Regular Methods for Operator Precedence Languages

Thomas A. Henzinger
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Pavol Kebis
University of Oxford, Oxford, United Kingdom

Nicolas Mazzocchi[1]
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

N. Ege Saraç
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

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 ExpTime-hard language inclusion problem for OPLs in exponential time.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases operator precedence automata, syntactic congruence, antichain algorithm

Funding This work was supported in part by the ERC-2020-AdG 101020093.

Acknowledgements We thank Pierre Ganty for early discussions and the anonymous reviewers for their helpful comments.

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

[1] Corresponding author

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 parantheses to disambiguate. For this very purpose, Floyd introduced three elementary precedence relations between letters, namely, *equals in precedence* $\doteq$, *yields precedence* $\lessdot$, and *takes precedence* $\gtrdot$, 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 PTime (it is PTime-hard for VPDAs), and universality and inclusion in ExpTime (they are 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: PSpace-hard for finite automata, 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 congurence, 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 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 visibly pushdown automata [11], 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 Σ be a finite alphabet. We refer by Σ^* to the set of all words over Σ, by ε to the empty word, and we let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. Given a word $w \in \Sigma^*$, we denote by $|w|$ its length, by $w^{\lhd}$ its first letter, and by $w^{\rhd}$ its last letter. In particular $|\varepsilon| = 0$, $\varepsilon^{\lhd} = \varepsilon$, and $\varepsilon^{\rhd} = \varepsilon$.

An *operator precedence alphabet* $\widehat{\Sigma}$ is an alphabet Σ equipped with the precedence relations $\lessdot$, $\gtrdot$, $\doteq$, given by a matrix (see Figure 1). Formally, for each ordered pair of letters $(a, b) \in \Sigma^2$, exactly one[1] of the following holds:

- *a yields precedence to b*, denoted $a \lessdot b$,
- *a takes precedence over b*, denoted $a \gtrdot b$,
- *a equals in precedence with b*, denoted $a \doteq b$.

For $a, b \in \Sigma$, we write $a \geq b$ iff $a \gtrdot b$ or $a \doteq b$, and similarly $a \leqslant b$ iff $a \lessdot b$ or $a \doteq b$. It is worth emphasizing that, despite their appearance, the operator precedence relations $\lessdot$, $\leqslant$, $\gtrdot$, $\geq$ and $\doteq$ are in general neither reflexive nor transitive. We extend the precedence relations with ε such that $\varepsilon \lessdot a$, $a \gtrdot \varepsilon$, and $\varepsilon \doteq \varepsilon$ for all $a \in \Sigma$.

[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.

	$+$	$\times$	0	1	$(\!($	$)\!)$	ε
$+$	$\gtrdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\gtrdot$
$\times$	$\gtrdot$	$\gtrdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\gtrdot$
0	$\gtrdot$	$\gtrdot$	$\cdot$	$\cdot$	$\cdot$	$\gtrdot$	$\gtrdot$
1	$\gtrdot$	$\gtrdot$	$\cdot$	$\cdot$	$\cdot$	$\gtrdot$	$\gtrdot$
$(\!($	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$	$\gtrdot$
$)\!)$	$\gtrdot$	$\gtrdot$	$\cdot$	$\cdot$	$\cdot$	$\gtrdot$	$\gtrdot$
ε	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$

Figure 1 (left) Operator precedence matrix where parentheses take precedence over multiplication, which takes precedence over addition. The cells marked by $\cdot$ denote the irrelevant relations.

$$\begin{aligned}
\varepsilon \lessdot 1 \gtrdot + \lessdot 0 \gtrdot \times \lessdot (\!(\lessdot \mathbf{1} \gtrdot + \lessdot \mathbf{1} \gtrdot)\!) \gtrdot \varepsilon \qquad & 1 + 0 \times (\!(\mathbf{1} + \mathbf{1})\!) \\
\varepsilon \lessdot 1 \gtrdot + \lessdot 0 \gtrdot \times \lessdot (\!(\lessdot \mathbf{+} \gtrdot)\!) \gtrdot \varepsilon \qquad & 1 + 0 \times (\!(A \mathbf{+} B)\!) \\
\varepsilon \lessdot 1 \gtrdot + \lessdot \mathbf{0} \gtrdot \times \lessdot (\!(\doteq)\!) \gtrdot \varepsilon \qquad & 1 + \mathbf{0} \times (\!(A)\!) \\
\varepsilon \lessdot \mathbf{1} \gtrdot + \lessdot \mathbf{\times} \gtrdot \varepsilon \qquad & \mathbf{1} + B \mathbf{\times} C \\
\varepsilon \lessdot \mathbf{+} \gtrdot \varepsilon \qquad & A \mathbf{+} B \\
\varepsilon \doteq \varepsilon \qquad & A
\end{aligned}$$

Figure 2 (center) Computation of the collapsed from of $1 + 0 \times (\!(1 + 1)\!)$

Figure 3 (right) Derivation tree of the words $1 + 0 \times (\!(1 + 1)\!) \in L(G_{\text{arith}})$

Every word induces a sequence of precedences. For some words, this sequence corresponds to a *chain* [43], which is a building block of structured words.

▶ **Definition 1** (chain). *Let $a_i \in \widehat{\Sigma}$ and $u_i \in \widehat{\Sigma}^*$ for all $i \in \mathbb{N}$, and let $n \geq 1$. A word $w = a_0 a_1 \dots a_{n+1}$ is a* simple chain *when $a_0, a_{n+1} \in \widehat{\Sigma} \cup \{\varepsilon\}$ and $a_0 \lessdot a_1 \doteq a_2 \doteq \dots \doteq a_n \gtrdot a_{n+1}$. A word $w = a_0 u_0 a_1 u_1 \dots a_n u_n a_{n+1}$ is a* composite chain *when $a_0 a_1 \dots a_{n+1}$ is a simple chain and for all $0 \leq i \leq n$, either $a_i u_i a_{i+1}$ 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 $x, y, z \in \widehat{\Sigma}^*$, the predicate ${}^{x}[y]^{z}$ holds iff $(x^{\triangleright})y(z^{\triangleleft})$ is a chain. Note that, if ${}^{x}[y]^{z}$ then $xyz \neq \varepsilon$.

▶ **Example 2.** Let $\widehat{\Sigma}$ be the operator precedence alphabet in Figure 1 that specifies the precedence relations for generating arithmetic expressions. The word $(\!(()\!)\!)$ is a simple chain because $(\!(\lessdot (\!(\doteq)\!) \gtrdot)\!)$. Moreover, the word $(\!(1 + 1)\!)$ is a composite chain because the words $(\!(1+$, $+1)\!)$, and $(\!(+)\!)$ are simple chains.

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

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

For every $w \in \widehat{\Sigma}$, observe that $\lambda(w)$ is in the following collapsed form: there exist $1 \leq i \leq j \leq n = |\lambda(w)|$ such that $a_1 \gtrsim \dots \gtrsim a_{i-1} \gtrdot a_i \doteq a_{i+1} \doteq \dots \doteq a_j \lessdot a_{j+1} \lesssim \dots \lesssim a_n$.

▶ **Example 4.** Let $\widehat{\Sigma}$ be the operator precedence alphabet in Figure 1. Let $w = (\!(1{+}0)\!) \times (\!(1{+}1)\!)$ and observe that $\lambda(w) = (\!()\!) \times (\!()\!)$ since ${}^{(\!(}[1 + 0]^{)\!)}$ and ${}^{(\!(}[1 + 1]^{)\!)}$. Note also that $(\!(\doteq)\!) \gtrdot \times \lessdot (\!(\doteq)\!)$.

Note that the collapsed form is unique and allows us to generalize classical notions of well-nested words.

▶ **Definition 5** (structured words). *Let $\widehat{\Sigma}$ be an operator precedence alphabet. We define the following sets of words:*

$$\begin{aligned}
\widehat{\Sigma}^*_{\lesssim} &= \{w \in \widehat{\Sigma}^* \mid \lambda(w) = a_1 \dots a_n \text{ where } a_i \lesssim a_{i+1} \text{ for all } i, \text{ or } |\lambda(w)| \leq 1\} \\
\widehat{\Sigma}^*_{\gtrsim} &= \{w \in \widehat{\Sigma}^* \mid \lambda(w) = a_1 \dots a_n \text{ where } a_i \gtrsim a_{i+1} \text{ for all } i, \text{ or } |\lambda(w)| \leq 1\} \\
\widehat{\Sigma}^*_{\doteq} &= \{w \in \widehat{\Sigma}^* \mid \lambda(w) = a_1 \dots a_n \text{ where } a_i \doteq a_{i+1} \text{ for all } i, \text{ or } |\lambda(w)| \leq 1\} = \widehat{\Sigma}^*_{\lesssim} \cap \widehat{\Sigma}^*_{\gtrsim}
\end{aligned}$$

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

▶ **Example 6.** Let $\widehat{\Sigma}$ be the operator precedence alphabet in Figure 1. The word $+ \times (\!|\, |\!)$ is in $\widehat{\Sigma}^*_{\lessdot}$, the word $(\!|\, |\!) \times +$ is in $\widehat{\Sigma}^*_{\gtrdot}$, and the word $(\!|\, |\!)$ is in $\widehat{\Sigma}^*_{\doteq}$. Moreover, note that $+ \lessdot \times \lessdot (\!|\, \doteq \,|\!)$ and $(\!|\, \doteq \,|\!) \gtrdot \times \gtrdot +$.

2.2 Operator Precedence Grammars

A *context-free grammar* $G = (\Sigma, V, R, S)$ is tuple where Σ 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 $\alpha \to \beta$ when β can be derived from α with one rule, i.e., when there exists $(\alpha_2, \beta_2) \in R$, $\alpha = \alpha_1\alpha_2\alpha_3$ and $\beta = \alpha_1\beta_2\alpha_3$. Derivations using a sequence of rules are denoted by $\to^*$, the transitive closure of the relation $\to$. The language of G is $L(G) = \{w \in \Sigma^* \mid S \to^* w\}$. A derivation tree for $u \in L(G)$ is a tree over $\Sigma \cup V \cup \{\varepsilon\}$ such that the root is labeled by S, the concatenation of all leaves is u, and if a node is labeled by α and its children labeled by $\beta_1, \ldots, \beta_k$ then $(\alpha, \beta_1 \ldots \beta_k) \in R$. 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 $B \in V \cup \{\varepsilon\}$ and $\alpha \in (V \cup \Sigma)^*$:

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

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

- $a \lessdot_G b$ iff there exists a rule $A \to \alpha aC\beta$ where $C \in V$ and $b \in \mathcal{L}_G(C)$,
- $a \gtrdot_G b$ iff there exists a rule $A \to \alpha Cb\beta$ where $C \in V$ and $a \in \mathcal{R}_G(C)$,
- $a \doteq_G b$ iff there exists a rule $A \to \alpha aCb\beta$ where $C \in V \cup \{\varepsilon\}$.

Finally, G is an operator precedence grammar if and only if for all $a, b \in \Sigma$, we have that $|\{\odot \in \{\lessdot_G, \doteq_G, \gtrdot_G\} \mid a \odot b\}| \leq 1$.

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

$$A \to A + B \mid B \qquad B \to B \times C \mid C \qquad C \to (\!|A|\!) \mid 0 \mid 1$$

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_{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 $\widehat{\Sigma}$ is a tuple $\mathcal{A} = (Q, I, F, \Delta)$ 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 $\Delta \subseteq \big(Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma^+ \cup \{\bot\})\big)^2$ is the $\widehat{\Sigma}$-driven transition relation where $\Gamma = \Sigma \times Q$ is the stack alphabet and $\bot$ denotes the empty stack, meaning that, when $((s, a, \alpha), (t, b, \beta)) \in \Delta$ the following holds:*

- *If $\alpha = \bot$ or $\alpha = \langle q, a'\rangle\alpha'$ with $a' \lessdot a$, then the input triggers a* push *stack-operation implying that $b = \varepsilon$ and $\beta = \langle s, a\rangle\alpha$. We write $(s, \alpha) \xrightarrow{a} (t, \beta)$.*
- *If $\alpha = \langle q, a'\rangle\alpha'$ with $a' \doteq a$, then the input triggers a* shift *stack-operation implying that $b = \varepsilon$ and $\beta = \langle q, a\rangle\alpha'$. We write $(s, \alpha) \_\xrightarrow{a} (t, \beta)$.*
- *If $\alpha = \langle q, a'\rangle\alpha'$ with $a' \gtrdot a$, then the input triggers a* pop *stack-operation implying that $b = a$ and $\beta = \alpha'$. We write $(s, \alpha) \xRightarrow{a} (t, \beta)$.*

Let $\mathcal{A}$ be an OPA. A *configuration* of $\mathcal{A}$ is a triplet (q, u, θ) where $q \in Q$ is the current state, $u \in \Sigma^*$ is the input suffix left to be read, and $\theta \in \Gamma^+ \cup \{\bot\}$ is the current stack. A *run* of $\mathcal{A}$ is a finite sequence of configurations $((q_i, u_i, \theta_i))_{1 \le i \le n}$ for some $n \in \mathbb{N}$ such that, for all $1 \le i \le n$, the automaton fires (i) a push-transition $(q_{i-1}, \theta_{i-1}) \xrightarrow{a} (q_i, \theta_i)$ where $u_{i-1} = au_i$, (ii) a shift-transition $(q_{i-1}, \theta_{i-1}) \_\xrightarrow{a} (q_i, \theta_i)$ where $u_{i-1} = au_i$, or (iii) a pop-transition $(q_{i-1}, \theta_{i-1}) \xRightarrow{a} (q_i, \theta_i)$ where $u_{i-1} = u_i \in \{au \mid u \in \Sigma^*\}$. We write $(s, u, \alpha) \rightsquigarrow (t, v, \beta)$ when $(s, u, \alpha)(t, v, \beta)$ is a run, and let $(s, u, \alpha) \rightsquigarrow^* (t, v, \beta)$ be its reflexive transitive closure. For all $n \in \mathbb{N}$, we define the predicate $(s, u, \alpha) \rightsquigarrow^n (t, v, \beta)$ inductively by $(s, u, \alpha) = (t, v, \beta)$ when $n = 0$ and by $\exists (q, w, \theta), (s, u, \alpha) \rightsquigarrow (q, w, \theta) \rightsquigarrow^{n-1} (t, v, \beta)$ otherwise. The *language* of $\mathcal{A}$ is defined by $L(\mathcal{A}) = \{w \in \Sigma^* \mid q_0 \in I, q_F \in F, (q_0, w, \bot) \rightsquigarrow^* (q_F, \varepsilon, \bot)\}$. An OPA is *deterministic* when $|I| = 1$ and Δ is a function from $Q \times \Sigma \times (\Gamma^+ \cup \{\bot\})$ to $Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma^+ \cup \{\bot\})$, and it is *complete* when from every configuration (s, u, θ) there exists a run that ends in $(t, \varepsilon, \bot)$ for some state $t \in Q$. For a given stack $\theta \in \Gamma^+ \cup \{\bot\}$, we define $\theta^\top$ as the stack symbol at the top of θ if $\theta \in \Gamma^+$, and $\theta^\top = \varepsilon$ if $\theta = \bot$.

▶ **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 $\widehat{\Sigma}$, we say that L is a $\widehat{\Sigma}$-OPL.

▶ Remark 10. The literature on OPLs often assumes the $\doteq$-acyclicity of operator precedence relations of the alphabet, i.e., that there is no $n \ge 1$ and $a_1, \ldots, a_n \in \Sigma$ with $a_1 \doteq \ldots \doteq a_n \doteq a_1$. This assumption is used to bound the right-hand side of OPG derivation rules, and find a key application for constructing an OPG 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 OPG.

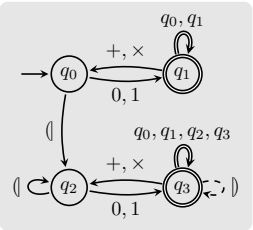
Now, we present an OPA that recognizes valid arithmetic expressions.

▶ **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.*



state	input	stack	state	input	stack
q_0	$1 \times (\!(0+1)\!)$	$\bot$	q_2	$1)\!)$	$\langle q_3,+\rangle\langle q_0,(\rangle\langle q_1,\times\rangle\bot$
q_1	$\times(\!(0+1)\!)$	$\langle q_0,1\rangle\bot$	q_3	$)\!)$	$\langle q_2,1\rangle\langle q_3,+\rangle\langle q_0,(\rangle\langle q_1,\times\rangle\bot$
q_1	$\times(\!(0+1)\!)$	$\bot$	q_3	$)\!)$	$\langle q_3,+\rangle\langle q_0,(\rangle\langle q_1,\times\rangle\bot$
q_0	$(\!(0+1)\!)$	$\langle q_1,\times\rangle\bot$	q_3	$)\!)$	$\langle q_0,(\rangle\langle q_1,\times\rangle\bot$
q_2	$0+1)\!)$	$\langle q_0,(\rangle\langle q_1,\times\rangle\bot$	q_3	ε	$\langle q_0,)\rangle\langle q_1,\times\rangle\bot$
q_3	$+1)\!)$	$\langle q_2,0\rangle\langle q_0,(\rangle\langle q_1,\times\rangle\bot$	q_3	ε	$\langle q_1,\times\rangle\bot$
q_3	$+1)\!)$	$\langle q_0,(\rangle\langle q_1,\times\rangle\bot$	q_3	ε	$\bot$

Figure 4 An OPA recognizing the arithmetic expressions generated by the OPG in Example 7 and its run on the input word $1 \times (\!(0+1)\!)$. Shift-, push-, and pop-transitions are respectively denoted by dashed, normal, and double arrows.

The language $L = \{a^n b a^n \mid n \geq 0\}$, 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., $a \lessdot a$), the last segment must pop (i.e., $a \gtrdot a$), 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 $L = \{a^n b^n \mid n \geq 1\} \cup \{c^n d^n \mid n \geq 1\} \cup \{e^n (bd)^n \mid n \geq 1\}$, 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 $\widehat{\Sigma}$ be an operator precedence alphabet throughout the section. A relation $\bowtie$ over $\widehat{\Sigma}^*$ is monotonic when $x \bowtie y$ implies $uxv \bowtie uyv$ for all $x, y, u, v \in \widehat{\Sigma}^*$. 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* $\bowtie$ *over* $\widehat{\Sigma}^*$ *is* chain-monotonic *when* $x \bowtie y$ *implies* $uu_0xv_0v \bowtie uu_0yv_0v$ *for all* $x, y, u, v, u_0, v_0 \in \widehat{\Sigma}^*$ *such that* $u_0z^\lhd \in \widehat{\Sigma}^*_{\geq}$, $z^\rhd v_0 \in \widehat{\Sigma}^*_{\leq}$, *and* ${}^u[u_0zv_0]^v$ *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* $\widehat{\Sigma}^*$ *as follows:*

$$x \approx y \iff \bigwedge \begin{cases} x^\lhd = y^\lhd \wedge x^\rhd = y^\rhd \\ \forall u, v, u_0, v_0 \in \widehat{\Sigma}^*, \left(u_0x^\lhd \in \widehat{\Sigma}^*_{\geq} \wedge x^\rhd v_0 \in \widehat{\Sigma}^*_{\leq}\right) \Rightarrow \left({}^u[u_0xv_0]^v \Leftrightarrow {}^u[u_0yv_0]^v\right) \end{cases}$$

We observe that ε is in relation with itself exclusively, i.e., $x = \varepsilon$ iff $\varepsilon \approx x$ iff $x \approx \varepsilon$. Consider a word $w \in \widehat{\Sigma}^+$ for which $\lambda(w)$ is of the form $a_1 \dots a_\ell b_1 \dots b_m c_1 \dots c_n$ for some $\ell, m, n \in \mathbb{N}$ such that $a_1 \geq \dots \geq a_\ell \gtrdot b_1 \doteq \dots \doteq b_m \lessdot c_1 \leq \dots \leq c_n$ where $a_i, b_j, c_k \in \Sigma$ for all i, j, k. We define the *profile* of w as $P_w = (w^\lhd, w^\rhd, P^\lhd_w, P^\rhd_w)$, where $P^\lhd_w = \{a_1, b_1\} \cup \{a_{i+1} \mid a_i \gtrdot a_{i+1}, 1 \leq i < \ell\}$ and $P^\rhd_w = \{b_m, c_n\} \cup \{c_k \mid c_k \lessdot c_{k+1}, 1 \leq k < n\}$. 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* $L \subseteq \widehat{\Sigma}^*$, *we define* $\equiv_L$ *as the following relation over* $\widehat{\Sigma}^*$*:*

$$x \equiv_L y \iff x \approx y \wedge \begin{cases} \forall u, v, u_0, v_0 \in \widehat{\Sigma}^*, \left(u_0x^\lhd \in \widehat{\Sigma}^*_{\geq} \wedge x^\rhd v_0 \in \widehat{\Sigma}^*_{\leq} \wedge {}^u[u_0xv_0]^v\right) \\ \qquad \Rightarrow \left(uu_0xv_0v \in L \Leftrightarrow uu_0yv_0v \in L\right) \end{cases}$$

Let us demonstrate the syntactic congruence.

▶ **Example 22.** Let $\Sigma = \{a, b\}$ and let $\widehat{\Sigma}$ be the operator precedence alphabet with the relations $a \lessdot a$, $a \doteq b$, $b \gtrdot a$, and $b \gtrdot b$. Consider the language $L = \{a^n b^n \mid n \geq 1\}$.

There are 17 potential profiles for $\widehat{\Sigma}$ in total. Although some of them cannot occur due to the precedence relations of $\widehat{\Sigma}$, the remaining ones correspond to the equivalence classes of $\approx$. For example, $(a, a, \{a\}, \{a, b\})$ cannot occur since $b \gtrdot a$, 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 $\approx$ 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 $w = aua$ where in each prefix of au there are no more b's than a's. Notice that for such w, $\lambda(w)$ is of the form $(ab)^*a^+$, where $a^+ = \{a^n \mid n > 0\}$.

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 $ab \in L$ 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 $x, y \in \widehat{\Sigma}^*$ 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 $u, v, u_0, v_0 \in \widehat{\Sigma}^*$ such that $u_0 a \in \widehat{\Sigma}^*_{\geq}$, $av_0 \in \widehat{\Sigma}^*_{\leq}$, and ${}^u[u_0 aav_0]^v$, we assume towards contradiction that $uu_0 aav_0 v \in L$. Since $uu_0 aav_0 v \in L$ and $u_0 a \in \widehat{\Sigma}^*_{\geq}$, we have $u_0 = \varepsilon$. Moreover, since $av_0 \in \widehat{\Sigma}^*_{\leq}$, we have that v_0 is either of the from a^* or a^*b. Consequently, $\lambda(u_0 aav_0)$ is aaa^* or aaa^*b. This contradicts that ${}^u[u_0 aav_0]^v$ because $a \lessdot a$, and therefore $uu_0 aav_0 v \notin L$. The same argument shows that $uu_0 aa^n v_0 v \notin L$, 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 $u, v, w \in \widehat{\Sigma}^*$ 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 $L \subseteq \widehat{\Sigma}^*$, $\equiv_L$ 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 $\widehat{\Sigma}$ be an operator precedence alphabet, $\mathcal{A} = (Q, I, F, \Delta)$ be an OPA over $\widehat{\Sigma}$, and $\star \notin \Sigma$ be a fresh letter for which we extend the precedence relation with $a \lessdot \star$ for all $a \in \Sigma$.

For every word $w \in \widehat{\Sigma}^*$, we define the functions $f_w \colon Q \times (\Gamma \cup \{\bot\}) \to 2^Q$ and $\Phi_w \colon Q \times (\Gamma \cup \{\bot\}) \to 2^{\Gamma^+ \cup \{\bot\}}$ such that for all $q \in Q$ and all $\gamma \in \Gamma \cup \{\bot\}$, we have $f_w(q, \gamma) = \{q_w \in Q \mid \exists \gamma_w \in \Gamma^+ \cup \{\bot\}, (q, w\star, \gamma) \rightsquigarrow^* (q_w, \star, \gamma_w)\}$ and $\Phi_w(q, \gamma) = \{\gamma_w \in \Gamma^+ \cup \{\bot\} \mid \exists q_w \in Q, (q, w\star, \gamma) \rightsquigarrow^* (q_w, \star, \gamma_w)\}$. Intuitively, the states in $f_w(q, \gamma)$ and the stacks in $\Phi_w(q, \gamma)$ come from the configurations that $\mathcal{A}$ 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 $w \in \widehat{\Sigma}^*$, we define the function $g_w \colon Q^2 \times (\Gamma \cup \{\bot\}) \to 2^Q$ such that for all $q_1, q_2 \in Q$ and all $\gamma \in \Gamma \cup \{\bot\}$ we have $g_w(q_1, q_2, \gamma) = \{p_w \in Q \mid \exists \gamma_w \in \Phi_w(q_1, \gamma), (q_2, \varepsilon, \gamma_w) \rightsquigarrow^* (p_w, \varepsilon, \bot)\}$. Intuitively, $g_w(q_1, q_2, \gamma)$ is the set of states that $\mathcal{A}$ 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 θ, which is ε 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 $\widehat{\Sigma}^*$ as follows:*

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

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

▶ **Lemma 26.** *For every OPA $\mathcal{A}$ with n states and m input letters, the structural congruence $\equiv_{\mathcal{A}}$ has at most $\mathcal{O}(m)^{\mathcal{O}(m \times n)^{\mathcal{O}(1)}}$ 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 $\mathcal{A}$, the congruence $\equiv_{L(\mathcal{A})}$ is coarser than the congruence $\equiv_{\mathcal{A}}$.*

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

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

3.2 From the Syntactic Congruence to Operator Precedence Automata

Consider a language $L \subseteq \widehat{\Sigma}^*$ 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 $w \in \widehat{\Sigma}^*$, 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 $Q = \{([u], [v]) \mid u, v \in \widehat{\Sigma}^*\}$, the initial state $q_0 = ([\varepsilon], [\varepsilon])$, the set of accepting states $F = \{([\varepsilon], [w]) \mid w \in L\}$, and the $\widehat{\Sigma}$-driven transition function $\Delta \colon Q \times \Sigma \times (\Gamma^+ \cup \{\bot\}) \to Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma^+ \cup \{\bot\})$, where $\Gamma = \Sigma \times Q$, is defined as follows: Δ maps $(([u], [v]), a, \langle b, ([u'], [v'])\rangle\theta)$ to $(([a], [\varepsilon]), \varepsilon, \langle a, ([u], [v])\rangle\langle b, ([u'], [v'])\rangle\theta)$ if $b \lessdot a$, it returns $(([uva], [\varepsilon]), \varepsilon, \langle a, ([u'], [v'])\rangle\theta)$ if $b \doteq a$, and $(([u'], [v'uv]), a, \theta)$ if $b \gtrdot a$. The soundness of our construction is given by the proof of the following lemma in Appendix.

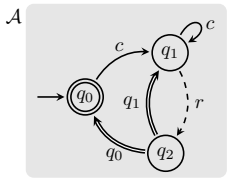
▶ **Lemma 29.** *For every $L \subseteq \widehat{\Sigma}^*$, if $\equiv_L$ has finite index then L is a $\widehat{\Sigma}$-OPL.*

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 $\overline{L}_2$ of L_2, and then decide the emptiness of $L_1 \cap \overline{L}_2$. 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 $\preccurlyeq$ be a binary relation over E. The relation $\preccurlyeq$ is a *quasi-order* when it is reflexive and transitive. A quasi-order $\preccurlyeq$ over E is *decidable* if for all $x, y \in E$, determining whether $x \preccurlyeq y$ holds is computable. Given a subset X of E, we define its *upward closure* with respect to the quasi-order $\preccurlyeq$ by ${}_{\preccurlyeq}\uparrow X = \{e \in E \mid \exists x \in X, x \preccurlyeq e\}$. Given two subsets $X, Y \subseteq E$ the set X is a *basis* for Y with respect to $\preccurlyeq$, denoted $\mathfrak{B}(X \preccurlyeq Y)$, whenever $X \subseteq Y$ and ${}_{\preccurlyeq}\uparrow X = {}_{\preccurlyeq}\uparrow Y$. The quasi-order $\preccurlyeq$ 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 $\mathfrak{B}(X \preccurlyeq Y)$. 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:
(†) For every sequence $\{e_i\}_{i \in \mathbb{N}}$ in E, there exists $i, j \in \mathbb{N}$ with $i < j$ such that $e_i \preccurlyeq e_j$.



$\widehat{\Sigma}_{cr}$	c	r	ε
c	$\lessdot$	$\doteq$	$\gtrdot$
r	$\gtrdot$	$\gtrdot$	$\gtrdot$
ε	$\lessdot$	$\lessdot$	$\doteq$

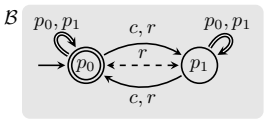


Figure 5 (left) OPA $\mathcal{A}$ over $\widehat{\Sigma}_{cr}$ recognizing the VPL of well-matched *call*/*return* words.

Figure 6 (right) OPA $\mathcal{B}$ over $\widehat{\Sigma}_{cr}$ recognizing the regular language of words of even length.

(‡) There is no sequence $\{X_i\}_{i\in\mathbb{N}}$ in 2^E such that $\preccurlyeq\restriction X_1 \subsetneq \preccurlyeq\restriction X_2 \subsetneq \dots$ 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 $\preccurlyeq$ that abstracts L_2. When $\preccurlyeq$ 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 $\mathfrak{B}(T \preccurlyeq L_1)$, 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 $w_0 \preccurlyeq w$ and w_0 is also a counterexample. Thus, we introduce the *language saturation* property asking a quasi-order $\preccurlyeq$ to satisfy the following: for all $w_0, w \in \widehat{\Sigma}^*$ if $w_0 \preccurlyeq w$ and $w_0 \in L_2$ then $w \in L_2$, or equivalently, $\preccurlyeq\restriction L_2 = L_2$. 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 $\preccurlyeq$ 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 $L \subseteq \widehat{\Sigma}^*$. A quasi-order $\preccurlyeq$ over $\widehat{\Sigma}^*$ 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 $\widehat{\Sigma}_{cr}$ of $\mathcal{A}$ and $\mathcal{B}$ from Figures 5 and 6 induces four families of words: (1) the words of $\widehat{\Sigma}^*_{\doteq}$ where every c matches an r, (2) the words of $\widehat{\Sigma}^*_{\lessdot} = \widehat{\Sigma}^*_{\leqslant} \setminus \widehat{\Sigma}^*_{\doteq}$ where some c is pending for an r on its right, (3) the words of $\widehat{\Sigma}^*_{\gtrdot} = \widehat{\Sigma}^*_{\geqslant} \setminus \widehat{\Sigma}^*_{\doteq}$ where some r is pending for a c on its left, and (4) all other words of $\widehat{\Sigma}^*_{\not\doteq} = \Sigma^* \setminus \left(\widehat{\Sigma}^*_{\leqslant} \cup \widehat{\Sigma}^*_{\geqslant}\right)$.

We focus on deciding whether $L(\mathcal{B})$ is a subset of $L(\mathcal{A})$ and suppose that we are given the quasi-order $\ll$ that is a language abstraction of $L(\mathcal{A})$. Additionally, we have that two words compare with $\ll$ only if they belong to the same family, and we have the following bases: $\mathfrak{B}(\{cr\} \ll \widehat{\Sigma}^*_{\doteq})$, $\mathfrak{B}(\{c\} \ll \widehat{\Sigma}^*_{\lessdot})$, $\mathfrak{B}(\{r\} \ll \widehat{\Sigma}^*_{\gtrdot})$, and $\mathfrak{B}(\{rc\} \ll \widehat{\Sigma}^*_{\not\doteq})$. We observe that $\ll$ saturates $L(\mathcal{A})$ since $\widehat{\Sigma}^*_{\doteq} \subseteq L(A)$ and $\widehat{\Sigma}^*_{\leqslant}, \widehat{\Sigma}^*_{\geqslant}, \widehat{\Sigma}^*_{\not\doteq} \not\subseteq L(\mathcal{A})$.

Among the representatives cr, c, r, and rc, we can construct the set $T = \{cr, rc\}$ since $c, r \notin L(\mathcal{B})$. The set T is a query-basis for deciding whether $L(\mathcal{B})$ is a subset of $L(\mathcal{A})$. In particular, $rc \in T$ witnesses that $L(\mathcal{B}) \not\subseteq L(\mathcal{A})$.

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 $\mathcal{A} = (Q, I, F, \Delta)$, we define the relation $\leqslant_{\mathcal{A}}$ over $\widehat{\Sigma}^*$ as follows:*

$$x \leqslant_{\mathcal{A}} y \iff x \approx y \wedge \forall q, q' \in Q, \forall \gamma \in \Gamma \cup \{\bot\} \bigwedge \begin{cases} f_x(q,\gamma) \subseteq f_y(q,\gamma) \\ g_x(q,q',\gamma) \subseteq g_y(q,q',\gamma) \\ (\Phi_x(q,\gamma))^\top \subseteq (\Phi_y(q,\gamma))^\top \end{cases}$$

▶ Remark 34. For every OPA $\mathcal{A}$, the quasi-order $\leqslant_{\mathcal{A}}$ relaxes the congruence $\equiv_A$ from Section 3. For every OPA $\mathcal{A}$, the quasi-order $\leqslant_{\mathcal{A}}$ relaxes the congruence $\equiv_A$ from Section 3.

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

▶ **Proposition 35.** *For every OPA $\mathcal{A}$, $\leqslant_{\mathcal{A}}$ is a computable chain-monotonic well quasi-order.*

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

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

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

▶ **Corollary 37.** *For every OPA $\mathcal{A}$, $\leqslant_{\mathcal{A}}$ is a language abstraction of $L(\mathcal{A})$.*

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 $\leqslant_{\mathcal{A}}$ of the OPA $\mathcal{A}$ in Figure 5. Indeed, due to the constraint that two comparable words $x, y \in \widehat{\Sigma}^*$ should be chain equivalent, i.e., $x \approx y$, the quasi-order $\leqslant_{\mathcal{A}}$ compares only the words from the same families among $\widehat{\Sigma}^*_{\doteq}, \widehat{\Sigma}^*_{\lessdot}, \widehat{\Sigma}^*_{\gtrdot}$, and $\widehat{\Sigma}^*_{\not\approx}$. We also note that, for all words, adding a factor in $\widehat{\Sigma}^*_{\doteq}$ cannot change the accessibility in $\mathcal{A}$ 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 $\mathcal{A}$. As a consequence, the base predicates mentioned in Example 31 hold, that is, $\mathfrak{B}(\{cr\} \leqslant_{\mathcal{A}} \widehat{\Sigma}^*_{\doteq})$, $\mathfrak{B}(\{c\} \leqslant_{\mathcal{A}} \widehat{\Sigma}^*_{\lessdot})$, $\mathfrak{B}(\{r\} \leqslant_{\mathcal{A}} \widehat{\Sigma}^*_{\gtrdot})$, and $\mathfrak{B}(\{rc\} \leqslant_{\mathcal{A}} \widehat{\Sigma}^*_{\not\approx})$. Yet, we have that $cr \leqslant_{\mathcal{A}} \varepsilon$ because $(q_0, cr, \bot) \rightsquigarrow^* (q_2, \varepsilon, \langle c, q_0\rangle)$ but $(q_0, \varepsilon, \bot) \not\rightsquigarrow^* (q_2, \varepsilon, \langle c, q_0\rangle)$.

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 `Cat` to construct words that follow the runs of the given OPA. Iterating the `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 $\mathcal{A} = (Q, I, F, \Delta)$ 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 $a, b, c \in \widehat{\Sigma} \cup \{\varepsilon\}$. 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, α) where $\alpha^\top = a$ allows reaching (t, c, β) where $\beta^\top = b$ in $\mathcal{A}$. We recall that $\bot^\top = \varepsilon$. As the base case, we take $\vec{X}^{a,b,c}_{s,t} = \varepsilon$ when $a = b$ and $s = t$, otherwise $\vec{X}^{a,b,c}_{s,t} = \varnothing$. Then, we introduce operations (more explicitly, functions from sets of words to sets of words) that use the transitivity of $\rightsquigarrow^*$ in $\mathcal{A}$ to extend the sets of $\vec{X}$. We first introduce:

$$\texttt{CatShift}(\vec{X}^{a,b,c}_{s,t}) = \left\{ ub'v \;\middle|\; \begin{array}{l} a', b' \in \Sigma,\, q, s', t' \in Q,\, u \in \vec{X}^{a,a',b'}_{s,s'},\, v \in \vec{X}^{b',b,c}_{t',t}, \\ (s', \langle a', q\rangle\bot) \xrightarrow{\;b'\;} (t', \langle b', q\rangle\bot) \end{array} \right\}$$

Essentially, `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:

$$\texttt{CatChain}(\vec{X}^{a,b,c}_{s,t}) = \left\{ ub'v \;\middle|\; \begin{array}{l} a', b', c' \in \Sigma,\, q, s', t' \in Q,\, u \in \vec{X}^{a,b,b'}_{s,q},\, v \in \vec{X}^{b',c',c}_{s',t'}, \\ b \lessdot b' \wedge (q, \bot) \xrightarrow{\;b'\;} (s', \langle b', q\rangle\bot) \wedge (t', \langle c', q\rangle\bot) \Longrightarrow (t, \bot) \end{array} \right\}$$

Intuitively, `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:

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

Note that the function `Cat` never removes words from the sets of $\vec{X}$, i.e., $\vec{X}^{a,b,c}_{s,t} \subseteq \texttt{Cat}(\vec{X}^{a,b,c}_{s,t})$. Iterating the `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 $\mathcal{A}$. In particular, `Cat` characterizes the language of $\mathcal{A}$ by $w \in L(\mathcal{A})$ if and only if $w \in \texttt{Cat}^*(\vec{X}^{\varepsilon,\varepsilon,\varepsilon}_{q_I,q_F})$ for some $q_I \in I$ and $q_F \in F$. This is formalized by the following lemma.

▶ **Lemma 39.** *Let $\mathcal{A} = (Q, I, F, \Delta)$ be an OPA, and let $\Gamma = \Sigma \times Q$. Considering $\vec{U}^{a,b,c}_{s,t} = \varepsilon$ when $a = b$ and $s = t$, otherwise $\vec{U}^{a,b,c}_{s,t} = \varnothing$. The following holds for all $n > 0$:*

$$\texttt{Cat}^n(\vec{U}^{a,b,c}_{s,t}) = \big\{u \mid (s, uc, \alpha) \rightsquigarrow^* (t, c, \beta), |u| = n, \alpha \in \Theta_a, \beta \in \Theta_b, au \in \widehat{\Sigma}^*_{\lessdot}, uc \in \widehat{\Sigma}^*_{\gtrdot}, u^\triangleright = b\big\}$$

where, for all $a \in \widehat{\Sigma}$, the set of stack symbols $\Theta_a \subseteq \Gamma \cup \{\bot\}$ is defined by $\Theta_a = \{\bot\}$ if $a = \varepsilon$, and $\Theta_a = \{\langle a, q\rangle \mid q \in Q\}$ otherwise.

We continue Example 31, showing that `Cat` agrees with the considered query-basis.

▶ **Example 40.** Let $\vec{U}^{a,b,c}_{s,t} = \varepsilon$ when $a = b$ and $s = t$, otherwise $\vec{U}^{a,b,c}_{s,t} = \varnothing$. Thanks to Lemma 39, we have that $L(\mathcal{B}) = \texttt{Cat}^*(\vec{U}^{\varepsilon,\varepsilon,\varepsilon}_{p_0,p_0})$. First observe that $c, r \notin \texttt{Cat}^*(\vec{U}^{\varepsilon,\varepsilon,\varepsilon}_{p_0,p_0})$. This comes from Lemma 39 and the fact that there is no run of $\mathcal{B}$ from p_0 to p_0 that reads a single letter. Next, we prove that $cr, rc \in \texttt{Cat}^2(\vec{U}^{\varepsilon,\varepsilon,\varepsilon}_{p_0,p_0})$.

We show that $r \in \texttt{Cat}(\vec{U}^{\varepsilon,\varepsilon,c}_{p_0,p_1})$ by `CatChain`. Indeed, we have $\varepsilon \in \vec{U}^{\varepsilon,\varepsilon,r}_{p_0,p_0}$, $\varepsilon \in \vec{U}^{r,r,c}_{p_1,p_1}$, $\varepsilon \lessdot r$, and $(p_0, \bot) \xrightarrow{\;r\;} (p_1, \langle r, p_1\rangle\bot) \xRightarrow{\;c\;} (p_1, \bot)$. Then, $rc \in \texttt{Cat}^2(\vec{U}^{\varepsilon,\varepsilon,\varepsilon}_{p_0,p_0})$ by `CatChain` since $r \in \texttt{Cat}(\vec{U}^{\varepsilon,\varepsilon,c}_{p_0,p_1})$, $\varepsilon \in \vec{U}^{c,c,\varepsilon}_{p_0,p_0}$, $\varepsilon \lessdot c$, and $(p_1, \bot) \xrightarrow{\;c\;} (p_0, \langle c, p_1\rangle\bot) \xRightarrow{\;\varepsilon\;} (p_1, \bot)$.

We show that $r \in \texttt{Cat}(\vec{U}^{c,r,\varepsilon}_{p_1,p_0})$ by `CatShift`. Indeed, we have $\varepsilon \in \vec{U}^{c,c,r}_{p_1,p_1}$, $\varepsilon \in \vec{U}^{r,r,\varepsilon}_{p_0,p_0}$, and $(p_1, \langle c, p\rangle\bot) \xrightarrow{\;r\;} (p_0, \langle r, p\rangle\bot)$, for all $p \in \{p_0, p_1\}$. Then, $cr \in \texttt{Cat}^2(\vec{U}^{\varepsilon,\varepsilon,\varepsilon}_{p_0,p_0})$ by `CatChain` since $\varepsilon \in \vec{U}^{\varepsilon,\varepsilon,c}_{p_0,p_0}$, $r \in \texttt{Cat}(\vec{U}^{c,r,\varepsilon}_{p_1,p_0})$, $\varepsilon \lessdot c$, $(p_0, \bot) \xrightarrow{\;c\;} (p_1, \langle c, p_0\rangle\bot)$, and $(p_0, \langle r, p_0\rangle) \xRightarrow{\;\varepsilon\;} (p_0, \bot)$.

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

▶ **Lemma 41.** *Let $\preccurlyeq$ be a chain-monotonic quasi-order over $\widehat{\Sigma}^*$. For every $A = (Q, I, F, \Delta)$ and $\vec{X}, \vec{Y}$ such that $\mathfrak{B}(\vec{X}^{a,b,c}_{s,t} \preccurlyeq \vec{Y}^{a,b,c}_{s,t})$ holds for all $s, t \in Q$ and all $a, b, c \in \Sigma \cup \{\varepsilon\}$, we have $\mathfrak{B}(\mathtt{Cat}(\vec{X}^{a,b,c}_{s,t}) \preccurlyeq \mathtt{Cat}(\vec{Y}^{a,b,c}_{s,t}))$ holds also for all $s, t \in Q$ and all $a, b, c \in \Sigma \cup \{\varepsilon\}$.*

Input: an OPL L_1 given by the OPA (Q, I, F, Δ)
Input: a language L_2 with a procedure deciding if $w \in L_2$
Input: a quasi-order $\preccurlyeq$ that is a language abstraction of L_2
Output: Returns `ok` if $L_1 \subseteq L_2$ and `ko` otherwise

1 **Function:**
2 **let** $\vec{U}$ **as** $\vec{U}^{a,b,c}_{s,t} := \varepsilon$ **if** $a = b \wedge s = t$ **else** $\vec{U}^{a,b,c}_{s,t} := \varnothing$
3 $\vec{X} := \vec{U}$
4 **repeat**
5 **let** $\vec{X}$ **as** $\vec{X}^{a,b,c}_{s,t} := \texttt{Cat}(\vec{X}^{a,b,c}_{s,t})$
6 **until** $\mathfrak{B}(\vec{X}^{a,b,c}_{s,t} \preccurlyeq \mathtt{Cat}(\vec{X}^{a,b,c}_{s,t}))$ *for all* $s, t \in Q$ *and all* $a, b, c \in \Sigma \cup \{\varepsilon\}$
7 **for each** $(q_I, q_F) \in I \times F$ **do**
8 **for each** $w \in \vec{X}^{\varepsilon,\varepsilon,\varepsilon}_{q_I,q_F}$ **do**
9 **if** $w \notin L_2$ **then return** ko
10 **return** ok

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 $\vec{X}$ 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 $\mathcal{A}, \mathcal{B}$ with respectively $n_\mathcal{A}, n_\mathcal{B}$ states and m input letters, the inclusion algorithm from Figure 7 with $\leqslant_\mathcal{B}$ as the language abstraction quasi-order decides if $L(\mathcal{A}) \subseteq L(\mathcal{B})$ in time $\mathcal{O}(m \times n_\mathcal{A})^{\mathcal{O}(m \times n_\mathcal{B})^{\mathcal{O}(1)}}$.*

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

1 Rajeev Alur and Dana Fisman. Colored nested words. *Formal Methods Syst. Des.*, 58(3):347–374, 2021. doi:10.1007/s10703-021-00384-2.

2 Rajeev Alur, Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan. Congruences for visibly pushdown languages. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium,/pdf/0907.2130.pdf ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 1102–1114. Springer, 2005. doi:10.1007/11523468_89.

3 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM, 2004. doi:10.1145/1007352.1007390.

4 André Arnold. A syntactic congruence for rational omega-language. *Theor. Comput. Sci.*, 39:333–335, 1985. doi:10.1016/0304-3975(85)90148-3.

5 Alessandro Barenghi, Stefano Crespi-Reghizzi, Dino Mandrioli, and Matteo Pradella. Parallel parsing of operator precedence grammars. *Inf. Process. Lett.*, 113(7):245–249, 2013. doi:10.1016/j.ipl.2013.01.008.

6 Ezio Bartocci and Yliès Falcone, editors. *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*. Springer, 2018. doi:10.1007/978-3-319-75632-5.

7 Benedikt Bollig. One-counter automata with counter observability. In Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen, editors, *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India*, volume 65 of *LIPIcs*, pages 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.20.

8 Ahmed Bouajjani, Peter Habermehl, Lukás Holík, Tayssir Touili, and Tomás Vojnar. Antichain-based universality and inclusion testing over nondeterministic finite tree automata. In Oscar H. Ibarra and Bala Ravikumar, editors, *Implementation and Applications of Automata, 13th*

International Conference, CIAA 2008, San Francisco, California, USA, July 21-24, 2008. Proceedings, volume 5148 of *Lecture Notes in Computer Science*, pages 57–67. Springer, 2008. doi:10.1007/978-3-540-70844-5_7.

9 Thomas Brihaye, Véronique Bruyère, Laurent Doyen, Marc Ducobu, and Jean-François Raskin. Antichain-based QBF solving. In Tevfik Bultan and Pao-Ann Hsiung, editors, *Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings*, volume 6996 of *Lecture Notes in Computer Science*, pages 183–197. Springer, 2011. doi:10.1007/978-3-642-24372-1_14.

10 Véronique Bruyère, Marc Ducobu, and Olivier Gauwin. Right-universality of visibly pushdown automata. In Axel Legay and Saddek Bensalem, editors, *Runtime Verification - 4th International Conference, RV 2013, Rennes, France, September 24-27, 2013. Proceedings*, volume 8174 of *Lecture Notes in Computer Science*, pages 76–93. Springer, 2013. doi:10.1007/978-3-642-40787-1_5.

11 Véronique Bruyère, Marc Ducobu, and Olivier Gauwin. Visibly pushdown automata: Universality and inclusion via antichains. In Adrian-Horia Dediu, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 7th International Conference, LATA 2013, Bilbao, Spain, April 2-5, 2013. Proceedings*, volume 7810 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 2013. doi:10.1007/978-3-642-37064-9_18.

12 Véronique Bruyère, Guillermo A. Pérez, and Gaëtan Staquet. Learning realtime one-counter automata. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 244–262. Springer, 2022. doi:10.1007/978-3-030-99524-9_13.

13 Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Algorithms for omega-regular games with imperfect information' . In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, volume 4207 of *Lecture Notes in Computer Science*, pages 287–302. Springer, 2006. doi:10.1007/11874683_19.

14 Michele Chiari, Dino Mandrioli, and Matteo Pradella. Operator precedence temporal logic and model checking. *Theor. Comput. Sci.*, 848:47–81, 2020. doi:10.1016/j.tcs.2020.08.034.

15 Christian Choffrut. Minimizing subsequential transducers: a survey. *Theor. Comput. Sci.*, 292(1):131–143, 2003. doi:10.1016/S0304-3975(01)00219-5.

16 Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Robert M. Graham, Michael A. Harrison, and Ravi Sethi, editors, *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, pages 238–252. ACM, 1977. doi:10.1145/512950.512973.

17 Stefano Crespi-Reghizzi, Giovanni Guida, and Dino Mandrioli. Operator precedence grammars and the noncounting property. *SIAM J. Comput.*, 10(1):174–191, 1981. doi:10.1137/0210013.

18 Stefano Crespi-Reghizzi, Violetta Lonati, Dino Mandrioli, and Matteo Pradella. Toward a theory of input-driven locally parsable languages. *Theor. Comput. Sci.*, 658:105–121, 2017. doi:10.1016/j.tcs.2016.05.003.

19 Stefano Crespi-Reghizzi and Dino Mandrioli. Algebraic properties of structured context-free languages: old approaches and novel developments. *CoRR*, abs/0907.2130, 2009. URL: http://arxiv.org/abs/0907.2130, arXiv:0907.2130.

20 Stefano Crespi-Reghizzi and Dino Mandrioli. Operator precedence and the visibly pushdown property. *J. Comput. Syst. Sci.*, 78(6):1837–1867, 2012. doi:10.1016/j.jcss.2011.12.006.

21 Stefano Crespi-Reghizzi, Dino Mandrioli, and David F. Martin. Algebraic properties of operator precedence languages. *Inf. Control.*, 37(2):115–133, 1978. doi:10.1016/S0019-9958(78)90474-6.

22 Stefano Crespi-Reghizzi and Matteo Pradella. Beyond operator-precedence grammars and languages. *J. Comput. Syst. Sci.*, 113:18–41, 2020. doi:10.1016/j.jcss.2020.04.006.

23 Aldo de Luca and Stefano Varricchio. Well quasi-orders and regular languages. *Acta Informatica*, 31(6):539–557, 1994. doi:10.1007/BF01213206.

24 Kyveli Doveri, Pierre Ganty, and Luka Hadži-Đokić. Antichains Algorithms for the Inclusion Problem Between ω-VPL. In *TACAS'23: Proc. 29th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 13993 of *LNCS*. Springer, 2023.

25 Kyveli Doveri, Pierre Ganty, and Nicolas Mazzocchi. Forq-based language inclusion formal testing. In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II*, volume 13372 of *Lecture Notes in Computer Science*, pages 109–129. Springer, 2022. doi:10.1007/978-3-031-13188-2_6.

26 Kyveli Doveri, Pierre Ganty, Francesco Parolini, and Francesco Ranzato. Inclusion testing of büchi automata based on well-quasiorders. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 3:1–3:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CONCUR.2021.3.

27 Manfred Droste, Stefan Dück, Dino Mandrioli, and Matteo Pradella. Weighted operator precedence languages. *Inf. Comput.*, 282:104658, 2022. doi:10.1016/j.ic.2020.104658.

28 Tomás Fiedor, Lukás Holík, Ondrej Lengál, and Tomás Vojnar. Nested antichains for WS1S. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9035 of *Lecture Notes in Computer Science*, pages 658–674. Springer, 2015. doi:10.1007/978-3-662-46681-0_59.

29 Tomás Fiedor, Lukás Holík, Ondrej Lengál, and Tomás Vojnar. Nested antichains for WS1S. *Acta Informatica*, 56(3):205–228, 2019. doi:10.1007/s00236-018-0331-z.

30 Emmanuel Filiot, Olivier Gauwin, and Nathan Lhote. Logical and algebraic characterizations of rational transductions. *Log. Methods Comput. Sci.*, 15(4), 2019. doi:10.23638/LMCS-15(4:16)2019.

31 Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. An antichain algorithm for LTL realizability. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, volume 5643 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2009. doi:10.1007/978-3-642-02658-4_22.

32 Michael J. Fischer. Some properties of precedence languages. In Patrick C. Fischer, Seymour Ginsburg, and Michael A. Harrison, editors, *Proceedings of the 1st Annual ACM Symposium on Theory of Computing, May 5-7, 1969, Marina del Rey, CA, USA*, pages 181–190. ACM, 1969. doi:10.1145/800169.805432.

33 Robert W. Floyd. Syntactic analysis and operator precedence. *J. ACM*, 10(3):316–333, 1963. doi:10.1145/321172.321179.

34 Pierre Ganty, Elena Gutiérrez, and Pedro Valero. A congruence-based perspective on automata minimization algorithms. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 77:1–77:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.77.

35 Pierre Ganty, Francesco Ranzato, and Pedro Valero. Language inclusion algorithms as complete abstract interpretations. In Bor-Yuh Evan Chang, editor, *Static Analysis - 26th International Symposium, SAS 2019, Porto, Portugal, October 8-11, 2019, Proceedings*, volume 11822 of *Lecture Notes in Computer Science*, pages 140–161. Springer, 2019. doi:10.1007/978-3-030-32304-2_8.

36 Pierre Ganty, Francesco Ranzato, and Pedro Valero. Complete abstractions for checking language inclusion. *ACM Trans. Comput. Log.*, 22(4):22:1–22:40, 2021. doi:10.1145/3462673.

37 Ferenc Gécseg. Classes of tree languages determined by classes of monoids. *Int. J. Found. Comput. Sci.*, 18(6):1237–1246, 2007. doi:10.1142/S0129054107005285.

38 Jelena Ignjatovic, Miroslav Ciric, and Stojan Bogdanovic. Determinization of fuzzy automata with membership values in complete residuated lattices. *Inf. Sci.*, 178(1):164–180, 2008. doi:10.1016/j.ins.2007.08.003.

39 Barbara Jobstmann and Roderick Bloem. Optimizations for LTL synthesis. In *Formal Methods in Computer-Aided Design, 6th International Conference, FMCAD 2006, San Jose, California, USA, November 12-16, 2006, Proceedings*, pages 117–124. IEEE Computer Society, 2006. doi:10.1109/FMCAD.2006.22.

40 Nils Klarlund, Anders Møller, and Michael I. Schwartzbach. MONA implementation secrets. *Int. J. Found. Comput. Sci.*, 13(4):571–586, 2002. doi:10.1142/S012905410200128X.

41 Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan. Minimization, learning, and conformance testing of boolean programs. In Christel Baier and Holger Hermanns, editors, *CONCUR 2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006, Proceedings*, volume 4137 of *Lecture Notes in Computer Science*, pages 203–217. Springer, 2006. doi:10.1007/11817949_14.

42 Martin Lange. P-hardness of the emptiness problem for visibly pushdown languages. *Inf. Process. Lett.*, 111(7):338–341, 2011. doi:10.1016/j.ipl.2010.12.013.

43 Violetta Lonati, Dino Mandrioli, Federica Panella, and Matteo Pradella. Operator precedence languages: Their automata-theoretic and logic characterization. *SIAM J. Comput.*, 44(4):1026–1088, 2015. doi:10.1137/140978818.

44 Oded Maler and Ludwig Staiger. On syntactic congruences for omega-languages. *Theor. Comput. Sci.*, 183(1):93–112, 1997. doi:10.1016/S0304-3975(96)00312-X.

45 Dino Mandrioli and Matteo Pradella. Generalizing input-driven languages: Theoretical and practical benefits. *Comput. Sci. Rev.*, 27:61–87, 2018. doi:10.1016/j.cosrev.2017.12.001.

46 Jakub Michaliszyn and Jan Otop. Learning deterministic visibly pushdown automata under accessible stack. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22-26, 2022, Vienna, Austria*, volume 241 of *LIPIcs*, pages 74:1–74:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.MFCS.2022.74.

47 Jean-Eric Pin. Profinite methods in automata theory. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, volume 3 of *LIPIcs*, pages 31–50. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2009. doi:10.4230/LIPIcs.STACS.2009.1856.

48 Francesco Pontiggia, Michele Chiari, and Matteo Pradella. Verification of programs with exceptions through operator precedence automata. In Radu Calinescu and Corina S. Pasareanu, editors, *Software Engineering and Formal Methods - 19th International Conference, SEFM 2021, Virtual Event, December 6-10, 2021, Proceedings*, volume 13085 of *Lecture Notes in Computer Science*, pages 293–311. Springer, 2021. doi:10.1007/978-3-030-92124-8_17.

49 Grigore Rosu, Feng Chen, and Thomas Ball. Synthesizing monitors for safety properties: This time with calls and returns. In Martin Leucker, editor, *Runtime Verification, 8th International Workshop, RV 2008, Budapest, Hungary, March 30, 2008. Selected Papers*, volume 5289 of *Lecture Notes in Computer Science*, pages 51–68. Springer, 2008. doi:10.1007/978-3-540-89247-2_4.

50 Nguyen Van Tang and Hitoshi Ohsaki. On model checking for visibly pushdown automata. In Adrian-Horia Dediu and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications - 6th International Conference, LATA 2012, A Coruña, Spain, March 5-9, 2012. Proceedings*, volume 7183 of *Lecture Notes in Computer Science*, pages 408–419. Springer, 2012. doi:10.1007/978-3-642-28332-1_35.

51 Niklaus Wirth and Helmut Weber. EULER: a generalization of ALGOL and it formal definition: Part 1. *Commun. ACM*, 9(1):13–25, 1966. doi:10.1145/365153.365162.

52 Martin De Wulf, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Antichains: A new algorithm for checking universality of finite automata. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle,*

WA, USA, August 17-20, 2006, Proceedings, volume 4144 of *Lecture Notes in Computer Science*, pages 17–30. Springer, 2006. doi:10.1007/11817963_5.

53 Martin De Wulf, Laurent Doyen, Nicolas Maquet, and Jean-François Raskin. Antichains: Alternative algorithms for LTL satisfiability and model-checking. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 63–77. Springer, 2008. doi:10.1007/978-3-540-78800-3_6.

A Omitted Proofs

Proposition 20

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

Proof. The reflexivity, symmetry, and transitivity of $\approx$ is trivial, and the chain-monotonicity follows from the definitions. We prove here that $\approx$ has finitely many equivalence classes.

Consider $w \in \widehat{\Sigma}^+$ for which $\lambda(w)$ is of the form $a_1 \dots a_\ell b_1 \dots b_m c_1 \dots c_n$ for some $\ell, m, n \in \mathbb{N}$ and such that

$$a_1 \geqslant a_2 \geqslant \dots \geqslant a_\ell \succ b_1 \doteq b_2 \doteq \dots \doteq b_m \prec c_1 \leqslant c_2 \leqslant \dots \leqslant c_n$$

where $a_i, b_j, c_k \in \Sigma$ for all i, j, k. Let $P_w^{\lhd} = \{a_1, b_1\} \cup \{a_{i+1} \mid a_i \succ a_{i+1}\}$ and $P_w^{\rhd} = \{b_m, c_n\} \cup \{c_i \mid c_i \prec c_{i+1}\}$. By convention, we define $P_\varepsilon^{\lhd} = P_\varepsilon^{\rhd} = \{\varepsilon\}$. We can define the profile of a word as $P_w = (w^{\lhd}, w^{\rhd}, P_w^{\lhd}, P_w^{\rhd})$. Note that there are at most $|\widehat{\Sigma}|^2 \times 2^{2|\widehat{\Sigma}|}$ distinct profiles. We show that $P_x = P_y$ implies $x \approx y$ for all $x, y \in \widehat{\Sigma}^*$.

Let $x, y \in \widehat{\Sigma}^*$ be such that $P_x = P_y$. By the equality of profiles, we directly get $x^{\lhd} = y^{\lhd}$ and $x^{\rhd} = y^{\rhd}$. Now, let $u_0, v_0 \in \widehat{\Sigma}^*$ be such that $u_0 x^{\lhd} \in \widehat{\Sigma}^*_{\geqslant}$ and $x^{\rhd} v_0 \in \widehat{\Sigma}^*_{\leqslant}$. Since x and y agree on the first and last letters, we u_0 and v_0 also satisfy $u_0 y^{\lhd} \in \widehat{\Sigma}^*_{\geqslant}$ and $y^{\rhd} v_0 \in \widehat{\Sigma}^*_{\leqslant}$. Now, let $u, v \in \widehat{\Sigma}^*$ be arbitrary. We want to show that ${}^{u}[u_0 x v_0]^{v}$ iff ${}^{u}[u_0 y v_0]^{v}$. Note that $\lambda(x)$ is of the form given above, and since $u_0 x^{\lhd} \in \widehat{\Sigma}^*_{\geqslant}$ and $x^{\rhd} v_0 \in \widehat{\Sigma}^*_{\leqslant}$, the word $u_0 x v_0$ is also of the same form. Moreover, since y has the same profile as x, the same holds for $u_0 y v_0$ as well. In particular, $P_{u_0 x v_0} = P_{u_0 y v_0}$. Then, if ${}^{u}[u_0 x v_0]^{v}$, we have $u^{\rhd} \prec a$ for all $a \in P^{\lhd}_{u_0 x v_0}$ and $b \succ v^{\lhd}$ for all $b \in P^{\rhd}_{u_0 x v_0}$. Since the same holds for $u_0 y v_0$, we get ${}^{u}[u_0 y v_0]^{v}$. The other direction is similar. ◀

Theorem 23

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

Proof. The reflexivity, symmetry, and transitivity of $\equiv_L$ is trivial. We prove that $\equiv_L$ is chain-monotonic. Let us define the following relation over $\widehat{\Sigma}^*$:

$$x \sim y \iff \forall u, v, u_0, v_0 \in \widehat{\Sigma}^* \begin{cases} \left(u_0 x^{\lhd} \in \widehat{\Sigma}^*_{\geqslant} \wedge x^{\rhd} v_0 \in \widehat{\Sigma}^*_{\leqslant} \wedge {}^{u}[u_0 x v_0]^{v}\right) \\ \Downarrow \\ \left(u u_0 x v_0 v \in L \Leftrightarrow u u_0 y v_0 v \in L\right) \end{cases}$$

Recall that for every $x, y \in \widehat{\Sigma}^*$ we have $x \equiv_L y$ if and only if both $x \approx y$ and $x \sim y$ holds, where $\approx$ is the chain equivalence.

Let $x, y \in \widehat{\Sigma}^*$ such that $x \equiv_L y$, and let $u, v, u_0, v_0 \in \widehat{\Sigma}^*$ such that $u_0 x^{\lhd}, u_0 y^{\lhd} \in \widehat{\Sigma}^*_{\geqslant}$, $x^{\rhd} v_0, y^{\rhd} v_0 \in \widehat{\Sigma}^*_{\leqslant}$, ${}^{u}[u_0 x v_0]^{v}$, and ${}^{u}[u_0 y v_0]^{v}$. We claim that $u u_0 x v_0 v \equiv_L u u_0 y v_0 v$. Since the chain equivalence $\approx$ is chain-monotonic, we get $u u_0 x v_0 v \approx u u_0 y v_0 v$. Then, we only need to show that $u u_0 x v_0 v \sim u u_0 y v_0 v$. In particular, for every $u', v', u_0', v_0' \in \widehat{\Sigma}^*$ such that $u_0'(u u_0 x v_0 v)^{\lhd} \in \widehat{\Sigma}^*_{\geqslant}$, $(u u_0 x v_0 v)^{\rhd} v_0' \in \widehat{\Sigma}^*_{\leqslant}$, ${}^{u'}[u_0' u u_0 x v_0 v v_0']^{v'}$ we have $u' u_0' u u_0 x v_0 v v_0' v' \in L$ iff $u' u_0' u u_0 y v_0 v v_0' v' \in L$.

Let $u', v', u_0', v_0' \in \widehat{\Sigma}^*$ be such that $u_0'(u u_0 x v_0 v)^{\lhd} \in \widehat{\Sigma}^*_{\geqslant}$, $(u u_0 x v_0 v)^{\rhd} v_0' \in \widehat{\Sigma}^*_{\leqslant}$, and ${}^{u'}[u_0' u u_0 x v_0 v v_0']^{v'}$. Furthermore, since $u u_0 x v_0 v \approx u u_0 y v_0 v$, we have $u_0'(u u_0 y v_0 v)^{\lhd} \in \widehat{\Sigma}^*_{\geqslant}$, $(u u_0 y v_0 v)^{\rhd} v_0' \in \widehat{\Sigma}^*_{\leqslant}$, and ${}^{u'}[u_0' u u_0 y v_0 v v_0']^{v'}$.

Now, take $u_0'' = u_0$, $u'' = u' u_0' u$, $v_0'' = v_0$, and $v'' = v v_0' v'$. Observe that since $x \equiv_L y$ and thanks to the choice of $u, v, u_0, v_0 \in \widehat{\Sigma}^*$ given above, we have $u_0'' x^{\lhd} \in \widehat{\Sigma}^*_{\geqslant}$, $u_0'' y^{\lhd} \in \widehat{\Sigma}^*_{\geqslant}$,

$x^{\triangleright}v_0'' \in \widehat{\Sigma}^*_{\lessdot}$, $y^{\triangleright}v_0'' \in \widehat{\Sigma}^*_{\lessdot}$, ${}^{u''}[u_0''xv_0'']^{v''}$, and ${}^{u''}[u_0''yv_0'']^{v''}$. Moreover, $u''u_0''xv_0''v'' \in L$ iff $u''u_0''yv_0''v'' \in L$, which is the same as $u'u_0'uu_0xv_0vv_0'v' \in L$ iff $u'u_0'uu_0yv_0vv_0'v' \in L$. Then, $uu_0xv_0v \sim uu_0yv_0v$, and thus $uu_0xv_0v \equiv_L uu_0yv_0v$. Therefore, $\equiv_L$ is chain-monotonic. ◀

Lemma 26

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

Proof. Suppose that L is an OPL over $\widehat{\Sigma}$, and let $\mathcal{A} = (Q, \{q_I\}, F, \Delta)$ be a complete deterministic OPA with the unique initial state q_I such that $L(\mathcal{A}) = L$. For every $w \in \widehat{\Sigma}^*$ the functions f_w and g_w have a finite input domain and a finite output range. The functions Φ_w however, have a finite input domain but an infinite output range. Nevertheless, only the top of the output stack of Φ_w is used in $\equiv_{\mathcal{A}}$ and, for all $w \in \widehat{\Sigma}^*$, the functions $(q,\gamma) \mapsto (\Phi(q,\gamma))^{\top}$ do have a finite output range. Then, it is easy to see that $\equiv_{\mathcal{A}}$ has finitely many equivalence classes, thanks to Proposition 20. In fact, it has at most:

$$\left(|\widehat{\Sigma}|^2 \times 2^{2|\widehat{\Sigma}|}\right) \times \left(2^{|Q|}\right)^{|Q|\times(|\Gamma|+1)} \times \left(2^{|Q|}\right)^{|Q|^2\times(|\Gamma|+1)} \times \left(2^{|\Gamma|+1}\right)^{|Q|\times(|\Gamma|+1)}$$

equivalence classes. We recall that $\Gamma = \widehat{\Sigma} \times Q$. Hence, in Landau's notation we obtain $|\equiv_{\mathcal{A}}| \leq \mathcal{O}(|\widehat{\Sigma}|)^{\mathcal{O}(|\Sigma|\times|Q|)^{\mathcal{O}(1)}}$. ◀

Lemma 27

Statement. For every OPA $\mathcal{A}$, the congruence $\equiv_{L(\mathcal{A})}$ is coarser than the congruence $\equiv_{\mathcal{A}}$.

Proof. We claim that every class of $\equiv_{\mathcal{A}}$ is contained in a class of $\equiv_L$, thus establishing that $\equiv_L$ also has finitely many equivalence classes. Consider $x, y, u, v, u_0, v_0 \in \widehat{\Sigma}^*$ such that $x \equiv_{\mathcal{A}} y$. Assume that $u_0x^{\triangleleft} \in \widehat{\Sigma}^*_{\gtrdot}$, $x^{\triangleright}v_0 \in \widehat{\Sigma}^*_{\lessdot}$, and ${}^{u}[u_0xv_0]^{v}$ hold. Then, since $x \approx y$, we have that $u_0y^{\triangleleft} \in \widehat{\Sigma}^*_{\gtrdot}$, $y^{\triangleright}v_0 \in \widehat{\Sigma}^*_{\lessdot}$ and ${}^{u}[u_0yv_0]^{v}$ hold for all $u, v, u_0, v_0 \in \widehat{\Sigma}^*$ as well. Next, we prove that all configurations (t, v, θ) where $t \in Q$ and $\theta \in \Phi_u(q_I, \bot)$ reachable from $(q_I, uu_0xv_0v, \bot)$ is also reachable from $(q_I, uu_0yv_0v, \bot)$. As the role of x and y is symmetrical, it implies that $uu_0xv_0v \in L \Leftrightarrow uu_0yv_0v \in L$. There are four cases, depending on the precedences between $u_0^{\triangleright} \gtrsim x^{\triangleleft}$ and $x^{\triangleright} \lesssim v_0^{\triangleleft}$.

We have $u^{\triangleright} \lessdot u_0^{\triangleleft}$ by definition, and we consider only the case where $u_0^{\triangleright} \gtrdot x^{\triangleleft}$ and $x^{\triangleright} \doteq v_0^{\triangleleft}$, as the other can be tackled similarly. From $(q_I, uu_0xv_0v, \bot)$ all configurations that $\mathcal{A}$ can reach after reading u are of the from $(q_u, u_0xv_0v, \theta_u \cdot \bot)$ where $q_u \in f_u(q_I, \bot)$ and $\theta_u \in \Phi_u(q_I, \bot)$, because $\varepsilon \lessdot u^{\triangleleft}$. After reading u_0, $\mathcal{A}$ can reach configurations of the form $(q_{u_0}, xv_0v, \theta_{u_0} \cdot \theta_u \cdot \bot)$ where $q_{u_0} \in f_{u_0}(q_u, \bot)$ and $\theta_{u_0} \in \Phi_{u_0}(q_u, \bot)$, due to $u^{\triangleright}u_0 \in \widehat{\Sigma}^*_{\lessdot}$. Observe that we have been able to abstract the stack $\theta_u \cdot \bot$ with $\bot$ thanks to $u^{\triangleright} \lessdot u_0^{\triangleleft}$. Then $\mathcal{A}$ must performs pop-transitions since $u_0x^{\triangleleft} \in \widehat{\Sigma}^*_{\gtrdot}$. After popping θ_{u_0}, the reachable configurations are of the form $(p_{u_0}, xv_0v, \theta_u \cdot \bot)$ where $p_{u_0} \in g_{u_0}(q_u, q_{u_0}, \bot)$. Observe that, the stack is clear from the computation of u_0 due to $u_0^{\triangleright} \gtrdot x^{\triangleleft}$, in fact ${}^{u}[u_0]^{x}$. All configurations that $\mathcal{A}$ can reach after reading x are of the form $(q_x, v_0v, \theta_x \cdot \theta_u \cdot \bot)$ where $q_x \in f_x(p_{u_0}, \bot)$ and $\theta_x \in \Phi_x(p_{u_0}, \bot)$, because $u^{\triangleright}x \in \widehat{\Sigma}^*_{\lessdot}$. Once again, we have been able to abstract the stack $\theta_u \cdot \bot$ with $\bot$, this time it is thanks to $u^{\triangleright} \lessdot x^{\triangleleft}$. Now, as we are dealing with $x^{\triangleright} \doteq v_0^{\triangleleft}$, $\mathcal{A}$ must clear the stack from the computation of x after reading v_0, and so the function g_x must be called after reading g_{v_0}. We emphasize that the stack $\theta_x \cdot \theta_u \cdot \bot$ cannot be abstracted by $\bot$. However, since $x^{\triangleright}v_0 \in \widehat{\Sigma}^*_{\lessdot}$, it can be abstracted by its top symbol, denoted by $\theta_x^{\top}$. Observe that, in the case where $x^{\triangleright} \doteq v_0^{\triangleleft}$, we must have $\theta_x \neq \bot$, and thus the top of θ_x indeed correspond to the top of $\theta_x \cdot \theta_u \cdot \bot$. Let θ_x' be defined by $\theta_x = \theta_x^{\top}\theta_x'$.

After reading v_0, $\mathcal{A}$ can reach configurations of the from $(q_{v_0}, v, \theta_{v_0} \cdot \theta'_x \cdot \theta_u \cdot \bot)$. where $q_{v_0} \in f_{v_0}(q_x, \theta_x^\top)$ and $\theta_{v_0} \in \Phi_{v_0}(q_x, \theta_x^\top)$, because $x^\triangleright v_0 \in \widehat{\Sigma}^*_{\leqslant}$. It is worth noticing that the top of the stack $\theta_x^\top$ may have been modified while reading v_0, due to $x^\triangleright \doteq v_0^\triangleleft$. For instance, it is the case when v_0 is a single letter. Then $\mathcal{A}$ must performs pop-transitions since $v_0 v^\triangleleft \in \widehat{\Sigma}^*_{\gtrdot}$. After popping θ_{v_0}, the reachable configurations are of the form $(p_{v_0}, v, \theta'_x \cdot \theta_u \cdot \bot)$ where $p_{v_0} \in g_{v_0}(q_x, q_{v_0}, \theta_x^\top)$. Observe that, the stack is clear from the computation of v_0 due to $v_0^\triangleright \gtrdot v^\triangleleft$. Moreover, g_{v_0} pops the modified top of θ_x. More pop-transitions must be performed since $xv^\triangleleft \in \widehat{\Sigma}^*_{\gtrdot}$. After popping θ'_x, the reachable configurations are of the form $(p_x, v, \theta_u \cdot \bot)$ where $p_x = g_x(p_{u_0}, p_{v_0}, \bot)$. This time, the stack is clear from the computation of x due to $x^\triangleright \gtrdot v^\triangleleft$. We emphasize that g_x is used as it have been defined for. Indeed, g_x takes as parameters the current state p_{v_0} and the parameters on which f_x have been previously called, i.e., p_{u_0} and $\bot$. By taking $t = p_x$ and $\theta = \theta_u \cdot \bot$, we established that $(q_I, uu_0xv_0v, \bot) \rightsquigarrow^* (t, v, \theta)$. As a direct consequence of $x \equiv_{\mathcal{A}} y$, making the above reasoning with y instead of x results dealing with identical intermediate sets of states and sets of stacks. Hence, we proved that all configurations of the from (t, v, θ) where $t \in Q$ and $\theta \in \Phi_u(q_I, \bot)$ reachable from $(q_I, uu_0xv_0v, \bot)$ is also reachable from $(q_I, uu_0yv_0v, \bot)$. ◀

Lemma 29

Statement. For every $L \subseteq \widehat{\Sigma}^*$, if $\equiv_L$ has finite index then L is a $\widehat{\Sigma}$-OPL.

Proof. Consider a language $L \subseteq \widehat{\Sigma}^*$ 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 $w \in \widehat{\Sigma}^*$, we denote $[w]$ its equivalence class with respect to $\equiv_L$. We construct $\mathcal{A} = (Q, \{q_0\}, F, \Delta)$ with the set of states $Q = \{([u], [v]) \mid u, v \in \widehat{\Sigma}^*\}$, the initial state $q_0 = ([\varepsilon], [\varepsilon])$, the set of accepting states $F = \{([\varepsilon], [w]) \mid w \in L\}$, and the $\widehat{\Sigma}$-driven transition function $\Delta \colon Q \times \Sigma \times (\Gamma^+ \cup \{\bot\}) \to Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma^+ \cup \{\bot\})$, where $\Gamma = \Sigma \times Q$, is defined as follows: Δ maps $(([u], [v]), a, \langle b, ([u'], [v'])\rangle\theta)$ to $(([a], [\varepsilon]), \varepsilon, \langle a, ([u], [v])\rangle\langle b, ([u'], [v'])\rangle\theta)$ if $b \lessdot a$, it returns $(([uva], [\varepsilon]), \varepsilon, \langle a, ([u'], [v'])\rangle\theta)$ if $b \doteq a$, and $(([u'], [v'uv]), a, \theta)$ if $b \gtrdot a$.

Invariants. We show that the automaton $\mathcal{A}$ satisfies the following invariants after reaching the state $([u], [v]) \in Q$ from the initial state q_0:

1. The top of the stack is $u^\triangleright$ if $u \neq \varepsilon$, and $\bot$ otherwise.
2. All outgoing on $a \in \Sigma$ satisfies $va \in \widehat{\Sigma}^*_{\gtrdot}$ and $v \neq \varepsilon \Rightarrow v^\triangleright \gtrdot a$.
3. $u \in \widehat{\Sigma}^*_{\doteq}$.
4. $u^\triangleright v \in \widehat{\Sigma}^*_{\leqslant}$ and $v \neq \varepsilon \Rightarrow u^\triangleright \lessdot v^\triangleleft$.

We prove all invariants together by induction on the length of the run from q_0 that reaches the state $([u], [v]) \in Q$. The run starts in $q_0 = ([\varepsilon], [\varepsilon])$ which trivially satisfies all invariants. By induction we assume the invariants to hold for all run of length $n > 0$. Let $([u], [v])$ be the state reached after n transitions, $a \in \Sigma$ be the incoming letter and $(b, ([u'], [v']))$ be the top of the stack.

If $b \lessdot a$, the automaton $\mathcal{A}$ performs a push-transition to reach $([a], [\varepsilon])$, which satisfies (1) by definition of push-transition. The invariants (2, 3, 4) hold trivially.

If $b \doteq a$, the automaton $\mathcal{A}$ performs a shift-transition to reach $([uva], [\varepsilon])$, which satisfies (1) by definition of the shift-transition. Since a shift-transition is triggered, the stack is not $\bot$. By the induction hypothesis, (1) ensures that $u^\triangleright = b$ and (2) ensures that $va \in \widehat{\Sigma}^*_{\gtrdot}$. Consequently and because $b \doteq a$, we have that $uva \in \widehat{\Sigma}^*_{\doteq}$, i.e., (3) is preserved. The invariants (2, 4) hold trivially.

If $b \gtrdot a$, the automaton $\mathcal{A}$ performs a pop-transition to reach $([u'], [v'uv])$. Since a pop-transition is triggered, the stack is not $\bot$. By the induction hypothesis, (1) ensures

that $u^{\triangleright} = b$, thus $u^{\triangleright} \succcurlyeq a$. In particular $u \neq \varepsilon$, and thus $u^{\triangleleft} \in \Sigma$. Additionally, the induction hypothesis gives us that $va \in \widehat{\Sigma}^*_{\geq}$ and $v \neq \varepsilon \Rightarrow v^{\triangleright} \succcurlyeq a$ by (2). We have that $uva \in \widehat{\Sigma}^*_{\geq}$ because $u^{\triangleright} \succcurlyeq a$, $u \in \widehat{\Sigma}^*_{=}$ and ${}^{u}[v]^{a}$ from (2, 3, 4). Finally, (4) ensures that $u^{\triangleright}v \in \widehat{\Sigma}^*_{\leqslant}$.

Since $([u'], [v'])$ is on the stack, there exists a strict prefix of the current run that ends in $([u'], [v'])$ and such that popping $(b, ([u'], [v']))$ recovers its stack. By the induction hypothesis on such smaller run, (1) ensures that if $u' = \varepsilon$ then the stack is $\bot$ otherwise the top is $u'^{\triangleright}$, and (3) ensures that $u' \in \widehat{\Sigma}^*_{=}$. So, (1) and (3) are directly preserved. Continuing the induction hypothesis, (2) ensures that $v'u^{\triangleleft} \in \widehat{\Sigma}^*_{\geqslant}$ and (4) ensures that $u'^{\triangleright}v' \in \widehat{\Sigma}^*_{\leqslant}$ and $v' \neq \varepsilon \Rightarrow u'^{\triangleright} \lessdot v'^{\triangleleft}$. In the case where $u' \neq \varepsilon$ and $v' = \varepsilon$, we get that $v'uv \neq \varepsilon \Rightarrow u'^{\triangleright} \lessdot (v'uv)^{\triangleleft}$ since $u'^{\triangleright} \lessdot u^{\triangleleft}$ as the automaton $\mathcal{A}$ pushed $([u'], [v'])$ on the stack while reading $u^{\triangleleft}$. The other cases are immediate. Moreover, $u'^{\triangleright}v'uv \in \widehat{\Sigma}^*_{\leqslant}$ comes as we established that $u'^{\triangleright}v' \in \widehat{\Sigma}^*_{\leqslant}$, $u \in \widehat{\Sigma}^*_{=}$, $u^{\triangleright}v \in \widehat{\Sigma}^*_{\leqslant}$, and $u'^{\triangleright} \lessdot u^{\triangleleft}$. Hence (4) is preserved. For the invariant (2), we established that $uva \in \widehat{\Sigma}^*_{\geq}$ and $v'u^{\triangleleft} \in \widehat{\Sigma}^*_{\geqslant}$, which implies that $v'uva \in \widehat{\Sigma}^*_{\geqslant}$. We also established that $v \neq \varepsilon \Rightarrow v^{\triangleright} \succcurlyeq a$ and $u^{\triangleright} \succcurlyeq a$, which implies that $(v'uv)^{\triangleright} \succcurlyeq a$. In particular, the invariant (2) is preserved.

Determinism. For all states $([a], [\varepsilon]), ([b], [\varepsilon]) \in Q$ reachable in $\mathcal{A}$ with a push-transition, if $a \neq b$ then $a \not\approx b$ which implies that $a \not\equiv_L b$. Reciprocally, if $a \equiv_L b$ then $a \approx b$, which implies that $a = b$.

For all states $([u_1], [v_1]), ([u_2], [v_2]) \in Q$ and all $a \in \Sigma$, let $([u_1v_1a], [\varepsilon]), ([u_2v_2a], [\varepsilon])$ be two states reachable in $\mathcal{A}$ with a shift-transition. We show that, if $u_1 \equiv_L u_2$ and $v_1 \equiv_L v_2$ then $u_1v_2a \equiv_L u_2v_2a$. If $v_1 = \varepsilon$, then $v_2 \equiv_L v_1$ implies $v_2 = \varepsilon$. Also $u_1 \equiv_L u_2$ implies $u_1a \equiv_L u_2a$ by chain-monotonicity of $\equiv_L$. Otherwise, if $v_1 \neq \varepsilon$, then $v_2 \equiv_L v_1$ implies $v_2 \neq \varepsilon$. Furthermore, ${}^{u_1}[v_1]^{a}$ and ${}^{u_1}[v_2]^{a}$, since $u_1 \approx u_2$ and the invariants (2) and (4) hold. In particular, $u_1^{\triangleright}v_2a \in \widehat{\Sigma}^*_{=}$. By chain-monotonicity of $\equiv_L$, we have $v_1 \equiv_L v_2$ implies $u_1v_1a \equiv_L u_1v_2a$, and $u_1 \equiv_L u_2$ implies $u_1v_2a \equiv_L u_2v_2a$ Hence, $u_1v_1a \equiv_L u_2v_2a$, by transitivity of $\equiv_L$.

For all states $([u_1], [v_1])$, $([u_2], [v_2])$, $([u_1'], [v_1'])$, $([u_2'], [v_2']) \in Q$, we let $([u_1'], [v_1'u_1v_1])$ and $([u_2'], [v_2'u_2v_2])$ be two states reachable in $\mathcal{A}$ with a pop-transition. We show that, if $u_1 \equiv_L u_2$, $v_1 \equiv_L v_2$, and $v_1' \equiv_L v_2'$, then $v_1'u_1v_1 \equiv_L v_2'u_2v_2$. As a direct consequence of the invariants, we have that ${}^{v_1'u_1}[u_1]^{\varepsilon}$, ${}^{v_1'u_1}[u_2]^{\varepsilon}$, ${}^{\varepsilon}[v_1'u_1v_2]^{\varepsilon}$, ${}^{\varepsilon}[v_1'u_2v_2]^{\varepsilon}$, ${}^{\varepsilon}[v_1']^{u_2v_2}$, and ${}^{\varepsilon}[v_2']^{u_2v_2}$. Additionally, $v_1'u^{\triangleleft}, v_1'u_2^{\triangleleft} \in \widehat{\Sigma}^*_{\geqslant}$ and $u_1^{\triangleright}v_2, u_2^{\triangleright}v_2 \in \widehat{\Sigma}^*_{\leqslant}$. By chain-monotonicity of $\equiv_L$, we have $v_1 \equiv_L v_2$ implies $v_1'u_1v_1 \equiv_L v_1'u_1v_2$, $\equiv_L$, $u_1 \equiv_L u_2$ implies $v_1'u_1v_2 \equiv_L v_1'u_2v_2$, and $v_1' \equiv_L v_2'$ implies $v_1'u_2v_2 \equiv_L v_2'u_2v_2$. Hence, $v_1'u_1v_1 \equiv_L v_2'u_2v_2$, by transitivity of $\equiv_L$.

Correctness. We have $L(\mathcal{A}) = L$ since $[w] \cap L = \varnothing$ or $[w] \subseteq L$, for all $w \in \Sigma^*$. More precisely, every $w \in \widehat{\Sigma}^*$ admits a unique run $(([\varepsilon], [\varepsilon]), w, \bot) \rightsquigarrow^* (([\varepsilon], [w']), \varepsilon, \bot)$ since $\mathcal{A}$ is deterministic. By induction we prove that, for all $x, y \in \Sigma^*$, all $\theta \in \Gamma^+ \cup \{\bot\}$, if $(q_0, xy, \bot) \rightsquigarrow^* (([u_0], [v_0], y, \theta)$ then there exist $n \in \mathbb{N}$, and $(a_i, ([u_i], [v_i]))_{i \in \{1 \dots n\}}$ such that $\theta = \langle a_1, ([u_1], [v_1])\rangle \dots \langle a_n, ([u_n], [v_n])\rangle \bot$ and $x \equiv_L u_n v_n \dots u_0 v_0$. In particular, we get that $w \equiv_L w'$ implying that $w \in L(\mathcal{A})$ iff $([\varepsilon], [w']) \in F$ iff $w' \in L$ iff $w \in L$. The base case, when the run has length $n = 0$, is trivial since $x = \varepsilon$ and $\theta = \bot$. Suppose that the property holds for all runs of length n, we prove that it holds for runs of length $n + 1$. Let $z = u_n v_n \dots u_0 v_0$. In the case where the last transition is a push-transition that reads $a \in \widehat{\Sigma}$ from $([u_0], [v_0])$. If $z = \varepsilon$ then $x = \varepsilon$ since $x \equiv_L z$. Hence $xa \equiv_L za$ holds trivially. Otherwise $z \neq \varepsilon$. Since a push-transition is triggered and $u_0^{\triangleright}v_0 \in \widehat{\Sigma}^*_{\leqslant}$ by invariant (2), we have that ${}^{\varepsilon}[z]^{a}$. Since $x \equiv_L z$ and ${}^{\varepsilon}[x]^{a}$ holds as well then we get $xa \equiv_L za$ by chain-monotonicity. In the case where the last transition is a pop-transition from $([u_0], [v_0])$. Then the reached state is $([u_1], [v_1u_0v_0])$ and the property is trivially preserved. In the case where the last transition

is a shift-transition that reads $a \in \widehat{\Sigma}$ from $([u_0], [v_0])$. Since a shift-transition is triggered, we that the $u_0^{\triangleright} \doteq a$ by invariant (1). In particular $u_0 \neq \varepsilon$. By invariant (2), if $v_0 \neq \varepsilon$ then $v_0^{\triangleright} \succcurlyeq a$. Due to $x \equiv_L z$, we have that $x^{\triangleright} = v_0^{\triangleright}$. So, $xa \equiv_L za$ since ${}^{\varepsilon}[x]^a$ and ${}^{\varepsilon}[z]^a$. Otherwise if $v_0 = \varepsilon$. Due to $x \equiv_L z$ and $u_0 \neq \varepsilon$, we have that $x^{\triangleright} = u_0^{\triangleright}$. So, $xa \equiv_L za$ since ${}^{\varepsilon}[x]^a$ and ${}^{\varepsilon}[z]^a$. ◀

Proposition 35

Statement. For every OPA $\mathcal{A}$, $\leqslant_{\mathcal{A}}$ is a computable chain-monotonic well quasi-order.

Proof. Let $\widehat{\Sigma}$ be an operator precedence alphabet, and $\mathcal{A} = (Q, I, F, \Delta)$ be an OPA. We only show that the structural quasi-order $\leqslant_{\mathcal{A}}$ is chain-monotonic since the rest is argued before the statement in the main body of the paper. Let us define the following relation over $\widehat{\Sigma}^*$:

$$x \ll y \iff \forall q \in Q, \forall \gamma \in \Gamma \cup \{\bot\} \bigwedge \begin{cases} f_x(q,\gamma) \subseteq f_y(q,\gamma) \\ g_x(q,q',\gamma) \subseteq g_y(q,q',\gamma) \\ (\Phi_x(q,\gamma))^{\top} \subseteq (\Phi_y(q,\gamma))^{\top} \end{cases}$$

Recall that for every $x, y \in \widehat{\Sigma}^*$ we have $x \leqslant_{\mathcal{A}} y$ iff $x \approx y$ and $x \ll y$, where $\approx$ is the chain equivalence. In particular, we want to show that for every $x, y, u, v, u_0, v_0 \in \widehat{\Sigma}^*$ such that $u_0 x^{\triangleleft}, u_0 y^{\triangleleft} \in \widehat{\Sigma}^*_{\geqslant}$, $x^{\triangleright} v_0, y^{\triangleright} v_0 \in \widehat{\Sigma}^*_{\leqslant}$, ${}^{u}[u_0 x v_0]^{v}$, ${}^{u}[u_0 y v_0]^{v}$, and $x \leqslant_{\mathcal{A}} y$, we have $uu_0 x v_0 v \leqslant_{\mathcal{A}} uu_0 y v_0 v$. Since the chain equivalence $\approx$ is chain-monotonic, we only need to show that $\ll$ is chain-monotonic, i.e., we have $uu_0 x v_0 v \ll uu_0 y v_0 v$ for every $x, y, u, v, u_0, v_0 \in \widehat{\Sigma}^*$ as above.

Let $\star \notin \Sigma$ be a fresh letter for which we extend the precedence relation with $a \lessdot \star$ for all $a \in \Sigma$. Let $w \in \widehat{\Sigma}^*$, $q, q' \in Q$, and $\gamma \in \Gamma \cup \{\bot\}$. Recall the following:

$$f_w(q,\gamma) = \{q_w \in Q \mid \exists \gamma_w \in \Gamma^+ \cup \{\bot\}, (q, w\star, \gamma) \rightsquigarrow^* (q_w, \star, \gamma_w)\}$$

$$\Phi_w(q,\gamma) = \{\gamma_w \in \Gamma^+ \cup \{\bot\} \mid \exists q_w \in Q, (q, w\star, \gamma) \rightsquigarrow^* (q_w, \star, \gamma_w)\}$$

$$g_w(q,q',\gamma) = \{p_w \in Q \mid \exists \gamma_w \in \Phi_w(q,\gamma), (q', \varepsilon, \gamma_w) \rightsquigarrow^* (p_w, \varepsilon, \bot)\}$$

Now, let $x, y, u, v, u_0, v_0 \in \widehat{\Sigma}^*$ such that $u_0 x^{\triangleleft}, u_0 y^{\triangleleft} \in \widehat{\Sigma}^*_{\geqslant}$, $x^{\triangleright} v_0, y^{\triangleright} v_0 \in \widehat{\Sigma}^*_{\leqslant}$, ${}^{u}[u_0 x v_0]^{v}$, ${}^{u}[u_0 y v_0]^{v}$, and $x \leqslant_{\mathcal{A}} y$. Since $x \leqslant_{\mathcal{A}} y$ implies $x \approx y$, which implies $x^{\triangleleft} = y^{\triangleleft}$ and $x^{\triangleright} = y^{\triangleright}$, it is clear that from the definitions that $f_{uu_0 x v_0 v}(q,\gamma) \subseteq f_{uu_0 y v_0 v}(q,\gamma)$ and $\Phi_{uu_0 x v_0 v}(q,\gamma) \subseteq \Phi_{uu_0 y v_0 v}(q,\gamma)$ for all $q \in Q$ and $\gamma \in \Gamma \cup \{\bot\}$. Intuitively, the reasoning for this is as follows: $\mathcal{A}$ reaches a set of configurations after processing uu_0. For every configuration in this set, consider the state and the top stack symbol given as inputs to f_x and f_y, as well as Φ_x and Φ_y, for whose outputs we know the inclusion relation above. It means that after reading $uu_0 x$ and $uu_0 y$ from any state and top stack symbol we have the same relations as well. Now, consider this time the states and top stack symbols of the configurations reached after $uu_0 x$ and $uu_0 y$. Proceeding the computation from these configurations with the suffix $v_0 v$ clearly preserves the inclusion relation for the sets of states reached. For the sets of stacks, note that ${}^{u}[u_0 x v_0]^{v}$ and ${}^{u}[u_0 y v_0]^{v}$, therefore the suffix $v_0 v$ pops the stack beyond what has been pushed while reading x and y, which is the same for both words. Finally, for all $q, q' \in Q$ and all $\gamma \in \Gamma \cup \{\bot\}$, we have that $g_{uu_0 x v_0 v}(q,q',\gamma) \subseteq g_{uu_0 y v_0 v}(q,q',\gamma)$ from $\Phi_{uu_0 x v_0 v}(q,\gamma) \subseteq \Phi_{uu_0 y v_0 v}(q,\gamma)$ which implies that $\{(q', \varepsilon, \gamma_w) \mid \gamma_w \in \Phi_{uu_0 x v_0 v}(q,\gamma)\} \subseteq \{(q', \varepsilon, \gamma_w) \mid \gamma_w \in \Phi_{uu_0 y v_0 v}(q,\gamma)\}$. Therefore, $uu_0 x v_0 v \ll uu_0 y v_0 v$, and thus $uu_0 x v_0 v \leqslant_{\mathcal{A}} uu_0 y v_0 v$, implying that $\leqslant_{\mathcal{A}}$ is chain-monotonic. ◀

Lemma 36

Statement. For every OPA $\mathcal{A}$ and $w_1, w_2 \in \widehat{\Sigma}^*$, if $w_1 \leqslant_{\mathcal{A}} w_2$ and $w_1 \in L(\mathcal{A})$ then $w_2 \in L(\mathcal{A})$.

Proof. Let $\mathcal{A} = (Q, I, F, \Delta)$. If $w_1 \in L(\mathcal{A})$ then $(q_I, w_1, \bot) \rightsquigarrow^* (q_F, \varepsilon, \bot)$ for some $q_I \in I$ and $q_F \in F$. Since $w_1 \leqslant_{\mathcal{A}} w_2$, we also have that $(q_I, w_1, \bot) \rightsquigarrow^* (q_F, \varepsilon, \bot)$ implying that $w_2 \in L(\mathcal{A})$. ◀

Lemma 39

Statement. Let $\mathcal{A} = (Q, I, F, \Delta)$ be an OPA, and let $\Gamma = \Sigma \times Q$. Considering $\vec{U}^{a,b,c}_{s,t} = \varepsilon$ when $a = b$ and $s = t$, otherwise $\vec{U}^{a,b,c}_{s,t} = \varnothing$. The following holds for all $n > 0$:

$$\texttt{Cat}^n(\vec{U}^{a,b,c}_{s,t}) = \left\{ u \mid (s, uc, \alpha) \rightsquigarrow^* (t, c, \beta), |u| = n, \alpha \in \Theta_a, \beta \in \Theta_b, au \in \widehat{\Sigma}^*_{\leqslant}, uc \in \widehat{\Sigma}^*_{\geqslant}, u^{\triangleright} = b \right\}$$

where, for all $a \in \widehat{\Sigma}$, the set of stack symbols $\Theta_a \subseteq \Gamma \cup \{\bot\}$ is defined by $\Theta_a = \{\bot\}$ if $a = \varepsilon$, and $\Theta_a = \{\langle a, q\rangle \mid q \in Q\}$ otherwise.

Proof. For readability, we define the set of runs $\Omega^{a,b,c}_{s,t,n}(w)$, for all $a, b, c \in \widehat{\Sigma}$, $s, t \in Q$, $n \in \mathbb{N}$ and $w \in \widehat{\Sigma}^*$ as follows.

$$\Omega^{a,b,c}_{s,t,m}(w) = \left\{ \rho \;\middle|\; \bigwedge \begin{array}{l} \rho = (s, wc, \alpha) \rightsquigarrow^m (t, c, \beta) \wedge m \leq 2|u| \\ \alpha \in \Theta_a \wedge \beta \in \Theta_b \\ aw \in \widehat{\Sigma}^*_{\leqslant} \wedge wc \in \widehat{\Sigma}^*_{\geqslant} \wedge (aw)^{\triangleright} = b \end{array} \right\}$$

We reformulate the statement as $u \in \texttt{Cat}^n(\vec{U}^{a,b,c}_{s,t})$ if and only if $\Omega^{a,b,c}_{s,t,2n}(u) \neq \varnothing$. For all $u \in \texttt{Cat}^n(\vec{U}^{a,b,c}_{s,t})$, it takes a simple induction on $|u|$ to prove that $\Omega^{a,b,c}_{s,t,2n}(u) \neq \varnothing$, because $\texttt{Cat}$ follows runs of $\mathcal{A}$ and preserves the invariants $m \leq 2|u|$, $au \in \widehat{\Sigma}^*_{\leqslant}$, $uc \in \widehat{\Sigma}^*_{\geqslant}$, $(au)^{\triangleright} = b$ by definition. Next, we prove that $\Omega^{a,b,c}_{s,t,2n}(u) \neq \varnothing$ implies $u \in \texttt{Cat}^n(\vec{U}^{a,b,c}_{s,t})$, where $n = |u|$.

The proof goes by induction on the structure of $w = auc$. In the base case, w does not admit any subchains, i.e., $\lambda(w) = w$. If $\Omega^{a,b,c}_{s,t,2n}(u) \neq \varnothing$ holds, then, $au \in \widehat{\Sigma}^*_{\leqslant}$ and $uc \in \widehat{\Sigma}^*_{\geqslant}$. Together with $\lambda(w) = w$, it implies that $u \in \widehat{\Sigma}^*_{\doteq}$. Hence, any run witnessing $\Omega^{a,b,c}_{s,t,2n}(u) \neq \varnothing$ performs exclusively shift-transitions. Actually, since that run performs exclusively shift-transitions, it length must be $n = |u|$. The base case goes by induction on such a witnessing run of the length n. Having $n = 0$ implies $u = \varepsilon$ and $(s, ua, \alpha) = (t, a, \beta)$. In fact $\vec{U}$ is defined such that $\varepsilon \in \vec{U}^{a,b,c}_{s,t,a}$ exactly when $a = b$ and $s = t$. By induction, we assume that there exists a run of $\Omega^{a,b,c}_{s,t,2n}(u)$ of length $n = |u|$ of the form $(s, uc, \alpha) \xrightarrow{a'} (q, vc, \theta) \rightsquigarrow^{n-1} (t, c, \beta)$ where $u = a'v$ and $a' = \theta^{\top}$. In particular $(q, vc, \theta) \rightsquigarrow^{n-1} (t, c, \beta)$ witnesses $\Omega^{a',b,c}_{q,t,2|v|}(v) \neq \varnothing$. So, $v \in \texttt{Cat}^{|v|}(\vec{U}^{a',b,c}_{q,t})$ by induction hypothesis. Finally, $u \in \texttt{Cat}^n(\vec{U}^{a,b,c}_{s,t})$ by definition of $\texttt{CatShift}$.

In the inductive step, we assume that w is of the form $a_0u_0a_1u_1 \dots a_ku_ka_{k+1}$ such that for all $0 \leq i \leq k$, either ${}^{a_i}[u_i]^{a_{i+1}}$ or $u_i = \varepsilon$, and $\lambda(a_0a_1 \dots a_{k+1}) = a_0a_1 \dots a_{k+1}$. It is worth emphasizing that $a_0 = a$, and $b = (au)^{\triangleright}$ and $a_{k+1} = c$ by definition of w. Also $k > 0$, since otherwise $\lambda(w) = w$, which is the base case of this induction. As in the base case, if $\Omega^{a,b,c}_{s,t,2n}(u) \neq \varnothing$ then $au \in \widehat{\Sigma}^*_{\leqslant}$ and $uc \in \widehat{\Sigma}^*_{\geqslant}$, which implies $a_0a_1 \dots a_{k+1} \in \widehat{\Sigma}^*_{\doteq}$ since $\lambda(a_0a_1 \dots a_{k+1}) = a_0a_1 \dots a_{k+1}$. If there exists a run ρ witnessing $\Omega^{a_0,(au)^{\triangleright},a_{k+1}}_{s,t,2n}(u) \neq \varnothing$ with $n = |u|$ then, due to $a_0a_1 \dots a_{k+1} \in \widehat{\Sigma}^*_{\doteq}$, for all $1 \leq i \leq k$, there exists a run ρ_i over u_i such that $\rho = \rho_0 \xrightarrow{a_1} \rho_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} \rho_k$. Since ${}^{a_i}[u_i]^{a_{i+1}}$ for all $0 \leq i \leq k$, each ρ_i witnesses $\Omega^{a_i,(a_iu_i)^{\triangleright},a_{i+1}}_{s_i,t_i,2n_i}(u_i) \neq \varnothing$ with $n_i = |u_i|$ and $s_i, t_i \in Q$. However, the induction hypothesis

cannot be apply on $\Omega^{a_i,(a_iu_i)^\triangleright,a_{i+1}}_{s_i,t_i,2n_i}(u_i) \neq \varnothing$, as it may have the same chain structure as w. Hence, we proceed by cases to prove that $u_i \in \texttt{Cat}^{n_i}(\vec{U}^{a_i,(a_iu_i)^\triangleright,a_{i+1}}_{s_i,t_i})$. In the case when $u_i = \varepsilon$ the run ρ_i witnessing $\Omega^{a_i,(a_iu_i)^\triangleright,a_{i+1}}_{s_i,t_i,2n_i}(u_i) \neq \varnothing$, or equivalently $\Omega^{a_i,a_i,a_{i+1}}_{s_i,t_i,2n_i}(u_i) \neq \varnothing$ must be empty since $a_i \doteq a_{i+1}$. This implies that $s_i = t_i$. In fact, we trivially have that $\varepsilon \in \vec{U}^{a_i,a_i,a_{i+1}}_{s_i,s_i,2n_i}(u_i)$ by definition of $\vec{U}$. Equivalently, $u_i \in \texttt{Cat}^{n_i}\vec{U}^{a_i,(a_iu_i)^\triangleright,a_{i+1}}_{s_i,t_i,2n_i}(u_i) \neq \varnothing$. Otherwise $u_i \neq \varepsilon$. Let $b'_i = u_i^\triangleleft$, $b_i = u_i^\triangleright$, and u'_i be such that $u_i = b'_iu'_i$. In this cases, the run ρ_i witnessing $\Omega^{a_i,(a_iu_i)^\triangleright,a_{i+1}}_{s_i,t_i,2n_i}(u_i) \neq \varnothing$ must starts with a push-transition on b'_i since $a_i \lessdot b'_i$, and must ends with a pop-transition on b_i since $b_i \gtrdot a_{i+1}$. In other words, there exists some run ρ'_i witnessing $\Omega^{b'_i,b_i,a_{i+1}}_{s'_i,t'_i,2n'_i}(u'_i) \neq \varnothing$ where $|u'_i| = n'_i$ and $s'_i, t'_i \in Q$. Now, we can apply the induction hypothesis on $w' = b'_iu'_ia_{i+1}$ which is a strict subchain of w. Hence, $u'_i \in \texttt{Cat}^{n'_i}(\vec{U}^{b'_i,b_i,a_{i+1}}_{s'_i,t'_i})$. The rest of the proof is straightforward. For all $1 \leq i \leq k$, we get $u_i \in \texttt{Cat}^{n_i}\vec{U}^{a_i,(a_iu_i)^\triangleright,a_{i+1}}_{s_i,t_i,2n_i}(u_i) \neq \varnothing$ by definition of `CatChain` and since ${}^{a_i}[u_i]^{a_{i+1}}$. Finally, we can prove that $u \in \texttt{Cat}^n(\vec{U}^{a,b,c}_{s,t})$ by induction on k, by definition of `CatShift` and since $a_0a_1 \dots a_{k+1} \in \widehat{\Sigma}^*_{\doteq}$. ◀

Lemma 41

Statement. Let $\preccurlyeq$ be a chain-monotonic quasi-order over $\widehat{\Sigma}^*$. For every $A = (Q, I, F, \Delta)$ and $\vec{X}, \vec{Y}$ such that $\mathfrak{B}(\vec{X}^{a,b,c}_{s,t} \preccurlyeq \vec{Y}^{a,b,c}_{s,t})$ holds for all $s, t \in Q$ and all $a, b, c \in \Sigma \cup \{\varepsilon\}$, we have $\mathfrak{B}(\texttt{Cat}(\vec{X}^{a,b,c}_{s,t}) \preccurlyeq \texttt{Cat}(\vec{Y}^{a,b,c}_{s,t}))$ holds also for all $s, t \in Q$ and all $a, b, c \in \Sigma \cup \{\varepsilon\}$.

Proof. Assume that $\mathfrak{B}(\vec{X}^{a,b,c}_{s,t} \preccurlyeq \vec{Y}^{a,b,c}_{s,t})$ holds for all $s, t \in Q$ and all $a, b, c \in \Sigma \cup \{\varepsilon\}$. In particular, for all $y_0 \in \vec{Y}^{a,b,c}_{s,t}$, there exists $x_0 \in \vec{X}^{a,b,c}_{s,t}$ such that $x_0 \preccurlyeq y_0$. Consider $y \in \texttt{Cat}(\vec{Y}^{a,b,c}_{s,t})$, we show that there exists $x \in \texttt{Cat}(\vec{X}^{a,b,c}_{s,t})$ such that $x \preccurlyeq y$. By definition of `Cat`, there are three cases: Either (1) $y \in \vec{Y}^{a,b,c}_{s,t}$ or, (2) $y \in \texttt{CatShift}(\vec{Y}^{a,b,c}_{s,t})$, or (3) $y \in \texttt{CatChain}(\vec{Y}^{a,b,c}_{s,t})$. We show (2) since (3) can be prove similarly and (1) is trivial from $\mathfrak{B}(\vec{X}^{a,b,c}_{s,t} \preccurlyeq \vec{Y}^{a,b,c}_{s,t})$. Suppose that y is of the form $y_1b'y_2$ for some $y_1 \in \vec{Y}^{a,a',b'}_{s,s'}$, $b' \in \Sigma$, and $y_2 \in \vec{Y}^{b',b,c}_{t',t}$. By hypothesis, there exist $x_1 \in \vec{X}^{a,a',b'}_{s,s'}$ and $x_2 \in \vec{X}^{b',b,c}_{t',t}$ such that $x_1 \preccurlyeq y_1$ and $x_2 \preccurlyeq y_2$. If $y_1 = \varepsilon$ then $x_1 = \varepsilon$ and thus $x_1b' \preccurlyeq y_1b'$. If $y_1 \neq \varepsilon$ then $y_1^\triangleright = a' \trianglelefteq b'$ and $x_1^\triangleright \trianglelefteq b'$ since $y_1 \approx x_1$. We have that ${}^\varepsilon[x_1b']^\varepsilon$ and ${}^\varepsilon[y_1b']^\varepsilon$. So, $x_1b' \preccurlyeq y_1b'$. We have that ${}^\varepsilon[x_1b'y_2]^\varepsilon$ and ${}^\varepsilon[y_1b'y_2]^\varepsilon$. So, $x_1b'y_2 \preccurlyeq y_1b'y_2$. If $b' \lessdot y_2$ then $b' \lessdot x_2$. We have that ${}^{x_1b'}[x_2]^\varepsilon$ and ${}^{x_1b'}[y_2]^\varepsilon$. So, $x_1b'x_2 \preccurlyeq x_1b'y_2$. By transitivity, $x_1b'x_1 \preccurlyeq x_1b'y_2$. If $b' \doteq y_2$ then $b' \doteq x_2$. We have that ${}^\varepsilon[x_1b'x_2]^\varepsilon$ and ${}^\varepsilon[x_1b'y_2]^\varepsilon$. So, $x_1b'x_2 \preccurlyeq x_1b'y_2$. By transitivity, $x_1b'x_1 \preccurlyeq y_1b'y_2$. ◀

Theorem 42

Statement. The algorithm from Figure 7 terminates and decides language inclusion.

Proof. First, we show that the inclusion algorithm from Figure 7 always terminates. From the definition of `Cat` and the constant $\vec{U}$, we have that each component of $\vec{X}$ holds a finite set of words after executing finitely many instructions. The halting conditions of the repeat/until loop is effectively computable. Indeed, deciding $\mathfrak{B}(X \preccurlyeq Y)$ where X, Y are finite sets and $\preccurlyeq$ a decidable quasi-order, can be done by checking whether $X \subseteq Y$ and that for every $y \in Y$ there exists $x \in X$ such that $x \preccurlyeq y$. Additionally, the quasi-order $\preccurlyeq$ is a well-quasiorders and thus, there is no infinite sequence $\{X_i\}_{i\in\mathbb{N}}$ such that $\preccurlyeq\restriction X_1 \subsetneq \preccurlyeq\restriction X_2 \subsetneq \dots$. Since $\mathfrak{B}(X \preccurlyeq Y)$ is defined by $X \subseteq Y \wedge \preccurlyeq\restriction X = \preccurlyeq\restriction Y$ and since `Cat` only extends the upward closures of the components of $\vec{X}$, we find that the repeat/until loop must terminate after finitely many iterations.

Now, we show that the inclusion algorithm from Figure 7 is correct. Supposing that the algorithm returns `ko`. Once $\vec{X}$ reached the fixpoint computed by the repeat/until loop, all $w \in \vec{X}^{\varepsilon,\varepsilon,\varepsilon}_{q_I,q_F}$ where $q_I \in I$ and $q_F \in F$ belong to L_1 by Lemma 39. Hence, when the algorithm returns `ko`, then $L_1 \not\subseteq L_2$. Conversely, supposing that $L_1 \not\subseteq L_2$, in particular let $w \in L_2 \setminus L_1$. In fact, w belongs to $\mathtt{Cat}^n(\vec{U}^{\varepsilon,\varepsilon,\varepsilon}_{q_I,q_F})$ for some $q_I \in I$, $q_F \in F$ and $n \in \mathbb{N}$, by Lemma 39. Additionally, observe that once $\vec{X}$ reached the fixpoint computed by the repeat/until loop, $\vec{X}^{\varepsilon,\varepsilon,\varepsilon}_{q_I,q_F}$ is a base of $\mathtt{Cat}^n(\vec{U}^{\varepsilon,\varepsilon,\varepsilon}_{q_I,q_F})$, i.e. $\mathfrak{B}(\vec{X}^{\varepsilon,\varepsilon,\varepsilon}_{q_I,q_F} \preccurlyeq \mathtt{Cat}^n(\vec{U}^{\varepsilon,\varepsilon,\varepsilon}_{q_I,q_F}))$. This can be proved by induction thanks to Lemma 41. Hence, there exists $w_0 \in \vec{X}^{\varepsilon,\varepsilon,\varepsilon}_{q_I,q_F}$ such that $w_0 \preccurlyeq w$. Since $\preccurlyeq$ satisfies that $(w_0 \preccurlyeq w \wedge w_0 \in L_2) \implies w \in L_2$, we get $w_0 \notin L_2$ and thus the algorithm returns `ko`. ◀