L_2 \). Then, \( L_1 \subseteq L_2 \) holds if and only if each word of \( T \) belongs to \( L_2 \), which is checked via finitely many membership queries. The computation of a query-basis consists of collecting enough words of \( L_1 \) to obtain a finite basis \( T \) for \( L_1 \) with respect to a quasi-order \( \preceq \) that abstracts \( L_2 \). When \( \preceq \) is a well quasi-order, some basis is guaranteed to exist thanks to the finite basis property. To ensure the equivalence \( L_1 \subseteq L_2 \Leftrightarrow T \subseteq L_2 \) for any \( T \) such that \( \mathcal{B}(T \preceq L_1) \), a counterexample \( w \in L_1 \setminus L_2 \) can be discarded (not included in \( T \)), only if it there exists \( w_0 \in T \) such that \( w_0 \not\preceq w \) and \( w_0 \) is also a counterexample. Thus, we introduce the language saturation property asking a quasi-order \( \preceq \) to satisfy the following: for all \( w_0, w \in \hat{\Sigma}^* \) if \( w_0 \not\preceq w \) and \( w_0 \in L_2 \) then \( w \in L_2 \), or equivalently, \( \preceq | L_2 = L_2 \). Intuitively, language saturation ensures the completeness of the language abstraction with respect to the inclusion. Finally, to guarantee that the query-basis \( T \) is iteratively constructible with an effective fixpoint computation, the quasi-order \( \preceq \) must be both chain-monotonic and decidable. We now define the notion of language abstraction to identify the properties for a quasi-order over structured words that allow an effectively computable query-basis, as was done in [25, 36] in the context of Büchi automata for quasi-orders over unstructured infinite words.

**Definition 30 (language abstraction).** Let \( L \subseteq \hat{\Sigma}^* \). A quasi-order \( \preceq \) over \( \hat{\Sigma}^* \) is a language abstraction of \( L \) iff (1) it is decidable, (2) it is chain-monotonic, (3) it is a well quasi-order, and (4) it saturates \( L \).

In the next section, we provide an effective computation of a query-basis for an OPA, thanks to a quasi-order that abstracts its language.

**Example 31.** The operator precedence alphabet \( \hat{\Sigma}_{cr} \) of \( A \) and \( B \) from Figures 5 and 6 induces four families of words: (1) the words of \( \hat{\Sigma}_{cr}^{=}_c \) where every \( c \) matches an \( r \), (2) the words of \( \hat{\Sigma}_{cr}^= = \hat{\Sigma}_{cr}^= \setminus \hat{\Sigma}_{cr}^> \) where some \( c \) is pending for an \( r \) on its right, (3) the words of \( \hat{\Sigma}_{cr}^> = \hat{\Sigma}_{cr}^> \setminus \hat{\Sigma}_{cr}^= \) where some \( r \) is pending for a \( c \) on its left, and (4) all other words of \( \hat{\Sigma}_{cr}^= = \Sigma^* \setminus \left( \hat{\Sigma}_{cr}^ \cup \hat{\Sigma}_{cr}^\triangleright \right) \).
We focus on deciding whether $L(B)$ is a subset of $L(A)$ and suppose that we are given the quasi-order $\ll$ that is a language abstraction of $L(A)$. Additionally, we have that two words compare with $\ll$ only if they belong to the same family, and we have the following bases: $\mathcal{B}([c] \ll \mathcal{A} \Sigma^*_c)$, $\mathcal{B}([e] \ll \mathcal{A} \Sigma^*_e)$, $\mathcal{B}([r] \ll \mathcal{A} \Sigma^*_r)$, and $\mathcal{B}([rc] \ll \mathcal{A} \Sigma^*_{cr})$. We observe that $\ll$ saturates $L(A)$ since $\hat{\Sigma}^*_\varnothing \subseteq L(A)$ and $\hat{\Sigma}^*_c, \hat{\Sigma}^*_e, \hat{\Sigma}^*_{cr} \not\subseteq L(A)$.

Among the representatives $c, r, cr$ mentioned in Example 31 hold, that is, $c, r \not\in L(B)$. The set $T$ is a query-basis for deciding whether $L(B)$ is a subset of $L(A)$. In particular, $rc \in T$ witnesses that $L(B) \not\subseteq L(A)$.

Note that the syntactic congruence is a natural language abstraction of OPLs.

**Proposition 32.** For every OPL $L$, $\equiv_L$ is a language abstraction of $L$.

When the language to be abstracted is given by an OPA we are able to define a quasi-order, called structural quasi-order, that is based on the underlying structure of the automaton.

**Definition 33 (structural quasi-order).** Given an OPA $A = (Q, I, F, \Delta)$, we define the relation $\leq_A$ over $\hat{\Sigma}^*$ as follows:

\[
\begin{align*}
x \leq_A y & \iff x \approx y \land \forall q, q' \in Q, \forall \gamma \in \Gamma \cup \{\top\} \bigwedge \begin{cases} f_x(q, \gamma) \leq f_y(q, \gamma) \\ g_x(q, q', \gamma) \leq g_y(q, q', \gamma) \\ (\Phi_x(q, \gamma))^T \subseteq (\Phi_y(q, \gamma))^T \end{cases} 
\end{align*}
\]

**Remark 34.** For every OPA $A$, the quasi-order $\leq_A$ relaxes the congruence $\equiv_A$ from Section 3. For every OPA $A$, the quasi-order $\leq_A$ relaxes the congruence $\equiv_A$ from Section 3.

Note that, for every OPA $A$, the set $Q \times (\Gamma \cup \{\top\})$ is finite. Consequently, $\leq_A$ is computable, and it is a well quasi-order since there cannot exist an infinite sequence of incomparable elements, i.e., ($\triangleright$) holds.

**Proposition 35.** For every OPA $A$, $\leq_A$ is a computable chain-monotonic well quasi-order.

Next, we establish that structural quasi-orders saturate their languages.

**Lemma 36.** For every OPA $A$ and $w_1, w_2 \in \hat{\Sigma}^*$, if $w_1 \leq_A w_2$ and $w_1 \in L(A)$ then $w_2 \in L(A)$.

The following comes as a direct consequence of Proposition 35 and Lemma 36.

**Corollary 37.** For every OPA $A$, $\leq_A$ is a language abstraction of $L(A)$.

We continue Example 31, showing that the structural quasi-order agrees with the considered bases above.

**Example 38.** The quasi-order $\ll$ described in Example 31 agrees with the structural quasi-order $\leq_A$ of the OPA $A$ in Figure 5. Indeed, due to the constraint that two comparable words $x, y \in \hat{\Sigma}^*$ should be chain equivalent, i.e., $x \approx y$, the quasi-order $\leq_A$ compares only the words from the same families among $\hat{\Sigma}^*_c, \hat{\Sigma}^*_e, \hat{\Sigma}^*_{cr}$, and $\hat{\Sigma}^*_{cr}$. We also note that, for all words, adding a factor in $\hat{\Sigma}^*_{cr}$ cannot change the accessibility in $A$ since reading such a factor has no effect on the stack or the current state. Additionally, reading several $c$ in a row triggers a self loop and reading several $r$ is not possible in $A$. As a consequence, the base predicates mentioned in Example 31 hold, that is, $\mathcal{B}([c] \leq_A \hat{\Sigma}^*_c), \mathcal{B}([e] \leq_A \hat{\Sigma}^*_e), \mathcal{B}([r] \leq_A \hat{\Sigma}^*_r)$, and $\mathcal{B}([rc] \leq_A \hat{\Sigma}^*_{cr})$. Yet, we have that $cr \leq_A \epsilon$ because $(q_0, cr, \top) \prec^* (q_2, \epsilon, (c, q_0))$ but $(q_0, \epsilon, \top) \prec^* (q_2, \epsilon, (c, q_0))$. 

**ICALP 2023**
4.2 Fixpoint Characterization of Languages and Inclusion

In order to formulate our inclusion algorithm, it remains to give an effective computation of a query-basis. We do so through a fixpoint characterization of the languages recognized by OPAs. We introduce the function \( \text{Cat} \) to construct words that follow the runs of the given OPA. Iterating the \( \text{Cat} \) function \( n \in \mathbb{N} \) times captures all words of length up to \( n \), and the fixpoint of the iteration captures the entire language of a given OPA.

Let \( A = (Q, I, F, \Delta) \) be an OPA. Consider a vector of set of words \( \vec{X} \) that accesses its fields with two states \( s, t \in Q \), and three letters \( a, b, c \in \Sigma \). Intuitively, we aim at constructing \( \vec{X} \) iteratively such that, reading any \( w \in \vec{X}_{a,b,c}^{s,t} \) from the configuration \( (s, wc, \alpha) \) where \( \alpha = a \) allows reaching \( (t, c, \beta) \) where \( \beta = b \) in \( A \). We recall that \( \bot = \epsilon \). As the base case, we take \( \vec{X}_{a,b,c}^{s,t} = \epsilon \) when \( a = b \) and \( s = t \), otherwise \( \vec{X}_{a,b,c}^{s,t} = \emptyset \). Then, we introduce operations (more explicitly, functions from sets of words to sets of words) that use the transitivity of \( \triangleright \) in \( A \) to extend the sets of \( \vec{X} \). We first introduce:

\[
\text{CatShift}(\vec{X}_{a,b,c}^{s,t}) = \left\{ ub'v \mid a', b' \in \Sigma, q, s', t' \in Q, u \in \vec{X}_{a,a',b'}^{s,s'}, v \in \vec{X}_{b',c}^{t,t'} \right\}
\]

Essentially, \( \text{CatShift} \) adds \( ub'v \) to \( \vec{X}_{a,b,c}^{s,t} \) when some run over \( u \) can be appended with \( b' \) thanks to a shift-transition, and some run of \( v \) requires starting with \( b' \) at the top of the stack. Next, we introduce:

\[
\text{CatChain}(\vec{X}_{a,b,c}^{s,t}) = \left\{ ub'v \mid a', b', c' \in \Sigma, q, s', t' \in Q, u \in \vec{X}_{a,b,b'}^{s,q}, v \in \vec{X}_{c',c}^{t,t'} \right\}
\]

Intuitively, \( \text{CatChain} \) adds \( ub'v \) to \( \vec{X}_{a,b,c}^{s,t} \) when some run over \( u \) can be appended with \( b' \) thanks to a push-transition, and some run of \( v \) requires starting with \( b' \) at the top of the stack. Additionally, \( b' \) is guaranteed to be removed from the stack thanks to a pop-transition on the incoming letter \( c' \). Finally, we define:

\[
\text{Cat}(\vec{X}_{a,b,c}^{s,t}) = \vec{X}_{a,b,c}^{s,t} \cup \text{CatShift}(\vec{X}_{a,b,c}^{s,t}) \cup \text{CatChain}(\vec{X}_{a,b,c}^{s,t})
\]

Note that the function \( \text{Cat} \) never removes words from the sets of \( \vec{X} \), i.e., \( \vec{X}_{a,b,c}^{s,t} \subseteq \text{Cat}(\vec{X}_{a,b,c}^{s,t}) \).

Iterating the \( \text{Cat} \) function \( n \in \mathbb{N} \) times allows us to extend the sets of \( \vec{X} \) to words of length at most \( n \) that follow some run of \( A \). In particular, \( \text{Cat} \) characterizes the language of \( A \) by \( w \in L(A) \) if and only if \( w \in \text{Cat}^n(\vec{X}_{a,b,c}^{s,t}) \) for some \( q_f \in I \) and \( q_F \in F \). This is formalized by the following lemma.

\begin{lemma}
Let \( A = (Q, I, F, \Delta) \) be an OPA, and let \( \Gamma = \Sigma \times Q \). Considering \( \vec{U}_{a,b,c}^{s,t} = \epsilon \) when \( a = b \) and \( s = t \), otherwise \( \vec{U}_{a,b,c}^{s,t} = \emptyset \). The following holds for all \( n \in \mathbb{N} \):

\[
\text{Cat}^n(\vec{U}_{a,b,c}^{s,t}) = \left\{ u \mid (s, uc, \alpha) \overset{\omega}{\rightarrow}^n (t, c, \beta), |u| = n, \alpha \in \Theta_a, \beta \in \Theta_b, au \in \hat{\Sigma}_a^*, uc \in \hat{\Sigma}_c^*, \alpha = b \right\}
\]

where, for all \( a \in \hat{\Sigma} \), the set of stack symbols \( \Theta_a \subseteq \Gamma \cup \{ \bot \} \) is defined by \( \Theta_a = \{ \bot \} \) if \( a = \epsilon \), and \( \Theta_a = \{ (a, q) \mid q \in Q \} \) otherwise.
\end{lemma}

We continue Example 31, showing that \( \text{Cat} \) agrees with the considered query-basis.

\begin{example}
Let \( \vec{U}_{a,b,c}^{s,t} = \epsilon \) when \( a = b \) and \( s = t \), otherwise \( \vec{U}_{a,b,c}^{s,t} = \emptyset \). Thanks to Lemma 39, we have that \( L(B) = \text{Cat}^n(\vec{U}_{a,b,c}^{s,t}) \). First observe that \( c, r \notin \text{Cat}^n(\vec{U}_{a,b,c}^{s,t}) \). This comes from Lemma 39 and the fact that there is no run of \( B \) from \( p_0 \) to \( p_0 \) that reads a single letter. Next, we prove that \( cr, rc \in \text{Cat}^2(\vec{U}_{a,b,c}^{s,t}) \).
\end{example}
We show that $r \in \text{Cat}(\tilde{U}_{p_0,p_1})$ by CatChain. Indeed, we have $\varepsilon \in \tilde{U}_{p_0,p_1}$, $\varepsilon \in \tilde{U}_{p_0,p_1}$, and $(p_0, \perp) \xrightarrow{\varepsilon} (p_1, (r,p_1) \perp) \xrightarrow{r} (p_1, \perp)$. Then, $rc \in \text{Cat}(\tilde{U}_{p_0,p_1})$ by CatChain since $r \in \text{Cat}(\tilde{U}_{p_0,p_1})$, $\varepsilon \in \tilde{U}_{p_0,p_1}$, $(p_0, \perp) \xrightarrow{\varepsilon} (p_0, (c,p_1) \perp) \xrightarrow{c} (p_1, \perp)$.

We show that $r \in \text{Cat}(\tilde{U}_{p_0,p_1})$ by CatShift. Indeed, we have $\varepsilon \in \tilde{U}_{p_0,p_1}$, $\varepsilon \in \tilde{U}_{p_0,p_1}$, and $(p_1, (c,p) \perp) \xrightarrow{\varepsilon} (p_0, (r,p) \perp)$, for all $p \in \{p_0, p_1\}$. Then, $cr \in \text{Cat}(\tilde{U}_{p_0,p_1})$ by CatChain since $\varepsilon \in \tilde{U}_{p_0,p_1}$, $r \in \text{Cat}(\tilde{U}_{p_0,p_1})$, $\varepsilon \in \tilde{U}_{p_0,p_1}$, $(p_0, \perp) \xrightarrow{\varepsilon} (p_0, (c,p) \perp)$, and $(p_0, (r,p_0) \perp) \xrightarrow{c} (p_1, \perp)$.

The computation of a query-basis for deciding whether $L_1$ is a subset of $L_2$ consists of iterating Cat to collect enough words to obtain a vector of finite bases with respect to the quasi-order $\preceq$ that is a language abstraction of $L_2$. In other words, we search for $n \in \mathbb{N}$ such that $\text{Cat}^n(\tilde{X}^{a,b,c}_{s,t})$ is a basis for $\lim_{k \rightarrow \infty} \text{Cat}^k(\tilde{U}^{a,b,c}_{s,t})$ with respect to $\preceq$. The following lemma shows that when $\mathcal{B}(\text{Cat}^n(\tilde{X}^{a,b,c}_{s,t})) \preceq \text{Cat}^{n+1}(\tilde{X}^{a,b,c}_{s,t})$ holds for some $n \in \mathbb{N}$, then $\mathcal{B}(\text{Cat}^n(\tilde{X}^{a,b,c}_{s,t})) \preceq \lim_{k \rightarrow \infty} \text{Cat}^k(\tilde{X}^{a,b,c}_{s,t})$ holds also, as long as the used quasi-order is chain-monotonic.

**Lemma 41.** Let $\preceq$ be a chain-monotonic quasi-order over $\tilde{\Sigma}^*$. For every $A = (Q, I, F, \Delta)$ and $\tilde{X}, \tilde{Y}$ such that $\mathcal{B}(\text{Cat}^n(\tilde{X}^{a,b,c}_{s,t})) \preceq \text{Cat}^{n+1}(\tilde{X}^{a,b,c}_{s,t})$ holds for all $s,t \in Q$ and all $a, b, c \in \Sigma \cup \{\varepsilon\}$, we have $\mathcal{B}(\text{Cat}(\tilde{X}^{a,b,c}_{s,t})) \preceq \text{Cat}(\tilde{Y}^{a,b,c}_{s,t}))$ holds also for all $s,t \in Q$ and all $a, b, c \in \Sigma \cup \{\varepsilon\}$.

**Figure 7** Antichain inclusion algorithm.

Our inclusion algorithm is given in Figure 7. We can prove that it always terminates thanks to the finite base property of language abstractions. Additionally, its correctness is based on the following: Lemmas 39 and 41 ensure that the repeat-until loop computes a basis of the language $L_1$ given by an OPA while the language saturation ensures the completeness of this basis with respect to the inclusion problem.

**Theorem 42.** The algorithm from Figure 7 terminates and decides language inclusion.

We establish that our inclusion algorithm for OPAs is in ExpTime as a consequence of Lemma 26, Remark 34, the facts that the vector $\tilde{X}$ maintains polynomially many sets of words and the membership problem for OPAs is in PTIME (Remark 17). We recall that inclusion and universality are ExpTime-C for both OPLs and VPLs [3, 43].
Theorem 43. For all OPAs $A, B$ with respectively $n_A, n_B$ states and $m$ input letters, the inclusion algorithm from Figure 7 with $\leq_B$ as the language abstraction quasi-order decides if $L(A) \subseteq L(B)$ in time $O(m \times n_A)O(m \times n_B)^{O(1)}$.

5 Conclusion

We provided, for the first time, a syntactic congruence that characterizes operator precedence languages (OPLs) in the following exact sense: for any language $L$, the syntactic congruence has finitely many equivalence classes if and only if $L$ is an OPL. Second, we gave sufficient conditions for a quasi-order to yield an antichain algorithm for solving the universality and language inclusion problems for nondeterministic automata. These conditions are satisfied by our syntactic congruence, which, like any finite congruence, is monotonic for structured words (i.e., chain-monotonic) and saturates its language. This results in an exponential-time antichain algorithm for the inclusion of operator precedence automata (OPAs), which is the optimal worst-case complexity for the ExpTime-hard problem. This will allow efficient symbolic implementations of antichain algorithms to be extended to OPLs.

The possibility of future research directions regarding OPLs is still vast. One promising direction is to study OPAs from a runtime verification [6] perspective. For example, extending the runtime approaches for visibly pushdown automata [10, 49], one can study the monitor synthesis and right-universality problems for OPAs to establish them as an expressively powerful class of monitors. Also other methods developed for visibly pushdown automata may be generalizable to OPAs based on our syntactic congruence, such as learning algorithms [41].

While OPLs characterize the weakest known restrictions on stack operations which enable decidability of the inclusion problem, one may try to push the frontier of decidability by relaxing the restrictions on stack operations further. Investigating similar restrictions in the context of observability for counter automata can also provide new decidability results. For example, [7] shows that hardcoding the counter operations (increments and decrements) in the input letters yields decidable inclusion for one-counter automata. Another natural direction is to investigate quantitative versions of OPAs, for instance, through the addition of Presburger acceptance constraints, and to identify decidable fragments thereof [27].

References


