arXiv:1303.0966v1 [cs.FL] 5 Mar 2013

# Efficient Separability of Regular Languages by Subsequences and Suffixes

Wojciech Czerwiński Wim Martens Tomáš Masopust
Institute for Computer Science, University of Bayreuth

March 6, 2013

**Abstract**

When can two regular word languages $K$ and $L$ be separated by a simple language? We investigate this question and consider separation by piecewise- and suffix-testable languages and variants thereof. We give characterizations of when two languages can be separated and present an overview of when these problems can be decided in polynomial time if $K$ and $L$ are given by nondeterministic automata.

## 1 Introduction

In this paper we are motivated by scenarios in which we want to describe something complex by means of a simple language. The technical core of our scenarios consists of *separation* problems, which are usually of the following form:

> Given are two languages $K$ and $L$. Does there exist a language $S$, coming from a family $\mathcal{F}$ of *simple* languages, such that $S$ contains everything from $K$ and nothing from $L$?

The family $\mathcal{F}$ of simple languages could be, for example, languages definable in FO, piecewise testable languages, or languages definable with small automata.

Our work is specifically motivated by two seemingly orthogonal problems coming from practice: (a) increasing the user-friendliness of XML Schema and (b) efficient approximate query answering. We explain these next.

Our first motivation comes from simplifying XML Schema. XML Schema is currently the only industrially accepted and widely supported schema language for XML. Historically, it is designed to alleviate the limited expressiveness of Document Type Definition (DTD) [7], thereby making DTDs obsolete. Unfortunately, XML Schema's extra expressiveness comes at the cost of simplicity. Its code is designed to be machine-readable rather than human-readable and its logical core, based on *complex types*, does not seem well-understood by users [18]. One reason may be that the specification of XML Schema's core [9] consists of over 100 pages of intricate text. The BonXai schema language [18, 19] is an attempt to overcome these issues and to combine the simplicity of DTDs with the expressiveness of XML Schema. It has exactly the same expressive power as XML Schema, is designed to be human-readable, and avoids the use of complex types. Therefore, it aims at simplifying the development or analysis of XSDs. In its core, a BonXai schema is a set of rules

$L_1 \to R_1, \ldots, L_n \to R_n$ in which all $L_i$ and $R_i$ are regular expressions. An unranked tree $t$ (basically, an XML document) is in the language of the schema if, for every node $u$, the word formed by the labels of $u$'s children is in the language $R_k$, where $k$ is the largest number such that the word of ancestors of $u$ is in $L_k$. This semantical definition is designed to ensure full back-and-forth compatibility with XML Schema [18].

When translating an XML Schema Definition (XSD) into an equivalent BonXai schema, the regular expressions $L_i$ are obtained from a finite automaton that is embedded in the XSD. Since the current state-of-the-art in translating automata to expressions does not yet generate sufficiently clean results for our purposes, we are investigating simpler classes of expressions which we expect to suffice in practice. Practical and theoretical studies show evidence that regular expressions of the form $\Sigma^* w$ (with $w \in \Sigma^+$) and $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n$ (with $a_1, \ldots, a_n \in \Sigma$) and variations thereof seem to be quite well-suited [10, 14, 20]. We study these kinds of expressions in this paper.

Our second motivation comes from efficient approximate query answering. Efficiently evaluating regular expressions is relevant in a very wide array of fields. We choose one: in graph databases and in the context of the SPARQL language [6, 11, 16, 22] for querying RDF data. Typically, regular expressions are used in this context to match paths between nodes in a huge graph. In fact, the data can be so huge that exact evaluation of a regular expression $r$ over the graph (which can lead to a product construction between an automaton for the expression and the graph [16, 22]) may not be feasible within reasonable time. Therefore, as a compromise to exact evaluation, one could imagine that we try to rewrite the regular expression $r$ as an expression that we can evaluate much more efficiently and is close enough to $r$. Concretely, we could specify two expressions $r_{\text{pos}}$ (resp., $r_{\text{neg}}$) that define the language we want to (resp., do not want to) match in our answer and ask whether there exists a simple query (e.g., defining a piecewise testable language) that satisfies these constraints. Notice that the scenario of approximating an expression $r$ in this way is very general and not even limited to databases. (Also, we can take $r_{\text{neg}}$ to be the complement of $r_{\text{pos}}$.)

At first sight, these two motivating scenarios may seem to be fundamentally different. In the first, we want to compute an *exact* simple description of a complex object and in the second one we want to compute an *approximate* simple query that can be evaluated more efficiently. However, both scenarios boil down to the same underlying question of language separation. Our contributions are:
(1) We formally define separation problems that closely correspond to the motivating scenarios. Query approximation will be abstracted as *separation* and schema simplification as *layer-separation* (Section 2.1).
(2) We give a general characterization of separability of languages $K$ and $L$ in terms of boolean combinations of simple languages, layer-separability, and the existence of an infinite sequence of words that goes back and forth between $K$ and $L$. This characterization shows how the exact and approximate scenario are related and does not require $K$ and $L$ to be regular (Sec. 3). Our characterization generalizes a result by Stern [26] that says that a regular language $L$ is piecewise testable iff every increasing infinite sequence of words (w.r.t. subsequence ordering) alternates finitely many times between $L$ and its complement.

(3) In Section 4 we prove a decomposition characterization for separability of regular languages by piecewise testable languages and we give an algorithm that decides separability. The decomposition characterization is in the spirit of an algebraic result by Almeida [2]. It is possible to prove our characterization using Almeida's result but we provide a self-contained, elementary proof which can be understood without a background in algebra. We then use this characterization to distill a polynomial time decision procedure for separability of languages of NFAs (or regular expressions) by piecewise testable languages. The state-of-the-art algorithm for separability by piecewise testable languages ([3, 5]) runs in time $O(\text{poly}(|Q|) \cdot 2^{|\Sigma|})$ when given DFAs for the regular languages, where $|Q|$ is the number of states in the DFAs and $|\Sigma|$ is the alphabet size. Our algorithm runs in time $O(\text{poly}(|Q| + |\Sigma|))$ even for NFAs. We explain the connection to [3, 5] more closely in the Appendix. Notice that $|\Sigma|$ can be large (several hundreds and more) in the scenarios that motivate us, so we believe the improvement with respect to the alphabet to be relevant in practice.
(4) Whereas Section 4 focuses exclusively on separation by piecewise testable languages, we broaden our scope in Section 5. Let's say that a *subsequence language* is a language of the form $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$ (with all $a_i \in \Sigma$). Similarly, a *suffix language* is of the form $\Sigma^* a_1 \cdots a_n$. We present an overview of the complexities of deciding whether regular languages can be separated by subsequence languages, suffix languages, finite unions thereof, or boolean combinations thereof. We prove all cases to be in polynomial time, except separability by a single subsequence language which is NP-complete. By combining this with the results from Section 3 we also have that layer-separability is in polynomial time for all languages we consider.

We now discuss further related work. There is a large body of related work that has not been mentioned yet. Piecewise testable languages are defined and studied by Simon [23, 24], who showed that a regular language is piecewise testable iff its syntactic monoid is J-trivial and iff both the minimal DFA for the language and the minimal DFA for the reversal are partially ordered. Stern [27] suggested an $O(n^5)$ algorithm in the size of a DFA to decide whether a regular language is piecewise testable. This was improved to quadratic time by Trahtman [28]. (Actually, from our proof, it now follows that this question can be decided in polynomial time if an NFA and its complement NFA are given.)

Almeida [3] established a connection between a number of separation problems and properties of families of monoids called pseudovarieties. Almeida shows, e.g., that deciding whether two given regular languages can be separated by a language with its syntactic monoid lying in pseudovariety $\mathsf{V}$ is algorithmically equivalent to computing two-pointlike sets for a monoid in pseudovariety $\mathsf{V}$. It is then shown by Almeida et al. [4] how to compute these two-pointlike sets in the pseudovariety $\mathsf{J}$ corresponding to piecewise testable languages. Henckell et al. [12] and Steinberg [25] show that the two-pointlike sets can be computed for pseudovarieties corresponding to languages definable in first order logic and languages of dot depth at most one, respectively. By Almeida's result [3] this implies that the separation problem is also decidable for these classes.

## 2 Preliminaries and Definitions

For a finite set $S$, we denote its cardinality by $|S|$. By $\Sigma$ we always denote an alphabet, that is, a finite set of symbols. A ($\Sigma$-)*word* $w$ is a finite sequence of symbols $a_1 \cdots a_n$, where $n \geq 0$ and $a_i \in \Sigma$ for all $i = 1, \ldots, n$. The *length of* $w$, denoted by $|w|$, is $n$ and the *alphabet of* $w$, denoted by $\mathtt{Alph}(w)$, is the set $\{a_1, \ldots, a_n\}$ of symbols occurring in $w$. The empty word is denoted by $\varepsilon$. The set of all $\Sigma$-words is denoted by $\Sigma^*$. A *language* is a set of words. For $v = a_1 \cdots a_n$ and $w \in \Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$, we say that $v$ is a *subsequence* of $w$, denoted by $v \preceq w$.

A *(nondeterministic) finite automaton* or *NFA* $\mathcal{A}$ is a tuple $(Q, \Sigma, \delta, q_0, F)$, where $Q$ is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of accepting states. We sometimes denote that $q_2 \in \delta(q_1, a)$ as $q_1 \xrightarrow{a} q_2 \in \delta$ to emphasize that $\mathcal{A}$ being in state $q_1$ can go to state $q_2$ reading an $a \in \Sigma$. A *run of* $\mathcal{A}$ *on word* $w = a_1 \cdots a_n$ is a sequence of states $q_0 \cdots q_n$ where, for each $i = 1, \ldots, n$, we have $q_{i-1} \xrightarrow{a_i} q_i \in \delta$. The run is *accepting* if $q_n \in F$. Word $w$ is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on $w$. The *language of* $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of all words accepted by $\mathcal{A}$. By $\delta^*$ we denote the extension of $\delta$ to words, that is, $\delta^*(q, w)$ is the set of states that can be reached from $q$ by reading $w$. The *size* $|\mathcal{A}| = |Q| + \sum_{q,a} |\delta(q, a)|$ of $\mathcal{A}$ is the total number of transitions and states. An NFA is *deterministic* (a *DFA*) when every $\delta(q, a)$ consists of at most one element.

The *regular expressions (RE)* over $\Sigma$ are defined as follows: $\varepsilon$ and every $\Sigma$-symbol is a regular expression; whenever $r$ and $s$ are regular expressions, then so are $(r \cdot s)$, $(r + s)$, and $(s)^*$. In addition, we allow $\emptyset$ as a regular expression, but we assume that $\emptyset$ does not occur in any other regular expression. For readability, we usually omit concatenation operators and parentheses in examples. We sometimes abbreviate an $n$-fold concatenation of $r$ by $r^n$. The *language* defined by an RE $r$ is denoted by $L(r)$ and is defined as usual. Often we simply write $r$ instead of $L(r)$. Whenever we say that expressions or automata are *equivalent*, we mean that they define the same language. The *size* $|r|$ of $r$ is the total number of occurrences of alphabet symbols, epsilons, and operators in $r$, i.e., the number of nodes in its parse tree. A regular expression is *union-free* if it does not contain the operator $+$. A language is *union-free* if it is defined by a union-free regular expression.

A quasi-order is a reflexive and transitive relation. For a quasi-order $\preccurlyeq$, the *(upward) $\preccurlyeq$-closure* of a language $L$ is the set $\mathrm{closure}^{\preccurlyeq}(L) = \{w \mid v \preccurlyeq w \text{ for some } v \in L\}$. We denote the $\preccurlyeq$-closure of a word $w$ as $\mathrm{closure}^{\preccurlyeq}(w)$ instead of $\mathrm{closure}^{\preccurlyeq}(\{w\})$. Language $L$ is *(upward) $\preccurlyeq$-closed* if $L = \mathrm{closure}^{\preccurlyeq}(L)$.

A quasi-order $\preccurlyeq$ on a set $X$ is a *well-quasi-ordering* (a *WQO*) if for every infinite sequence $(x_i)_{i=1}^{\infty}$ of elements of $X$ there exist indices $i < j$ such that $x_i \preccurlyeq x_j$. It is known that every WQO is also *well-founded*, that is, there exist no infinite descending sequences $x_1 \succcurlyeq x_2 \succcurlyeq \cdots$ such that $x_i \not\preccurlyeq x_{i+1}$ for all $i$.

Higman's Lemma [13] (which we use multiple times) states that, for every alphabet $\Sigma$, the subsequence relation $\preceq$ is a WQO on $\Sigma^*$. Notice that, as a corollary to Higman's Lemma, every $\preceq$-closed language is a finite union of languages of the form $\Sigma^* a_1 \Sigma^* \ldots \Sigma^* a_n \Sigma^*$ which means that it is also regular, see also [8]. A language is *piecewise testable* if it is a finite boolean combination of $\preceq$-closed languages (or, finite boolean combination of languages $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$). In this paper, all boolean combinations are finite.

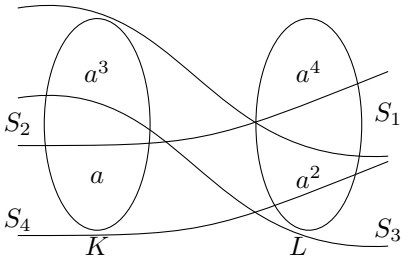


Figure 1: An example of a layer-separation.

## 2.1 Separability of Languages

A language $S$ *separates language* $K$ *from* $L$ if $S$ contains $K$ and does not intersect $L$. We say that $S$ *separates* $K$ *and* $L$ if it either separates $K$ from $L$ or $L$ from $K$. Let $\mathcal{F}$ be a family of languages. Languages $K$ and $L$ are *separable by* $\mathcal{F}$ if there exists a language $S$ in $\mathcal{F}$ that separates $K$ and $L$. Languages $K$ and $L$ are *layer-separable by* $\mathcal{F}$ if there exists a finite sequence of languages $S_1, \ldots, S_m$ in $\mathcal{F}$ such that

1. for all $1 \leq i \leq m$, language $S_i \setminus \bigcup_{j=1}^{i-1} S_j$ intersects at most one of $K$ and $L$;
2. $K$ or $L$ (possibly both) is included in $\bigcup_{j=1}^{m} S_j$.

Notice that separability always implies layer-separability. However, the opposite implication does not hold, as we demonstrate next.

**Example 1.** *Let* $\mathcal{F} = \{a^n a^* \mid n \geq 0\}$ *be a family of* $\preceq$*-closed languages over* $\Sigma = \{a\}$, $K = \{a, a^3\}$, *and* $L = \{a^2, a^4\}$. *We first show that languages* $K$ *and* $L$ *are not separable by* $\mathcal{F}$. *Indeed, assume that* $S \in \mathcal{F}$ *separates* $K$ *and* $L$. *If* $K$ *is included in* $S$, *then* $aa^* \subseteq S$, *hence* $L$ *and* $S$ *are not disjoint. Conversely, if* $L \subseteq S$, *then* $a^2a^* \subseteq S$ *and therefore* $S$ *and* $K$ *are not disjoint. This contradicts that* $S$ *separates* $K$ *and* $L$. *Now we show that the languages are layer-separable by* $\mathcal{F}$. *Consider languages* $S_1 = a^4a^*$, $S_2 = a^3a^*$, $S_3 = a^2a^*$, *and* $S_4 = aa^*$. *Then both* $K$ *and* $L$ *are included in* $S_4$, *and* $S_1$ *intersects only* $L$, $S_2 \setminus S_1 = a^3$ *intersects only* $K$, $S_3 \setminus (S_1 \cup S_2) = a^2$ *intersects only* $L$, *and* $S_4 \setminus (S_1 \cup S_2 \cup S_3) = a$ *intersects only* $K$; *see Fig. 1.*

Example 1 illustrates some intuition behind layered separability. Our motivation for layered separability comes from the BonXai schema language which is discussed in the introduction. We need to solve layer-separability if we want to decide whether an XML Schema has an equivalent BonXai schema with simple regular expressions (defining languages in $\mathcal{F}$). Layered separability implies that languages are, in a sense, separable by languages from $\mathcal{F}$ in a priority-based system: If we consider the ordered sequence of languages $S_1, S_2, S_3, S_4$ then, in order to classify a word $w \in K \cup L$ in either $K$ or $L$, we have to match it against the $S_i$ in increasing order of the index $i$. If we know the lowest index $j$ for which $w \in S_j$, we know whether $w \in K$ or $w \in L$.

We now define a tool (similar to and slightly more general than the *alternating towers* of Stern [26]) that allows us to determine when languages are *not* separable.

For languages $K$ and $L$ and a quasi-order $\preccurlyeq$, we say that a sequence $(w_i)_{i=1}^k$ of words is a *$\preccurlyeq$-zigzag between $K$ and $L$* if $w_1 \in K \cup L$ and, for all $i = 1, \ldots, k-1$:

(1) $w_i \preccurlyeq w_{i+1}$; (2) $w_i \in K$ implies $w_{i+1} \in L$; and (3) $w_i \in L$ implies $w_{i+1} \in K$.

We say that $k$ is the *length* of the $\preccurlyeq$-zigzag. We similarly define an infinite sequence of words to be an *infinite $\preccurlyeq$-zigzag between $K$ and $L$*. If the languages $K$ and $L$ are clear from the context then we sometimes omit them and refer to the sequence as a *(infinite) $\preccurlyeq$-zigzag*. If we consider the subsequence order $\preceq$, then we simply write a *zigzag* instead of a $\preceq$-zigzag. Notice that we do not require $K$ and $L$ to be disjoint. If there is a $w \in K \cap L$ then there clearly exists an infinite zigzag: $w, w, w, \ldots$

**Example 2.** *In order to illustrate infinite zigzags consider the languages $K = \{a(ab)^{2k}c(ac)^{2\ell} \mid k, \ell \geq 0\}$ and $L = \{b(ab)^{2k+1}c(ac)^{2\ell+1} \mid k, \ell \geq 0\}$. Then the following infinite sequence is an infinite zigzag between $K$ and $L$:*

$$w_i = \begin{cases} b(ab)^i c(ac)^i & \text{if } i \text{ is odd} \\ a(ab)^i c(ac)^i & \text{if } i \text{ is even} \end{cases}$$

*Indeed $w_1 \in L$, words from the sequence alternately belong to $K$ and $L$, and for all $i \geq 1$ we have $w_i \preceq w_{i+1}$.* □

# 3 A Characterization of Separability

The aim of this section is to prove the following theorem. It extends a result by Stern that characterizes piecewise testable languages [26]. In particular, it applies to general languages and does not require $K$ to be the complement of $L$.

**Theorem 3.** *For languages $K$ and $L$ and a WQO $\preccurlyeq$ on words, the following are equivalent.*

*(1) $K$ and $L$ are separable by a boolean combination of $\preccurlyeq$-closed languages.*

*(2) $K$ and $L$ are layer-separable by $\preccurlyeq$-closed languages.*

*(3) There does not exist an infinite $\preccurlyeq$-zigzag between $K$ and $L$.*

Some of the equivalences in the theorem still hold when the assumptions are weakened. For example the equivalence between (1) and (2) does not require $\preccurlyeq$ to be a WQO.

Since the subsequence order $\preceq$ is a WQO on words, we know from Theorem 3 that languages are separable by piecewise testable languages if and only if they are layer-separable by $\preceq$-closed languages. Actually, since $\preceq$ is a WQO (and therefore only has finitely many minimal elements within a language), the latter is equivalent to being layer-separable by languages of the form $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$.

In Example 1 we illustrated two languages $K$ and $L$ that are layer-separable by $\preceq$-closed languages. Notice that $K$ and $L$ can also be separated by a boolean combination of the languages $a^*a^1$, $a^*a^2$, $a^*a^3$, and $a^*a^4$ from $\mathcal{F}$, as $K \subseteq ((a^*a^1 \setminus a^*a^2) \cup (a^*a^3 \setminus a^*a^4))$ and $L \cap ((a^*a^1 \setminus a^*a^2) \cup (a^*a^3 \setminus a^*a^4)) = \emptyset$.

We now give an overview of the proof of Theorem 3. The next lemma proves the equivalence between (1) and (2), but is slightly more general. In particular, it does not rely on a WQO.

**Lemma 4.** *Let $\mathcal{F}$ be a family of languages closed under intersection and containing $\Sigma^*$. Then languages $K$ and $L$ are separable by a finite boolean combination of languages from $\mathcal{F}$ if and only if $K$ and $L$ are layer-separable by $\mathcal{F}$.*

The proof (given in the Appendix) is constructive. The *only if* direction is the more complex one and shows how to exploit the implicit negation in the first condition in the definition of layer-separability in order to simulate separation by boolean combinations. Notice that the families of $\preccurlyeq$-closed languages in Theorem 3 always contain $\Sigma^*$ and are closed under intersection.

The following lemma shows that the implication (2) $\Rightarrow$ (3) in Theorem 3 does not require well-quasi ordering.

**Lemma 5.** *Let $\preccurlyeq$ be a quasi order on words and assume that languages $K$ and $L$ are layer-separable by $\preccurlyeq$-closed languages. Then there is no infinite $\preccurlyeq$-zigzag between $K$ and $L$.*

To prove that (3) implies (2), we need the following technical lemma in which we require $\preccurlyeq$ to be a WQO. In the proof of the lemma, we argue how we can see $\preccurlyeq$-zigzags in a tree structure. Intuitively, every path in the tree structure corresponds to a $\preccurlyeq$-zigzag. We need the fact that $\preccurlyeq$ is a WQO in order to show that we can assume that every node in this tree structure has a finite number of children. We then apply König's lemma to show that arbitrarily long $\preccurlyeq$-zigzags imply the existence of an infinite $\preccurlyeq$-zigzag. The lemma then follows by contraposition.

**Lemma 6.** *Let $\preccurlyeq$ be a WQO on words. If there is no infinite $\preccurlyeq$-zigzag between languages $K$ and $L$, then there exists a constant $k \in \mathbb{N}$ such that no $\preccurlyeq$-zigzag between $K$ and $L$ is longer than $k$.*

If there is no infinite $\preccurlyeq$-zigzag, then we can put a bound on the maximal length of zigzags by Lemma 6. This bound actually has a close correspondence to the number of "layers" we need to separate $K$ and $L$.

**Lemma 7.** *Let $\preccurlyeq$ be a WQO on words and assume that there is no infinite $\preccurlyeq$-zigzag between languages $K$ and $L$. Then the languages $K$ and $L$ are layer-separable by $\preccurlyeq$-closed languages.*

# 4 Testing Separability by Piecewise Testable Languages

Whereas Section 3 proves a result for general WQOs, we focus in this section exclusively on the ordering $\preceq$ of subsequences. Therefore, if we say *zigzag* in this section, we always mean *$\preceq$-zigzag*. We show here how to decide the existence of an infinite zigzag between two regular word languages, given by their regular expressions or NFAs, in polynomial time. According to Theorem 3, this is equivalent to deciding if the two languages can be separated by a piecewise testable language.

To this end, we first prove a decomposition result that is reminiscent of a result of Almeida ([2], Theorem 4.1 in [4]). We show that, if there is an infinite zigzag between regular languages, then there is an infinite zigzag of a special form and in which every word can be decomposed in some synchronized manner. We can find these

special forms of zigzags in polynomial time in the NFAs for the languages. The main features are that our algorithm runs exponentially faster in the alphabet size than the current state-of-the-art [5] and that our algorithm and its proof of correctness do not require knowledge of the algebraic perspective on regular languages.

A regular language is a *cycle language* if it is of the form $u(v)^*w$, where $u, v, w$ are words and $(\texttt{Alph}(u) \cup \texttt{Alph}(w)) \subseteq \texttt{Alph}(v)$. We say that $v$ is the *cycle* of the language and that $\texttt{Alph}(v)$ is its *cycle alphabet*. Regular languages $L^{\mathcal{A}}$ and $L^{\mathcal{B}}$ are *synchronized in one step* if they are of one of the following forms:

- $L^{\mathcal{A}} = L^{\mathcal{B}} = \{w\}$, that is, they are the same singleton word, or
- $L^{\mathcal{A}}$ and $L^{\mathcal{B}}$ are cycle languages with equal cycle alphabets.

We say that regular languages $L^{\mathcal{A}}$ and $L^{\mathcal{B}}$ are *synchronized* if they are of the form $L^{\mathcal{A}} = D_1^{\mathcal{A}} D_2^{\mathcal{A}} \dots D_k^{\mathcal{A}}$ and $L^{\mathcal{B}} = D_1^{\mathcal{B}} D_2^{\mathcal{B}} \dots D_k^{\mathcal{B}}$ where, for all $1 \leq i \leq k$, languages $D_i^{\mathcal{A}}$ and $D_i^{\mathcal{B}}$ are synchronized in one step. So, languages are synchronized if they can be decomposed into (equally many) components that can be synchronized in one step. Notice that synchronized languages are always non-empty.

**Example 8.** *Languages $L^{\mathcal{A}} = a(ba)^*aab\,ca\,bb(bc)^*$ and $L^{\mathcal{B}} = b(aab)^*ba\,ca\,cc(cbc)^*b$ are synchronized. Indeed, $L^{\mathcal{A}} = D_1^{\mathcal{A}} D_2^{\mathcal{A}} D_3^{\mathcal{A}}$ and $L^{\mathcal{B}} = D_1^{\mathcal{B}} D_2^{\mathcal{B}} D_3^{\mathcal{B}}$ for $D_1^{\mathcal{A}} = a(ba)^*aab$, $D_2^{\mathcal{A}} = ca$, $D_3^{\mathcal{A}} = bb(cb)^*$ and $D_1^{\mathcal{B}} = b(aab)^*ba$, $D_2^{\mathcal{B}} = ca$, and $D_3^{\mathcal{B}} = cc(cbc)^*b$.*

The next lemma shows that, in order to search for infinite zigzags, it suffices to search for synchronized sublanguages. The proof goes through a sequence of lemmas that gradually shows how the sublanguages of $L^{\mathcal{A}}$ and $L^{\mathcal{B}}$ can be made more and more specific.

**Lemma 9** (Synchronization / Decomposition)**.** *There is an infinite zigzag between regular languages $L^{\mathcal{A}}$ and $L^{\mathcal{B}}$ if and only if there exist synchronized languages $K^{\mathcal{A}} \subseteq L^{\mathcal{A}}$ and $K^{\mathcal{B}} \subseteq L^{\mathcal{B}}$.*

We now use this result to obtain a polynomial-time algorithm solving our problem. The first step is to define what it means for NFAs to contain synchronized sublanguages.

For an NFA $\mathcal{A}$ over an alphabet $\Sigma$, two states $p$, $q$, and a word $w \in \Sigma^*$, we write $p \xrightarrow{w} q$ if $q \in \delta^*(p, w)$ or, in other words, the automaton can go from state $p$ to state $q$ by reading $w$. For $\Sigma_0 \subseteq \Sigma$, states $p$ and $q$ are $\Sigma_0$-*connected* in $\mathcal{A}$ if there exists a word $uvw \in \Sigma_0^*$ such that:

1. $\texttt{Alph}(v) = \Sigma_0$ and
2. there is a state $m$ such that $p \xrightarrow{u} m$, $m \xrightarrow{v} m$, and $m \xrightarrow{w} q$.

Consider two NFAs $\mathcal{A} = (Q^{\mathcal{A}}, \Sigma, \delta^{\mathcal{A}}, q_0^{\mathcal{A}}, F^{\mathcal{A}})$ and $\mathcal{B} = (Q^{\mathcal{B}}, \Sigma, \delta^{\mathcal{B}}, q_0^{\mathcal{B}}, F^{\mathcal{B}})$. Let $(q^{\mathcal{A}}, q^{\mathcal{B}})$ and $(\bar{q}^{\mathcal{A}}, \bar{q}^{\mathcal{B}})$ be in $Q^{\mathcal{A}} \times Q^{\mathcal{B}}$. We say that $(q^{\mathcal{A}}, q^{\mathcal{B}})$ and $(\bar{q}^{\mathcal{A}}, \bar{q}^{\mathcal{B}})$ are *synchronizable in one step* if one of the following situations occurs:

- there exists a symbol $a$ in $\Sigma$ such that $q^{\mathcal{A}} \xrightarrow{a} \bar{q}^{\mathcal{A}}$ and $q^{\mathcal{B}} \xrightarrow{a} \bar{q}^{\mathcal{B}}$,
- there exists an alphabet $\Sigma_0 \subseteq \Sigma$ such that $q^{\mathcal{A}}$ and $\bar{q}^{\mathcal{A}}$ are $\Sigma_0$-connected in $\mathcal{A}$ and $q^{\mathcal{B}}$ and $\bar{q}^{\mathcal{B}}$ are $\Sigma_0$-connected in $\mathcal{B}$.

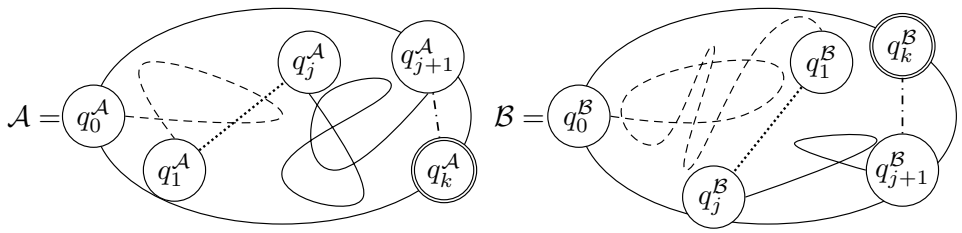


Figure 2: Synchronization of automata $\mathcal{A}$ and $\mathcal{B}$.

We say that automata $\mathcal{A}$ and $\mathcal{B}$ are *synchronizable* if there exists a sequence of pairs $(q_0^{\mathcal{A}}, q_0^{\mathcal{B}}), \ldots, (q_k^{\mathcal{A}}, q_k^{\mathcal{B}}) \in Q^{\mathcal{A}} \times Q^{\mathcal{B}}$ such that:

1. for all $0 \leq i < k$, $(q_i^{\mathcal{A}}, q_i^{\mathcal{B}})$ and $(q_{i+1}^{\mathcal{A}}, q_{i+1}^{\mathcal{B}})$ are synchronizable in one step;
2. states $q_0^{\mathcal{A}}$ and $q_0^{\mathcal{B}}$ are initial states of $\mathcal{A}$ and $\mathcal{B}$, respectively; and
3. states $q_k^{\mathcal{A}}$ and $q_k^{\mathcal{B}}$ are accepting states of $\mathcal{A}$ and $\mathcal{B}$, respectively.

Notice that if the automata $\mathcal{A}$ and $\mathcal{B}$ are synchronizable, then the languages $L(\mathcal{A})$ and $L(\mathcal{B})$ are not necessarily synchronized, only some of its sublanguages are necessarily synchronized.

**Lemma 10** (Synchronizability of automata)**.** *For two NFAs $\mathcal{A}$ and $\mathcal{B}$, the following conditions are equivalent.*

1. *Automata $\mathcal{A}$ and $\mathcal{B}$ are synchronizable.*
2. *There exist synchronized languages $K^{\mathcal{A}} \subseteq L(\mathcal{A})$ and $K^{\mathcal{B}} \subseteq L(\mathcal{B})$.*

The intuition behind Lemma 10 is depicted in Figure 2. The idea is that there is a sequence $(q_0^{\mathcal{A}}, q_0^{\mathcal{B}}), \ldots, (q_k^{\mathcal{A}}, q_k^{\mathcal{B}})$ that witnesses that $\mathcal{A}$ and $\mathcal{B}$ are synchronizable. The pairs of paths that have the same style of lines depict parts of the automaton that are synchronizable in one step. In particular, the dotted path from $q_1^{\mathcal{A}}$ to $q_j^{\mathcal{A}}$ has the same word as the one from $q_1^{\mathcal{B}}$ to $q_j^{\mathcal{B}}$. The other two paths contain at least one loop.

The following theorem states that synchronizability in automata captures exactly the existence of infinite zigzags between their languages. The theorem statement uses Theorem 3 for the connection between infinite zigzags and separability.

**Theorem 11.** *Let $\mathcal{A}$ and $\mathcal{B}$ be two NFAs. Then the languages $L(\mathcal{A})$ and $L(\mathcal{B})$ are separable by a piecewise testable language if and only if the automata $\mathcal{A}$ and $\mathcal{B}$ are not synchronizable.*

We can now show how the algorithm from [5] can be improved to test in polynomial time whether two given NFAs are synchronizable or not. Our algorithm computes quadruples of states that are synchronizable in one step and by linking such quadruples together so that they form a pair of paths as illustrated in Figure 2.

**Theorem 12.** *Given two NFAs $\mathcal{A}$ and $\mathcal{B}$, it is possible to test in polynomial time whether $L(\mathcal{A})$ and $L(\mathcal{B})$ can be separated by a piecewise testable language.*

| $\mathcal{F}(O, C)$ | single | unions | bc (boolean combinations) |
|---|---|---|---|
| $\preceq$ (subsequence) | NP-complete | PTIME | PTIME |
| $\preceq_s$ (suffix) | PTIME | PTIME | PTIME |

Table 1: The complexity of deciding separability for regular languages $K$ and $L$.

# 5 Asymmetric Separation and Suffix Order

We present a bigger picture on efficient separations that are relevant to the scenarios that motivate us. For example, we consider what happens when we restrict the allowed boolean combinations of languages. Technically, this means that separation is no longer symmetric. Orthogonally, we also consider the suffix order $\preceq_s$ between strings in which $v \preceq_s w$ if and only if $v$ is a (not necessarily strict) suffix of $w$. An important technical difference with the rest of the paper is that the suffix order is not a WQO. Indeed, the suffix order $\preceq_s$ has an infinite antichain, e.g., $a, ab, abb, abbb, \ldots$ The results we present here for suffix order hold true for prefix order as well.

Let $\mathcal{F}$ be a family of languages. Language $K$ is *separable from a language $L$ by $\mathcal{F}$* if there exists a language $S$ in $\mathcal{F}$ that separates $K$ from $L$, i.e., contains $K$ and does not intersect $L$. Thus, if $L$ is closed under complement, then $K$ is separable from $L$ implies $L$ is separable from $K$. The *separation problem by $\mathcal{F}$* asks, given an NFA for $K$ and an NFA for $L$, whether $K$ is separable from $L$ by $\mathcal{F}$.

We consider separation by families of languages $\mathcal{F}(O, C)$, where $O$ ("order") specifies the ordering relation and $C$ ("combinations") specifies how we are allowed to combine (upward) $O$-closed languages. Concretely, $O$ is either the subsequence order $\preceq$ or the suffix order $\preceq_s$. We allow $C$ to be one of *single*, *unions*, or *bc* (boolean combinations), meaning that each language in $\mathcal{F}(O, C)$ is either the $O$-closure of a single word, a finite union of the $O$-closures of single words, or a finite boolean combination of the $O$-closures of single words. Thus, $\mathcal{F}(\preceq, bc)$ is the family of piecewise testable languages and $\mathcal{F}(\preceq_s, bc)$ is the family of suffix-testable languages. With this convention in mind, the main result of this section is to provide a complete complexity overview of the six possible cases of separation by $\mathcal{F}(O, C)$. The case $\mathcal{F}(\preceq, \text{bc})$ has been proved in Section 4 and the remaining ones are proved in the Appendix.

**Theorem 13.** *For $O \in \{\preceq, \preceq_s\}$ and $C$ being one of single, unions, or boolean combinations, we have that the complexity of the separation problem by $\mathcal{F}(O, C)$ is as indicated in Table 1.*

Since the separation problem for prefix order is basically the same as the separation for suffix order and has the same complexity we didn't list it separately in the table. Furthermore, from Lemma 4 we immediately obtain that deciding layer-separability for all six cases in Table 1 is in PTIME.

# 6 Conclusions and Further Questions

Subsequence- and suffix languages seem to be very promising for obtaining "simple" separations of regular languages, since we can often efficiently decide if two given regular languages are separable (Table 1). Layer-separability is even in PTIME in

all cases. Looking back at our motivating scenarios, the obvious next questions are: if a separation exists, can we efficiently compute one? How large is it?

If we look at the broader picture, we wonder if our characterization of separability can be used in a wider context than regular languages and subsequence ordering. Are there other cases where it can be used lead to obtain efficient decision procedures? Another concrete question is whether we can decide in polynomial time if a given NFA defines a piecewise-testable language. Furthermore, we are also interested in efficient separation results by combinations of languages of the form $\Sigma^* w_1 \Sigma^* \cdots \Sigma^* w_n$ or variants thereof.

**Acknowledgments.** We thank Jean-Eric Pin and Marc Zeitoun for patiently answering our questions about the algebraic perspective on this problem. We are grateful to Mikołaj Bojańczyk, who pointed out the connection between layered separability and boolean combinations. We also thank Piotr Hofman for pleasant and insightful discussions about our proofs during his visit to Bayreuth.

# References

[1] S. Afonin and D. Golomazov. Minimal union-free decompositions of regular languages. In *Languages and Automata Theory and Applications*, pages 83–92, 2009.

[2] J. Almeida. Implicit operations on finite J-trivial semigroups and a conjecture of I. Simon. *Journal of Pure and Applied Algebra*, 69:205–218, 1990.

[3] J. Almeida. Some algorithmic problems for pseudovarieties. *Publicationes Mathematicae Debrecen*, 54:531–552, 1999.

[4] J. Almeida, J.C. Costa, and M. Zeitoun. Pointlike sets with respect to R and J. *Journal of Pure and Applied Algebra*, 212(3):486–499, 2008.

[5] J. Almeida and M. Zeitoun. The pseudovariety J is hyperdecidable. *RAIRO Informatique Théorique et Applications*, 31(5):457–482, 1997.

[6] M. Arenas, S. Conca, and J. Pérez. Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent the adoption of the standard. In *World Wide Web Conference*, p. 629–638, 2012.

[7] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language XML 1.0 (fifth edition). Tech. report, W3C Recommendation, November 2008. http://www.w3.org/TR/2008/REC-xml-20081126/.

[8] A. Ehrenfeucht, D. Haussler, and G. Rozenberg. On regularity of context-free languages. *Theoretical Computer Science*, 27(3):311–332, 1983.

[9] S. Gao, C.M. Sperberg-McQueen, H.S. Thompson, N. Mendelsohn, D. Beech, M. Maloney. W3C XML Schema Definition Language (XSD) 1.1 part 1. Tech. report, W3C, 2009. http://www.w3.org/TR/2009/CR-xmlschema11-1-20090430/.

[10] W. Gelade and F. Neven. Succinctness of pattern-based schema languages for XML. *Journal of Computer and System Sciences*, 77(3):505–519, 2011.

[11] S. Harris and A. Seaborne. SPARQL 1.1 query language. Tech. report, W3C, 2010.

[12] K. Henckell, J. Rhodes, and B. Steinberg. Aperiodic pointlikes and beyond. *International Journal of Algebra and Computation*, 20(2):287–305, 2010.

[13] G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, s3–2(1):326–336, 1952.

[14] G. Kasneci and T. Schwentick. The complexity of reasoning about pattern-based XML schemas. In *Principles of Database Systems*, pages 155–164, 2007.

[15] D. König. Über eine Schlussweise aus dem Endlichen ins Unendliche. *Acta Litterarum ac Scientiarum*, 3:121–130, 1927.

[16] K. Losemann and W. Martens. The complexity of evaluating path expressions in SPARQL. In *Principles of Database Systems*, pages 101–112, 2012.

[17] D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25(2):322–336, 1978.

[18] W. Martens, F. Neven, M. Niewerth, and T. Schwentick. Developing and analyzing XSDs through BonXai. *Proc. of the VLDB Endowment*, 5(12):1994–1997, 2012.

[19] W. Martens, F. Neven, M. Niewerth, and T. Schwentick. BonXai: Combining the simplicity of DTD with the expressiveness of XML Schema, 2013. Manuscript.

[20] W. Martens, F. Neven, T. Schwentick, and G.J. Bex. Expressiveness and complexity of XML Schema. *ACM Trans. on Database Systems*, 31(3):770–813, 2006.

[21] B. Nagy. Union-free regular languages and 1-cycle-free-path automata. *Publicationes Mathematicae Debrecen*, 68(1-2):183–197, 2006.

[22] J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. *Journal of Web Semantics*, 8(4):255–270, 2010.

[23] I. Simon. *Hierarchies of Events with Dot-Depth One*. PhD thesis, Dep. of Applied Analysis and Computer Science, University of Waterloo, Canada, 1972.

[24] I. Simon. Piecewise testable events. In *GI Conference on Automata Theory and Formal Languages*, pages 214–222. Springer, 1975.

[25] B. Steinberg. A delay theorem for pointlikes. *Semigroup Forum*, 63:281–304, 2001.

[26] J. Stern. Characterizations of some classes of regular events. *Theoretical Computer Science*, 35(1985):17–42, 1985.

[27] J. Stern. Complexity of some problems from the theory of automata. *Information and Control*, 66(3):163–176, 1985.

[28] A. N. Trahtman. Piecewise and local threshold testability of DFA. In *Fundamentals of Computation Theory*, p. 347–358, 2001.

# Appendix

## Connection to the Algorithm of Almeida and Zeitoun

Almeida and Zeitoun [5] show that the following problem is in polynomial time:

**Input:** Two DFAs $\mathcal{A} = (Q^{\mathcal{A}}, \Sigma, \delta^{\mathcal{A}}, q_0^{\mathcal{A}}, F^{\mathcal{A}})$ and $\mathcal{B} = (Q^{\mathcal{B}}, \Sigma, \delta^{\mathcal{B}}, q_0^{\mathcal{B}}, F^{\mathcal{B}})$ with constant-size alphabet $\Sigma$.

**Problem:** Are $L(\mathcal{A})$ and $L(\mathcal{B})$ separable by a piecewise testable language?

A result by Almeida [3] says that separability is equivalent to computing the intersection of topological closures of the regular languages that are to be separated. This is used by Almeida and Zeitoun [5], who prove that these topological closures can be represented by a class of automata (going beyond DFAs or NFAs) computable from the original automata. The main differences with the present procedure are that the construction of [5] is

(1) exponential in the size of the alphabet and

(2) defined on DFAs rather than on NFAs.

Actually, the exponential time bound w.r.t. the alphabet size has already been observed by Almeida and Zeitoun in the conclusions of their paper [4]. The reason why the algorithm from [5] is exponential in the size of the alphabet is that its first step consists of adding, to each loop that uses a subset $B$ of the alphabet $\Sigma$, a new loop containing $B^\omega$. (See Definition 4.1 from [5] – the notation $B^\omega$ is borrowed from that paper.) The number of these subsets can be exponential. In fact, the algorithm from [5] first adds these cycles to $\mathcal{A}$ and $\mathcal{B}$ separately and then (after some more operations) compares the automata to each other. However, the relevant cycles to add to $\mathcal{A}$ depend on $\mathcal{B}$.

**Example 14.** *Consider a language*

$$L(\mathcal{A}) = (a_1^* \cdots a_n^*)^*.$$

*Then, for every subset* $S = \{i_1, \ldots, i_j\} \subseteq \{1, \ldots, n\}$*, there exists a language*

$$L(\mathcal{B}_S) = (a_{i_1} \cdots a_{i_j})^*$$

*such that the intersection of the closures of the above languages contains* $u^\omega$ *only for words* $u$ *such that* $\mathtt{Alph}(u) = \{a_{i_1}, \ldots, a_{i_j}\}$.

The example shows that if we compute the closure of $L(\mathcal{A})$ without looking simultaneously at $\mathcal{B}$ we have to keep all of the exponentially many loops in order to be prepared for intersecting this closure with the closure of any possible language $L(\mathcal{B}_S)$. In fact, one needs to do more than naïvely compute largest common subsets of alphabets of loops that obviously correspond to each other. We show how to do this while avoiding the exponent in $|\Sigma|$.

The following is a slightly less trivial example that shows how alphabets of strongly connected components can correspond to each other.

**Example 15.** *Consider languages*

$$(a^*bc^*de^*)^*acb^*ac(ba)^*ca(bc)^*b$$

*and*

$$(ab)^*d(ab^*df^*)^*b(c(ab)^*c^*b^*a(cb)^*)^*b.$$

*These languages cannot be separated by a piecewise testable language. The example of profinite word in the intersection of the closures is*

$$(abd)^\omega cac(ab)^\omega ca(bc)^\omega.$$

*(Again, the notation $m^\omega$ is borrowed from [5] and is the standard one for the unique idempotent power of element $m$ of the semigroup.)*

# Proofs of Section 3

**Lemma 4.** *Let $\mathcal{F}$ be a family of languages closed under intersection and containing $\Sigma^*$. Then languages $K$ and $L$ are separable by a finite boolean combination of languages from $\mathcal{F}$ if and only if $K$ and $L$ are layer-separable by $\mathcal{F}$.*

*Proof.* For a sequence $S_1, S_2, \ldots, S_k$ denote, for all $i = 1, \ldots, k$

$$\mathrm{D}_i^S = S_i \setminus \bigcup_{j=1}^{i-1} S_j\,.$$

To show the *if* part, assume that $S_1, S_2, \ldots, S_m$ is the sequence of languages from $\mathcal{F}$ layer-separating $K$ and $L$. We will construct a finite boolean combination of languages from $\mathcal{F}$ that separates $K$ and $L$. By definition of layer separability, each language $\mathrm{D}_i^S$ intersects at most one of $K$ and $L$. Furthermore, $K$ or $L$ is included in $\bigcup_{j=1}^m \mathrm{D}_j^S = \bigcup_{j=1}^m S_j$. Without loss of generality, assume that $K$ is included in $\bigcup_{j=1}^m \mathrm{D}_j^S$, and set $J = \{j \mid \mathrm{D}_j^S \cap K \neq \emptyset\}$. Then the language $S = \bigcup_{j\in J} \mathrm{D}_j^S$ separates $K$ and $L$. As $S$ is a finite boolean combination of languages from $\mathcal{F}$, this part is shown.

To show the *only if* part, assume that $K$ and $L$ can be separated by a language $S$ that is a finite boolean combination of languages from $\mathcal{U} = \{U_1, \ldots, U_k\}$, a finite subset of $\mathcal{F}$. Without loss of generality, assume that $K \subseteq S$ and $L \cap S = \emptyset$. For any subset of indices $I \subseteq \{1, \ldots, k\}$ we denote

$$\mathrm{cell}_{\mathcal{U}}(I) = \Big(\bigcap_{i\in I} U_i\Big) \cap \Big(\bigcap_{i\notin I} \overline{U_i}\Big),$$

where $\overline{U_i} = \Sigma^* \setminus U_i$ and call this language a *cell*; see Fig. 3 for an illustration. Observe that the cells are pairwise disjoint and $S$ is a finite union of cells. As $S$ separates $K$ and $L$, every cell intersects at most one of the languages $K$ and $L$. The cells that form $S$ do not intersect $L$ and the others do not intersect $K$. Based on this, we construct a layer-separation of $K$ and $L$ by $\mathcal{F}$.

To this end, we show that there exists a sequence of languages $S_1, \ldots, S_{2^k}$ from $\mathcal{F}$ and a bijection $\pi : \{1, \ldots, 2^k\} \to \mathcal{P}(\{1, \ldots, k\})$ such that, for every $1 \leq j \leq 2^k$

$$\mathrm{D}_j^S = \mathrm{cell}_{\mathcal{U}}(\pi(j)).$$

We call $S_1, \ldots, S_{2^k}$ a sequence of *cell-separating* languages for $\mathcal{U}$. It is easy to see that this sequence $S_1, \ldots, S_{2^k}$ would layer-separate $K$ and $L$. Indeed, for each $1 \leq \ell \leq 2^k$, the set $\mathrm{D}_\ell^S$ is a cell. Thus it intersects at most one of $K$ and $L$, which is the first requirement of a layer-separation. Moreover, the union $\bigcup_{1 \leq i \leq 2^k} S_i = \bigcup_{1 \leq i \leq 2^k} \mathrm{D}_i^S$ includes all the cells, so it equals $\Sigma^*$, thus clearly includes both $K$ and $L$.

Therefore, it only remains to prove that there exists a sequence $S_1, \ldots, S_{2^k}$ of cell-separating languages for $\mathcal{U}$. Before we show it formally we present an illustrating example in order to give an intuition how the required sequence is constructed. For $\mathcal{U} = \{U_1, U_2, U_3\}$ the cell-separating sequence is as follows:

$$U_1 \cap U_2 \cap U_3, U_2 \cap U_3, U_1 \cap U_3, U_3, U_1 \cap U_2, U_2, U_1, \Sigma^*$$

We prove the fact in general by induction on $k$ that there is a sequence of cell-separating languages for every $k$-element set $\mathcal{U} \subseteq \mathcal{F}$. For the base step, i.e., $k = 1$, we have that $\mathcal{U} = \{S_1\}$. We can simply take $S_1 = U_1$ and $S_2 = \Sigma^*$ and we are done. Assume now that, for some $k$, the induction hypothesis is satisfied. We prove it for $k + 1$. Consider an arbitrary subset $\mathcal{U}' = \{U_1, \ldots, U_k, U_{k+1}\}$ of $\mathcal{F}$ and take $\mathcal{U} = \{U_1, \ldots, U_k\}$. Let $S_1, \ldots, S_{2^k}$ be the sequence of cell-separating languages for $\mathcal{U}$. We will show that the sequence

$$S_1 \cap U_{k+1}, \ldots, S_{2^k} \cap U_{k+1},\ S_1, \ldots, S_{2^k}$$

is cell-separating for $\mathcal{U}'$. We name this sequence $T_1, \ldots, T_{2^{k+1}}$, i.e.,

$$T_i = \begin{cases} S_i \cap U_{k+1}, & \text{if } \ i \leq 2^k \\ S_{i-2^k}, & \text{if } \ i > 2^k. \end{cases}$$

It is sufficient to show that there exists a bijection $g$ between $\{1, \ldots, 2^{k+1}\}$ and $\mathcal{P}(\{1, \ldots, k+1\})$ such that for $1 \leq i \leq 2^{k+1}$

$$\mathrm{D}_i^T = \mathrm{cell}_{\mathcal{U}'}(\sigma(i)).$$

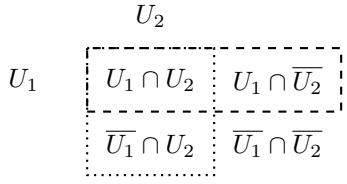


Figure 3: Cells for two languages $U_1$ and $U_2$.

Assume that $\pi$ is a bijection between $\{1,\dots,2^k\}$ and $\mathcal{P}(\{1,\dots,k\})$ such that

$$\mathrm{D}_i^S = \mathrm{cell}_{\mathcal{U}}(\pi(i)).$$

We will show that $\sigma$ defined as

$$\sigma(i) = \begin{cases} \pi(i) \cup \{k+1\}, & \text{if } \ i \le 2^k \\ \pi(i-2^k), & \text{if } \ i > 2^k \end{cases}$$

fulfills the necessary condition. If $i \le 2^k$ then

$$\begin{aligned} \mathrm{D}_i^T &= T_i \setminus \bigcup_{j=1}^{i-1} T_j = (S_i \cap U_{k+1}) \setminus \bigcup_{j=1}^{i-1} (S_j \cap U_{k+1}) = (S_i \setminus \bigcup_{j=1}^{i-1} S_j) \cap U_{k+1} \\ &= \mathrm{D}_i^S \cap U_{k+1} = \mathrm{cell}_{\mathcal{U}}(\pi(i)) \cap U_{k+1} = \mathrm{cell}_{\mathcal{U}'}(\sigma(i)). \end{aligned}$$

On the other hand if $i > 2^k$ then

$$\begin{aligned} \mathrm{D}_i^T &= T_i \setminus \bigcup_{j=1}^{i-1} T_j = (S_{i-2^k}) \setminus \Big( \bigcup_{j=1}^{2^k} (S_j \cap U_{k+1}) \cup \bigcup_{j=1}^{i-2^k-1} S_j \Big) \\ &= (S_{i-2^k}) \setminus \Big( U_{k+1} \cup \bigcup_{j=1}^{i-2^k-1} S_j \Big) = \mathrm{D}_{i-2^k}^S \setminus U_{k+1} \\ &= \mathrm{cell}_{\mathcal{U}}(\pi(i-2^k)) \setminus U_{k+1} = \mathrm{cell}_{\mathcal{U}'}(\sigma(i)), \end{aligned}$$

since $\bigcup_{j=1}^{2^k} S_j = \Sigma^*$, which completes the proof. □

**Lemma 5.** *Let $\preccurlyeq$ be a quasi-order on words and assume that languages $K$ and $L$ are layer-separable by $\preccurlyeq$-closed languages. Then there is no infinite $\preccurlyeq$-zigzag between $K$ and $L$.*

*Proof.* For the sake of contradiction, assume that there exists an infinite $\preccurlyeq$-zigzag $(w_i)_{i=1}^{\infty}$ between $K$ and $L$. Let $I = \{w_1, w_2, \dots\}$ and consider the sequence of languages $S_1, \dots, S_m$ layer-separating $K$ and $L$. Let $k$ in $\{1,\dots,m\}$ be the lowest index for which $S_k \cap I \neq \emptyset$. (Notice that $k$ exists by definition of layer-separations.) Since we chose $k$ to be minimal, for every $j \ge 1$ it holds that

$$w_j \notin \bigcup_{i=1}^{k-1} S_i \,.$$

Let $\ell \ge 1$ be such that $w_\ell \in S_k \cap I$. Without loss of generality, assume that $w_\ell \in K$. (Otherwise, we switch $K$ and $L$.) Then, by the definition of zigzag, $w_{\ell+1} \in L$. As $S_k$ is $\preccurlyeq$-closed and as $w_\ell \preccurlyeq w_{\ell+1}$, we have that also $w_{\ell+1} \in S_k$. Thus,

$$w_{\ell+1} \in L \cap (S_k \setminus \bigcup_{i=1}^{k-1} S_i) \qquad \text{and} \qquad w_\ell \in K \cap (S_k \setminus \bigcup_{i=1}^{k-1} S_i)\,.$$

But then the set $S_k \setminus \bigcup_{i=1}^{k-1} S_i$ intersects both languages $K$ and $L$, which is a contradiction with the assumption that $S_1, \dots, S_m$ layer-separates $K$ and $L$. □

In the next proof we use König's Lemma, which we recall next. A tree is *finitely branching* if every node has finitely many children. Note that, for every $n > 0$ there can be a node that has at least $n$ children.

**Lemma 16** (König [15])**.** *A finitely branching tree containing arbitrarily long paths contains an infinite path.*

**Lemma 6.** *Let $\preccurlyeq$ be a WQO on words. If there is no infinite $\preccurlyeq$-zigzag between languages $K$ and $L$, then there exists a constant $k \in \mathbb{N}$ such that no $\preccurlyeq$-zigzag between $K$ and $L$ is longer than $k$.*

*Proof.* All $\preccurlyeq$-zigzags considered in this proof are between languages $K$ and $L$. We show that the existence of arbitrarily long $\preccurlyeq$-zigzags implies the existence of an infinite $\preccurlyeq$-zigzag. To this end, we restrict the general form of $\preccurlyeq$-zigzag to be able to use König's Lemma. Note that any WQO allows equivalent elements. For a word $w$, let $[w] = \{v \in \Sigma^* \mid v \preccurlyeq w \text{ and } w \preccurlyeq v\}$ denote the equivalence class containing $w$. For languages $K$ and $L$, we arbitrarily pick two elements from the sets $[w] \cap K$ and $[w] \cap L$, if they exist, denoted by $[w]_K$ and $[w]_L$, respectively, and call them *canonical elements* of the class $[w]$. We say that a $\preccurlyeq$-zigzag $(w_i)_{i=1}^k$ is *canonical* if it consists only of canonical elements, that is, if $w_i \in K$ then $w_i = [w_i]_K$, and if $w_i \in L$ then $w_i = [w_i]_L$. Observe that if there exists a $\preccurlyeq$-zigzag of length $k$ then there also exists a canonical $\preccurlyeq$-zigzag of length $k$. Indeed, replacing all elements of the $\preccurlyeq$-zigzag with their corresponding canonical elements results in a canonical $\preccurlyeq$-zigzag. Thus, in what follows, we consider only canonical $\preccurlyeq$-zigzags. Note that we reduced the quasi order to an order. We say that a $\preccurlyeq$-zigzag $(w_i)_{i=1}^k$ is *denser* than a $\preccurlyeq$-zigzag $(v_i)_{i=1}^k$ if

- $w_i \preccurlyeq v_i$, for all $1 \leq i \leq k$;
- $w_i \in K \iff v_i \in K$, for all $1 \leq i \leq k$, and also symmetrically for $L$; and
- there exists $1 \leq j \leq k$ such that $w_j \neq v_j$.

A $\preccurlyeq$-zigzag is *densest* if there is no denser $\preccurlyeq$-zigzag.

Note that if a $\preccurlyeq$-zigzag $(w_i)_{i=1}^k$ is densest then $(w_i)_{i=1}^j$ is also densest for any $j < k$. Indeed, if $(v_i)_{i=1}^j$ is denser than $(w_i)_{i=1}^j$ then $v_1, \ldots, v_j, w_{j+1}, \ldots, w_k$ is also a valid $\preccurlyeq$-zigzag, which is denser than $(w_i)_{i=1}^k$. Furthermore, observe that if there exists a $\preccurlyeq$-zigzag of length $k$ then there also exists a densest $\preccurlyeq$-zigzag of length $k$ because the denser order is well founded, as a suborder of a $k$-componentwise product of well founded orders. Thus, by the assumtions, there exist arbitrarily long densest $\preccurlyeq$-zigzags. Their first element belongs either to $K$ or to $L$. Without loss of generality, we may assume that there are arbitrarily long densest $\preccurlyeq$-zigzags starting in $K$. Note that the first word in every densest $\preccurlyeq$-zigzag is the shortest canonical element with respect to the order $\preccurlyeq$ among the canonical elements of $K$. As the order $\preccurlyeq$ is a WQO there are only finitely many shortest canonical elements, thus there exists a word $w \in K$ such that there are arbitrarily long densest $\preccurlyeq$-zigzags starting from $w$. Consider a tree consisting of all these $\preccurlyeq$-zigzags forming its paths. By definition, this tree has arbitrary long paths. It is also finitely branching; otherwise, if a node has infinitely many children labelled by different words $v_1, v_2, \ldots$, the WQO property implies that we can find a pair of indices $i < j$

such that $v_i \preccurlyeq v_j$. Then the $\preccurlyeq$-zigzag obtained by choosing the path going through $v_j$ is not densest as we can change $v_j$ into $v_i$ in this zigzag obtaining the denser one. Thus, by Lemma 16, this tree contains an infinite path that forms an infinite $\preccurlyeq$-zigzag. □

**Lemma 7.** *Let $\preccurlyeq$ be a WQO on words and assume that there is no infinite $\preccurlyeq$-zigzag between languages $K$ and $L$. Then the languages $K$ and $L$ are layer-separable by $\preccurlyeq$-closed languages.*

*Proof.* For two languages $X$ and $Y$, let

$$\text{layer}(X, Y) = \{w \in X \mid \text{there does not exist } w' \text{ in } Y \text{ such that } w \preccurlyeq w'\}$$

denote the set of all words of $X$ that are not smaller or equal to a string of $Y$ in the WQO. We first show the following claim:

**Claim 17.** *There exists a $\preccurlyeq$-closed language $S_{(X,Y)}$ such that $S_{(X,Y)} \cap Y = \emptyset$ and $S_{(X,Y)} \cap X = \text{layer}(X, Y)$.*

The proof of the claim is simple. Let $S_{(X,Y)} = \bigcup_{w \in \text{layer}(X,Y)} \text{closure}^{\preccurlyeq}(w)$. By definition, $S_{(X,Y)}$ is $\preccurlyeq$-closed. For each $w$ in $\text{layer}(X, Y)$, we have that $\text{closure}^{\preccurlyeq}(w) \cap Y = \emptyset$ by definition of $\text{layer}(X, Y)$. Therefore, $S_{(X,Y)} \cap Y = \emptyset$. Moreover, we have that $\text{layer}(X, Y) = S_{(X,Y)} \cap X$ because $w \in \text{layer}(X, Y)$ implies that $(\text{closure}^{\preccurlyeq}(w) \cap X) \subseteq \text{layer}(X, Y)$. This concludes the proof of the claim.

We now proceed with the proof of Lemma 7. Let $B$ be a constant such that no $\preccurlyeq$-zigzag between $K$ and $L$ is longer than $B$. This constant exists by Lemma 6, since there is no infinite $\preccurlyeq$-zigzag between $K$ and $L$. Define the languages $K_0 = K$, $L_0 = L$, and, for each $i \in \mathbb{N}$,

$$K_{i+1} = K_i \setminus \text{layer}(K_i, L_i) \qquad\qquad L_{i+1} = L_i \setminus \text{layer}(L_i, K_i)\,.$$

We prove by induction on $i$ that every $\preccurlyeq$-zigzag between $K_i$ and $L_i$ has length at most $B - i$. The claim holds for $K_0$ and $L_0$, thus consider $K_{i+1}$ and $L_{i+1}$, for $i \geq 0$. Since $K_{i+1} \subseteq K_i$ and $L_{i+1} \subseteq L_i$ we have that every $\preccurlyeq$-zigzag between $K_{i+1}$ and $L_{i+1}$ would also be a $\preccurlyeq$-zigzag between $K_i$ and $L_i$. By induction we know that every $\preccurlyeq$-zigzag between $K_i$ and $L_i$ has length at most $B - i$. Therefore, every $\preccurlyeq$-zigzag between $K_{i+1}$ and $L_{i+1}$ also has length at most $B - i$. It remains to prove that there cannot be a $\preccurlyeq$-zigzag of length $B - i$ between $K_{i+1}$ and $L_{i+1}$. For the sake of contradiction, assume that $(w_k)_{k=1}^{B-i}$ is a $\preccurlyeq$-zigzag between $K_{i+1}$ and $L_{i+1}$ of length $B - i$. We either have that $w_{B-i} \in K_{i+1}$ or $w_{B-i} \in L_{i+1}$. We prove the case $w_{B-i} \in K_{i+1}$ since the other case is analogous. Here, we have that $w_{B-i} \notin \text{layer}(K_i, L_i)$. By definition of $\text{layer}(K_i, L_i)$, there exists a $w \in L_i$ such that $w_{B-i} \preccurlyeq w$. But this means that the sequence $w_1, \ldots, w_{B-i}, w$ would be a $\preccurlyeq$-zigzag between $K_i$ and $L_i$ of length $B - i + 1$, which is a contradiction.

We therefore have that, for every $i$, every $\preccurlyeq$-zigzag between $K_i$ and $L_i$ has length at most $B - i$. In particular, this means that if $i \geq B$, every $\preccurlyeq$-zigzag between languages $K_i$ and $L_i$ has length at most zero. Since any word $w \in K_i \cup L_i$ would already be a $\preccurlyeq$-zigzag of length one, this means that $K_i = L_i = \emptyset$.

We now show how the languages $K$ and $L$ can be layer-separated by $\preccurlyeq$-closed languages. Denote by $S_{(X,Y)}$ the $\preccurlyeq$-closed language obtained when applying Claim 17 to languages $X$ and $Y$. Then, the sequence

$$S_{(K_0,L_0)},\ S_{(L_0,K_0)},\ S_{(K_1,L_1)},\ S_{(L_1,K_1)},\ \ldots,\ S_{(K_{B-1},L_{B-1})},\ S_{(L_{B-1},K_{B-1})}$$

covering the layers

$$\text{layer}(K_0, L_0),\ \text{layer}(L_0, K_0),\ \ldots,\ \text{layer}(K_{B-1}, L_{B-1}),\ \text{layer}(L_{B-1}, K_{B-1}),$$

respectively, layer-separates $K$ and $L$. Condition 1 of the definition of layered separability is satisfied because all the languages covering layers with smaller numbers appear earlier in the sequence. Condition 2 is true because the union of all the considered layers includes $K \cup L$. □

# Proofs of Section 4

## Proof of Lemma 9 with Running Example

In this section we prove Lemma 9.

**Lemma 9.** *There is an infinite zigzag between regular languages $L^{\mathcal{A}}$ and $L^{\mathcal{B}}$ if and only if there exist synchronized languages $K^{\mathcal{A}} \subseteq L^{\mathcal{A}}$ and $K^{\mathcal{B}} \subseteq L^{\mathcal{B}}$.*

To prove it, we need several auxiliary results showing that if there is an infinite zigzag between two regular languages, then there is also an infinite zigzag between their sublanguages of a special form.

To illustrate the proofs of this section we use a running example with regular languages

$$L_1^{\mathcal{A}} = a(b^*a)^*a(bb)^*abcabb(bc)^* + (ab^*c)^* + b^*c(cb)^*$$

and

$$L_1^{\mathcal{B}} = abd + b(aab)^*baca(b(cb^*)^*c)^*cc(cbc)^*b + (aa)^* + ba(bb)^*$$

having an infinite zigzag between them. After each step we present how the considered languages have been modified.

We say that language $K$ *embeds* into $L$, denoted by $K \preceq L$, if for every $v \in K$, there exists a $w \in L$ such that $v \preceq w$. In order to be consistent we also say here that word $v$ *embeds* into word $w$ if $\{v\}$ embeds in $\{w\}$, i.e., $v$ is a subsequence of $w$. Languages $K^{\mathcal{A}}$ and $K^{\mathcal{B}}$ are *mutually embeddable* if $K^{\mathcal{A}} \preceq K^{\mathcal{B}}$ and $K^{\mathcal{B}} \preceq K^{\mathcal{A}}$. Note that there always exists an infinite zigzag between nonempty mutually-embeddable languages.

**Lemma 18** (Mutual embeddability)**.** *If there is an infinite zigzag between regular languages $L^{\mathcal{A}}$ and $L^{\mathcal{B}}$, then there exist nonempty mutually-embeddable regular languages $K^{\mathcal{A}} \subseteq L^{\mathcal{A}}$ and $K^{\mathcal{B}} \subseteq L^{\mathcal{B}}$.*

*Proof.* We define languages $K^{\mathcal{A}}$ and $K^{\mathcal{B}}$ and show that they possess the required properties. Let $I$ denote the set of all words that belong to any infinite zigzag between languages $L^{\mathcal{A}}$ and $L^{\mathcal{B}}$, and let $K^{\mathcal{A}} = L^{\mathcal{A}} \cap I$ and $K^{\mathcal{B}} = L^{\mathcal{B}} \cap I$. Then, for any $w \in K^{\mathcal{A}}$, let $I_w$ denote an infinite zigzag containing $w$. As $I_w$ is infinite, there exists $w' \in I_w \cap L^{\mathcal{B}}$ such that $w \preceq w'$, hence $w' \in K^{\mathcal{B}}$. Therefore $K^{\mathcal{A}} \preceq K^{\mathcal{B}}$. The case $K^{\mathcal{B}} \preceq K^{\mathcal{A}}$ is analogous.

It remains to show that $K^{\mathcal{A}}$ and $K^{\mathcal{B}}$ are regular. We prove it for $K^{\mathcal{A}}$ since the case for $K^{\mathcal{B}}$ is analogous. Let $M$ denote the set of all minimal words of $L^{\mathcal{A}} \setminus K^{\mathcal{A}}$, that is, words $w \in L^{\mathcal{A}} \setminus K^{\mathcal{A}}$ such that there is no $w' \in L^{\mathcal{A}} \setminus K^{\mathcal{A}}$ with $w' \preceq w$ and

$w' \neq w$. Note that any distinct words $w$ and $w'$ of $M$ are incomparable, i.e., $w \not\preceq w'$ and $w' \not\preceq w$. By Higman's lemma, $M$ is finite. If $w \in L^{\mathcal{A}} \setminus K^{\mathcal{A}}$, that is, $w \notin I$, then any $w' \in L^{\mathcal{A}}$ with $w \preceq w'$ also belongs to $L^{\mathcal{A}} \setminus K^{\mathcal{A}}$; otherwise, $w' \in K^{\mathcal{A}} = L^{\mathcal{A}} \cap I$ implies that $w \in I$, which is a contradiction. Thus,

$$L^{\mathcal{A}} \setminus K^{\mathcal{A}} = L^{\mathcal{A}} \cap \bigcup_{w \in M} \text{closure}(w)\,.$$

Notice that the language $\bigcup_{w \in M} \text{closure}(w)$ is $\preceq$-closed, hence regular, and so is the language $K^{\mathcal{A}} = L^{\mathcal{A}} \setminus (L^{\mathcal{A}} \setminus K^{\mathcal{A}})$. □

By Lemma 18 our running example could be reduced to

$$L_2^{\mathcal{A}} = a(b^*a)^*a(bb)^*abcabb(bc)^* + b^*c(cb)^*$$

and

$$L_2^{\mathcal{B}} = b(aab)^*baca(b(cb^*)^*c)^*cc(cbc)^*b + (aa)^* + ba(bb)^*,$$

since words from $(ab^*c)^* \subseteq L_1^{\mathcal{A}}$ and $abd \subseteq L_1^{\mathcal{B}}$ does not belong to any infinite zigzag.

Now we strengthen the result by imposing a union-free decomposition on the languages $K^{\mathcal{A}}$ and $K^{\mathcal{B}}$.

**Lemma 19** (Union-free languages)**.** *If there is an infinite zigzag between regular languages $L^{\mathcal{A}}$ and $L^{\mathcal{B}}$, then there exist nonempty mutually-embeddable union-free regular languages $K^{\mathcal{A}} \subseteq L^{\mathcal{A}}$ and $K^{\mathcal{B}} \subseteq L^{\mathcal{B}}$.*

*Proof.* By Lemma 18, there exist nonempty mutually-embeddable regular languages $M^{\mathcal{A}} \subseteq L^{\mathcal{A}}$ and $M^{\mathcal{B}} \subseteq L^{\mathcal{B}}$. By [21], see also [1], every regular language can be expressed as a finite union of union-free languages, hence we have

$$M^{\mathcal{A}} = D_1^{\mathcal{A}} \cup D_2^{\mathcal{A}} \cup \ldots \cup D_k^{\mathcal{A}} \qquad \text{and} \qquad M^{\mathcal{B}} = D_1^{\mathcal{B}} \cup D_2^{\mathcal{B}} \cup \ldots \cup D_\ell^{\mathcal{B}}\,,$$

where all languages $D_i^{\mathcal{A}}$ and $D_j^{\mathcal{B}}$ are union-free. It remains to show that there exist $1 \leq i \leq k$ and $1 \leq j \leq \ell$ such that $D_i^{\mathcal{A}}$ and $D_j^{\mathcal{B}}$ are mutually embeddable. We first show that for each $D_i^{\mathcal{A}}$ there exists a $D_j^{\mathcal{B}}$ such that $D_i^{\mathcal{A}} \preceq D_j^{\mathcal{B}}$. Consider a union-free regular expression for $D_i^{\mathcal{A}}$. For any $n \in \mathbb{N}$, we define $w_n$ as a word obtained from the expression for $D_i^{\mathcal{A}}$ by replacing stars with $n$. For the expression $(ab^*c)^*(cb)^*$ we have $w_n = (ab^nc)^n(ab)^n$. Note that for every $w \in D_i^{\mathcal{A}}$, there exists $n \in \mathbb{N}$ and a word $w_n \in D_i^{\mathcal{A}}$ such that $w \preceq w_n$. Number $n$ can be chosen s $n = |w|$ since $w_n$ is in $D_i^{\mathcal{A}}$ by definition.

Consider now the sequence $(w_n)_{n=1}^{\infty}$. Every word $w_n$ can be embedded to a word of $M^{\mathcal{B}}$, therefore to a word of $D_j^{\mathcal{B}}$, for some $j$. Thus, there exists a $j_0$ such that infinitely many words of the sequence $(w_n)_{n=1}^{\infty}$ embed to words of $D_{j_0}^{\mathcal{B}}$. We claim that $D_i^{\mathcal{A}} \preceq D_{j_0}^{\mathcal{B}}$. As mentioned above, for every $w \in D_i^{\mathcal{A}}$, there exists an $n$ such that $w \preceq w_n$. As there are infinitely many words $w_s$ embedding to $D_{j_0}^{\mathcal{B}}$, there exists $m \geq n$ such that $w_m$ embeds to $D_{j_0}^{\mathcal{B}}$. Clearly, $w_n \preceq w_m$, thus $w \preceq w_n \preceq w_m$ and, hence, $w$ embeds to $D_{j_0}^{\mathcal{B}}$, which shows that $D_i^{\mathcal{A}} \preceq D_{j_0}^{\mathcal{B}}$.

Thus, there is a function $f : \{1, \ldots, k\} \to \{1, \ldots, \ell\}$ such that $D_i^{\mathcal{A}} \preceq D_{f(i)}^{\mathcal{B}}$, and a function $g : \{1, \ldots, \ell\} \to \{1, \ldots, k\}$ such that $D_j^{\mathcal{B}} \preceq D_{g(j)}^{\mathcal{A}}$. Define the function $h : \{1, \ldots, k\} \to \{1, \ldots, k\}$ by $h(i) = g(f(i))$. Considering the sequence

$h^1(1), h^2(1), h^3(1), \ldots$ at some moment we encounter a repetition since all the values come from a finite set. In other words there exist numbers $c, d \in \mathbb{N}$, $c < d$, such that $h^c(1) = h^d(1) = i$. This means that $D_i^{\mathcal{A}} \preceq D_{f(i)}^{\mathcal{B}} \preceq D_i^{\mathcal{A}}$. We assign $K^{\mathcal{A}} = D_i^{\mathcal{A}}$ and $K^{\mathcal{B}} = D_{f(i)}^{\mathcal{B}}$. □

By Lemma 19 by we can simplify the example to

$$L_3^{\mathcal{A}} = a(b^*a)^*a(bb)^*abcabb(bc)^* \quad \text{and} \quad L_3^{\mathcal{B}} = b(aab)^*baca(b(cb^*)^*c)^*cc(cbc)^*b$$

by eliminating $b^*c(cb)^* \subseteq L_2^{\mathcal{A}}$ and $(aa)^* + ba(bb)^* \subseteq L_2^{\mathcal{B}}$.

We now consider a transformation $L \mapsto L'$ transforming a language to a simpler one, which is still mutually embeddable with the origin. By transitivity of the mutual-embeddability relation, we may transform two mutually embeddable languages and the results will also be mutually embeddable. Next four lemmas describe this transformation.

**Lemma 20** (Star depth one)**.** *For every union-free regular language $L$, there exists a union-free regular language $L'$ such that:*

- *$L'$ is of star depth at most one, i.e., it has a regular expression of the form $v_1(v_2)^*v_3 \ldots (v_{2i})^*v_{2i+1}$, where all $v_j \in \Sigma^*$,*
- *$L$ and $L'$ are mutually embeddable,*
- *$L' \subseteq L$.*

*Proof.* Consider a union-free regular expression $r$ such that $L(r) = L$. Then $r$ is of the form

$$r = v_1(r_2)^*v_3(r_4)^*v_5 \ldots (r_{2i})^*v_{2i+1},$$

where $v_{2j+1} \in \Sigma^*$ and $r_{2j}$ is a (union-free) regular expression, for $j \leq i$. Let $v_{2j}$ be obtained from $r_{2j}$ by deleting all star operations. For example, if $r_{2j} = a(bcd^*b)^*$ then $v_{2j} = abcdb$. Clearly, $L(r') \subseteq L$. Our aim is to show that $L$ and $L(r')$ with

$$r' = v_1(v_2)^*v_3(v_4)^*v_5 \ldots (v_{2i})^*v_{2i+1}$$

are mutually embeddable. Recall from the proof of Lemma 19 that for every word $w \in L$ there exists $n$ such that $w \preceq w_n$ (where $w_n$ denotes the word obtained from $r$ by replacing all star operations with $n$). It is now sufficient to show that $w_n \preceq L(r')$ for every $n \geq 1$. To this end, fix an arbitrary $n \in \mathbb{N}$ and assume that

$$w_n = v_1\bar{v}_2v_3 \ldots \bar{v}_{2i}v_{2i+1},$$

where $\bar{v}_{2j} \in L(r_{2j})$, for $j \leq i$. Note that each symbol occurring in $\bar{v}_{2j}$ also occurs in $v_{2j}$. Thus, $\bar{v}_{2j} \preceq v_{2j}^{|\bar{v}_{2j}|}$ and, therefore, $w_n \preceq L(r')$, which implies that $L(r) \preceq L(r')$. Since $L(r') \subseteq L(r)$ implies that $L(r') \preceq L(r)$, the proof is complete. □

Due to Lemma 20 we may eliminate the stars on depth more than one obtaining

$$L_4^{\mathcal{A}} = a(ba)^*a(bb)^*abcabb(bc)^* \quad \text{and} \quad L_4^{\mathcal{B}} = b(aab)^*baca(bcbc)^*cc(cbc)^*b.$$

Recall that every union-free language of star depth at most one is of the form $v_1(v_2)^*v_3 \ldots (v_{2i})^*v_{2i+1}$. We call the words $v_{2j}$ in the scope of a star operation *loops*. A loop $v_{2j}$ with $\mathtt{Alph}(v_{2j}) = \Sigma_0 \subseteq \Sigma$ is called a $\Sigma_0$-*loop*.

**Lemma 21** (Eliminating a loop)**.** *Let $L$ be a regular language of the form $L = v_1(v_2)^*v_3 \dots (v_{2i})^*v_{2i+1}$ and assume that for some $1 \leq k, \ell \leq i$, $k \neq \ell$,*

- $\mathtt{Alph}(v_{2\ell}) \subseteq \mathtt{Alph}(v_{2k})$*, and*
- $\mathtt{Alph}(v_j) \subseteq \mathtt{Alph}(v_{2k})$ *for all* $\min(2k, 2\ell) < j < \max(2k, 2\ell)$.

*Then the languages $L$ and $L' = v_1(v_2)^*v_3 \dots v_{2\ell-1}v_{2\ell+1} \dots (v_{2i})^*v_{2i+1}$ obtained from $L$ by eliminating the $v_{2\ell}$ loop are mutually embeddable.*

*Proof.* We can assume that $k < \ell$. Indeed, if the lemma holds for $k < \ell$ we can immediately infer that it also holds for $k > \ell$ because $K$ and $L$ are mutually embeddable if and only if the reversed languages $K^{\mathrm{rev}}$ and $L^{\mathrm{rev}}$ are mutually embeddable. As $L' \subseteq L$ we have that $L' \preceq L$. It remains to show that $L$ embeds to $L'$. Fix an arbitrary word $w \in L$ and assume that $w = v_1\bar{v}_2v_3 \dots \bar{v}_{2i}v_{2i+1}$ where $\bar{v}_{2j} \in v_{2j}^*$, for $j \leq i$. Note that every symbol occurring in $\bar{v}_{2k}v_{2k+1} \dots v_{2\ell-1}\bar{v}_{2\ell}$ belongs to $\mathtt{Alph}(v_{2k})$. Then $\bar{v}_{2k}v_{2k+1} \dots v_{2\ell-1}\bar{v}_{2\ell}$ embeds to $(v_{2k})^{|w|}$, which implies that $w \preceq v_1\bar{v}_2 \dots v_{2k-1}(v_{2k})^{|w|}v_{2\ell+1} \dots \bar{v}_{2i}v_{2i+1}$ and, therefore, also

$$w \preceq v_1\bar{v}_2 \dots v_{2k-1}(v_{2k})^{|w|}v_{2k+1} \dots v_{2\ell-1}v_{2\ell+1} \dots \bar{v}_{2i}v_{2i+1} \in L',$$

which completes the proof. □

Using Lemma 21 we may eliminate unnecessary loops $(bb)^*$ in $L_4^{\mathcal{A}}$ and $(bcbc)^*$ (or, alternatively, $(cbc)^*$) in $L_4^{\mathcal{B}}$ obtaining

$$L_5^{\mathcal{A}} = a(ba)^*aabcabb(bc)^* \qquad \text{and} \qquad L_5^{\mathcal{B}} = b(aab)^*bacacc(cbc)^*b.$$

Note that these are the languages from Example 8.

We call a union-free regular expression of star depth at most one with expressions $v_k$ and $v_\ell$ as mentioned in Lemma 21 *redundant* since, intuitively, it has a redundant loop. A union-free regular expression of star depth at most one that is not redundant is called *nonredundant*. We use the same notions for the corresponding languages. In what follows, when we speak about a regular expression of a nonredundant or redundant language, we mean the corresponding nonredundant or redundant regular expression, respectively. A nonredundant regular expression of the form

$$v_1(v_2)^*v_3(v_4)^* \dots (v_{2k})^*v_{2k+1},$$

where $v_i \in \Sigma^*$, for $1 \leq i \leq 2k+1$, is called *saturated* if for any two loops $v_m$ and $v_n$ all symbols from $\mathtt{Alph}(v_m) \cup \mathtt{Alph}(v_n)$ occur in between. The language of a saturated expression is called *saturated*. The intuition behind this notion is explained below. The following lemma shows that we may assume that our languages are saturated.

**Lemma 22** (Unfolding loops)**.** *Let $L$ be a nonredundant language. Then there exists a saturated language $L' \subseteq L$ such that $L$ and $L'$ are mutually embeddable.*

*Proof.* Let the regular expression of $L$ be $r = v_1(v_2)^*v_3(v_4)^* \dots (v_{2k})^*v_{2k+1}$ and define $r' = v_1v_2(v_2)^*v_2v_3v_4(v_4)^*v_4 \dots v_{2k}(v_{2k})^*v_{2k}v_{2k+1}$, where all the loops are unfolded once in every direction and the corresponding language is $L' = L(r')$. The nonredundancy of $L'$ is clear. Indeed, if there are two loops $v_i$ and $v_j$ in $r'$ such that one of them has a bigger alphabet and every symbol in between $v_i$ and $v_j$ belongs to

this alphabet, then the situation also takes place before the unfolding of the loops, in the regular expression $r$. Furthermore, it is easy to see that $L'$ is saturated. It is thus sufficient to show the mutual embeddability. Note that $L(r') \subseteq L(r)$, hence $L(r') \preceq L(r)$. On the other hand, every word from $L(r)$ either belongs to $L(r')$ or embeds to a word of $L(r')$ obtained by unfolding some loops several times. □

After unfolding the loops $(ba)$ and $(bc)$ in $L_5^{\mathcal{A}}$ and $(aab)^*$ and $(cbc)^*$ in $L_5^{\mathcal{B}}$ we obtain

$$L_6^{\mathcal{A}} = aba(ba)^*baaabcabbbc(bc)^*bc$$

and

$$L_6^{\mathcal{B}} = baab(aab)^*aabbacacccbc(cbc)^*cbcb.$$

In fact languages $L_5^{\mathcal{A}}$ and $L_5^{\mathcal{B}}$ were already saturated, but this is not always true in general for nonredundant languages.

The $\Sigma_0$-decomposition of a saturated regular expression $r$ is of the form

$$r_1\ u_1(v_1)^*w_1\ r_2\ u_2(v_2)^*w_2 \dots\ r_k\ u_k(v_k)^*w_k\ r_{k+1},$$

where words $v_1, v_2, \dots, v_k$ are $\Sigma_0$-loops in $r$, words $u_i$ and $w_i$ satisfy $\mathtt{Alph}(u_i) \cup \mathtt{Alph}(w_i) \subseteq \Sigma_0$, for $1 \leq i \leq k$, and $r_1, r_2, \dots, r_{k+1}$ are nonredundant expressions without $\Sigma_0$-loops starting and ending with symbols not belonging to $\Sigma_0$.

Notice that the $\Sigma_0$-decomposition may not exist for non-saturated expressions. Consider for instance the expression $(ab)^*a(ac)^*$, and try to compute its $\{a,b\}$-decomposition. It does not exists, as, intuitively, there is no symbol outside $\{a,b\}$ between the $\{a,b\}$-loop and $\{a,c\}$-loop. Thus it is not possible to start an expression $r_2$ by symbol not belonging to $\{a,b\}$, as required above. This is the reason why we need to make it saturated, for example by unfolding the loops like in the proof of Lemma 22. Then we obtain the expression $ab(ab)^*abaac(ac)^*ac$, which has the $\{a,b\}$-decomposition of the form $r_1 = \varepsilon$, $u_1 = ab$, $v_1 = ab$, $w_1 = abaa$ and $r_2 = c(ac)^*ac$ starting with symbol outside $\{a,b\}$, as needed.

For two saturated regular expressions $r^{\mathcal{A}}$ and $r^{\mathcal{B}}$ we say that an alphabet $\Sigma_0 \subseteq \Sigma$ is $(r^{\mathcal{A}}, r^{\mathcal{B}})$-*loop-maximal* if

1. there exists a $\Sigma_0$-loop either in $r^{\mathcal{A}}$ or in $r^{\mathcal{B}}$; and
2. there is no $\Sigma' \supsetneq \Sigma_0$ for which a $\Sigma'$-loop occurs either in $r^{\mathcal{A}}$ or in $r^{\mathcal{B}}$.

If $r^{\mathcal{A}}$ and $r^{\mathcal{B}}$ are clear from the context we simply say that an alphabet $\Sigma_0 \subseteq \Sigma$ is *loop-maximal*.

**Lemma 23** (Decompositions)**.** *Let $L^{\mathcal{A}}$ and $L^{\mathcal{B}}$ be two saturated and mutually-embeddable languages with $r^{\mathcal{A}}$ and $r^{\mathcal{B}}$ being their saturated regular expressions. Let $\Sigma_0 \subseteq \Sigma$ be loop-maximal. Let the $\Sigma_0$-decomposition of $r^{\mathcal{A}}$ be*

$$r^{\mathcal{A}} = r_1^{\mathcal{A}}\ u_1^{\mathcal{A}}(v_1^{\mathcal{A}})^*w_1^{\mathcal{A}}\ r_2^{\mathcal{A}}\ u_2^{\mathcal{A}}(v_2^{\mathcal{A}})^*w_2^{\mathcal{A}} \dots\ r_k^{\mathcal{A}}\ u_k^{\mathcal{A}}(v_k^{\mathcal{A}})^*w_k^{\mathcal{A}}\ r_{k+1}^{\mathcal{A}}\,.$$

*Then the numbers of $\Sigma_0$-loops in $r^{\mathcal{A}}$ and $r^{\mathcal{B}}$ coincide. Moreover, the $\Sigma_0$-decomposition of $r^{\mathcal{B}}$ is*

$$r^{\mathcal{B}} = r_1^{\mathcal{B}}\ u_1^{\mathcal{B}}(v_1^{\mathcal{B}})^*w_1^{\mathcal{B}}\ r_2^{\mathcal{B}}\ u_2^{\mathcal{B}}(v_2^{\mathcal{B}})^*w_2^{\mathcal{B}} \dots\ r_k^{\mathcal{B}}\ u_k^{\mathcal{B}}(v_k^{\mathcal{B}})^*w_k^{\mathcal{B}}\ r_{k+1}^{\mathcal{B}}\,,$$

*where, for all $1 \leq i \leq k+1$, the languages $L(r_i^{\mathcal{A}})$ and $L(r_i^{\mathcal{B}})$ are mutually embeddable and saturated.*

*Proof.* If $v = a_1 \cdots a_k$ embeds into $w = b_1 \cdots b_l$ such that $v = b_{i_1} \cdots b_{i_k}$ for $i_1 < \ldots < i_k$ then we say that symbol $a_j$ *embeds into the position* $i_j$ with respect to this embedding. Usually, if embedding is clear from the context, we omit it.

We first show that both $r^{\mathcal{A}}$ and $r^{\mathcal{B}}$ have the same number of $\Sigma_0$-loops. For the sake of contradiction, assume that there are more $\Sigma_0$-loops in $r^{\mathcal{A}}$ than in $r^{\mathcal{B}}$. We will exploit the fact that $L^{\mathcal{A}} \preceq L^{\mathcal{B}}$. Let $m$ be the size of $r^{\mathcal{B}}$, i.e., the number of symbols in it, and consider an arbitrary word

$$v = s_1^{\mathcal{A}} u_1^{\mathcal{A}} (v_1^{\mathcal{A}})^{m+1} w_1^{\mathcal{A}} s_2^{\mathcal{A}} u_2^{\mathcal{A}} (v_2^{\mathcal{A}})^{m+1} w_2^{\mathcal{A}} \ldots s_k^{\mathcal{A}} u_k^{\mathcal{A}} (v_k^{\mathcal{A}})^{m+1} w_k^{\mathcal{A}} s_{k+1}^{\mathcal{A}} \in L^{\mathcal{A}},$$

where $s_i^{\mathcal{A}} \in L(r_i^{\mathcal{A}})$, for $i \leq k+1$. There is a word $w \in L^{\mathcal{B}}$ such that $v \preceq w$. Consider an arbitrary $v_j^{\mathcal{A}}$, for $1 \leq j \leq k$. There are at least $m+1$ occurrences of $v_j^{\mathcal{A}}$ in $v$ and for each one the last symbol of $v_j^{\mathcal{A}}$ coincides with a symbol of $r^{\mathcal{B}}$. As there are $m+1$ words $v_j^{\mathcal{A}}$ there are also $m+1$ positions in $r^{\mathcal{B}}$ in which their first symbol embeds. By the pigeonhole principle, at least two of them coincide in $r^{\mathcal{B}}$. Recall that there is no $\Sigma_0'$-loop for $\Sigma_0' \supsetneq \Sigma_0$ in $r^{\mathcal{B}}$. Thus some repeated position $x$ in $r^{\mathcal{B}}$ has to be inside some $\Sigma_0$-loop; otherwise, it would not be possible to read several words $v_j^{\mathcal{A}}$ and after this end up in the same position in $r^{\mathcal{B}}$. Therefore we define a mapping from $\Sigma_0$-loops in $r^{\mathcal{A}}$ to $\Sigma_0$-loops in $r^{\mathcal{B}}$, which maps a loop from $r^{\mathcal{A}}$ to some loop in $r^{\mathcal{B}}$ in which the above discussed repeated position occurs. Note that there possibly could be more than one such loop in $r^{\mathcal{B}}$, then we pick one of them.

We will show that no $\Sigma_0$-loop in $r^{\mathcal{B}}$ is assigned to two different $\Sigma_0$-loops $v_i^{\mathcal{A}}$ and $v_j^{\mathcal{A}}$ from $r^{\mathcal{A}}$. Assume, to the contrary, that both $v_i^{\mathcal{A}}$ and $v_j^{\mathcal{A}}$, for $i < j$, are mapped to the same $\Sigma_0$-loop $v_s^{\mathcal{B}}$ in $r^{\mathcal{B}}$. Thus every symbol in between $v_i^{\mathcal{A}}$ and $v_j^{\mathcal{A}}$ have to embed in some position in the loop $v_s^{\mathcal{B}}$. However, recall that there exists a symbol $a \notin \Sigma_0$ in $r_j^{\mathcal{A}}$ between the loops $v_i^{\mathcal{A}}$ and $v_j^{\mathcal{A}}$, while loop $v_s^{\mathcal{B}}$ contains only symbols from $\Sigma_0$. This leads to the contradiction. Therefore, in particular, there are not more $\Sigma_0$-loops in $r^{\mathcal{A}}$ than in $r^{\mathcal{B}}$.

Thus, we may assume that the $\Sigma_0$-decomposition of $r^{\mathcal{B}}$ is of the form

$$r^{\mathcal{B}} = r_1^{\mathcal{B}}\ u_1^{\mathcal{B}}(v_1^{\mathcal{B}})^* w_1^{\mathcal{B}}\ r_2^{\mathcal{B}}\ u_2^{\mathcal{B}}(v_2^{\mathcal{B}})^* w_2^{\mathcal{B}} \ldots\ r_k^{\mathcal{B}}\ u_k^{\mathcal{B}}(v_k^{\mathcal{B}})^* w_k^{\mathcal{B}}\ r_{k+1}^{\mathcal{B}}\,.$$

By definition of the $\Sigma_0$-decomposition all $r_i^{\mathcal{A}}$ and $r_i^{\mathcal{B}}$ are nonredundant. It remains to show that the languages $L(r_i^{\mathcal{A}})$ and $L(r_i^{\mathcal{B}})$ are mutually embedded. Fix an index $i$. We show that $L(r_i^{\mathcal{A}}) \preceq L(r_i^{\mathcal{B}})$ since the other direction is analogous. Assume, to the contrary, that a word $u \in L(r_i^{\mathcal{A}})$ does not embed to $L(r_i^{\mathcal{B}})$. Note that the word $v$ above was chosen arbitrarily, with the only restriction that $\Sigma_0$-loops were repeated $m+1$ times each. Thus, put $s_i^{\mathcal{A}} = u$ and consider the position in word $w$ where the last symbol of $u$ could embed inspecting $r^{\mathcal{B}}$ from left to right. As shown above, $u$ cannot embed earlier than in $v_{i-1}^{\mathcal{B}}$. Recall that the last symbol of $s_i^{\mathcal{A}}$ does not belong to $\Sigma_0$, thus it does not embed to $v_{i-1}^{\mathcal{B}}$ and $w_{i-1}^{\mathcal{B}}$. As $u \not\preceq L(r_i^{\mathcal{B}})$, the last symbol of $u$ does not embed to the infix of $w$ corresponding to $r_i^{\mathcal{B}}$. One more time, as the last symbol of $s_i^{\mathcal{A}}$ does not belong to $\Sigma_0$ it does not embed to $u_i^{\mathcal{B}}(v_i^{\mathcal{B}})^* w_i^{\mathcal{B}}$. Thus, the first position where it could embed is somewhere in $r_{i+1}^{\mathcal{B}}$. Then, however, we have to assign $\Sigma_0$-loops corresponding to words $v_{i+1}^{\mathcal{A}}, \ldots, v_k^{\mathcal{A}}$ to $\Sigma_0$-loops corresponding to words $v_{i+2}^{\mathcal{B}}, \ldots, v_k^{\mathcal{B}}$ in (as shown above) an injective way, which is not possible. □

*Proof of Lemma 9.* It is easy to see that if the languages $K^{\mathcal{A}}$ and $K^{\mathcal{B}}$ are nonempty

and synchronized then there exists an infinite zigzag between them, thus also between languages $L^{\mathcal{A}}$ and $L^{\mathcal{B}}$.

To prove the opposite implication, assume that there exists an infinite zigzag between the languages $L^{\mathcal{A}}$ and $L^{\mathcal{B}}$. Applying Lemma 19 first we obtain nonempty union-free mutually-embeddable languages $M^{\mathcal{A}} \subseteq L^{\mathcal{A}}$ and $M^{\mathcal{B}} \subseteq L^{\mathcal{B}}$. Then, using Lemma 20, several times Lemma 21 and, finally, Lemma 22 we obtain languages $K^{\mathcal{A}}$ and $K^{\mathcal{B}}$ represented by saturated (thus also union free of star depth one and nonredundant) regular expressions that are mutually embeddable to the languages $M^{\mathcal{A}}$ and $M^{\mathcal{B}}$, respectively. As the mutual-embeddability relation is transitive, $K^{\mathcal{A}}$ and $K^{\mathcal{B}}$ are mutually embeddable. Note that $K^{\mathcal{A}} \subseteq M^{\mathcal{A}}$ and $K^{\mathcal{B}} \subseteq M^{\mathcal{B}}$ as the application of Lemmas 20, 21 and 22 results in sublanguages of the original languages. To complete the proof, we show that they are synchronized.

Consider the regular expressions $r^{\mathcal{A}}$ and $r^{\mathcal{B}}$ (with the properties listed above) for $K^{\mathcal{A}}$ and $K^{\mathcal{B}}$, respectively, and denote the number of loops in $r^{\mathcal{A}}$ by $i_{\mathcal{A}}$ and in $r^{\mathcal{B}}$ by $i_{\mathcal{B}}$. We prove the rest of the lemma by induction on $i_{\mathcal{A}} + i_{\mathcal{B}}$. For $i_{\mathcal{A}} + i_{\mathcal{B}} = 0$, $i_1 = i_2 = 0$ and $K^{\mathcal{A}} = \{w_1\}$ and $K^{\mathcal{B}} = \{w_2\}$, for some $w_1, w_2 \in \Sigma^*$. As there exists an infinite zigzag between $K^{\mathcal{A}}$ and $K^{\mathcal{B}}$, we have $w_1 = w_2$ and, hence, $K^{\mathcal{A}}$ and $K^{\mathcal{B}}$ are synchronized in one step. Note that this is the place where we use that the languages are not necessarily disjoint.

Assume that $i_{\mathcal{A}} + i_{\mathcal{B}} = k > 0$. Fix an alphabet $\Sigma_0$ which is $(r^{\mathcal{A}}, r^{\mathcal{B}})$-loop-maximal. Then, by Lemma 23, we obtain that the $\Sigma_0$-decomposition of $r^{\mathcal{A}}$ and $r^{\mathcal{B}}$ are

$$r^{\mathcal{A}} = s_1^{\mathcal{A}}\ u_1^{\mathcal{A}}(v_1^{\mathcal{A}})^* w_1^{\mathcal{A}}\ s_2^{\mathcal{A}}\ u_2^{\mathcal{A}}(v_2^{\mathcal{A}})^* w_2^{\mathcal{A}} \dots s_k^{\mathcal{A}}\ u_k^{\mathcal{A}}(v_k^{\mathcal{A}})^* w_k^{\mathcal{A}}\ s_{k+1}^{\mathcal{A}}$$

and

$$r^{\mathcal{B}} = s_1^{\mathcal{B}}\ u_1^{\mathcal{B}}\,(v_1^{\mathcal{B}})^*\, w_1^{\mathcal{B}}\ s_2^{\mathcal{B}}\ u_2^{\mathcal{B}}\,(v_2^{\mathcal{B}})^*\, w_2^{\mathcal{B}} \dots s_k^{\mathcal{B}}\ u_k^{\mathcal{B}}\,(v_k^{\mathcal{B}})^*\, w_k^{\mathcal{B}}\ s_{k+1}^{\mathcal{B}}$$

where the languages $L(s_i^{\mathcal{A}})$ and $L(s_i^{\mathcal{B}})$ are mutually embeddable and saturated for all $1 \leq i \leq k+1$. Thus, by induction hypothesis, all $L(s_i^{\mathcal{A}})$ and $L(s_i^{\mathcal{B}})$ are synchronized. As, by definition, $u_i^{\mathcal{A}}(v_i^{\mathcal{A}})^* w_i^{\mathcal{A}}$ and $u_i^{\mathcal{B}}(v_i^{\mathcal{B}})^* w_i^{\mathcal{B}}$ are synchronized in one step, for all $1 \leq i \leq k$, we have that $r^{\mathcal{A}}$ and $r^{\mathcal{B}}$ are synchronized, which completes the proof. □

## Remaining Proofs of Section 4

**Lemma 10.** *For two NFAs $\mathcal{A}$ and $\mathcal{B}$, the following conditions are equivalent.*

1. *Automata $\mathcal{A}$ and $\mathcal{B}$ are synchronizable.*
2. *There exist synchronized languages $K^{\mathcal{A}} \subseteq L(\mathcal{A})$ and $K^{\mathcal{B}} \subseteq L(\mathcal{B})$.*

*Proof.* The implication from left to right is immediate. To prove the opposite implication, let $K^{\mathcal{A}} = D_1^{\mathcal{A}} \dots D_k^{\mathcal{A}}$ and $K^{\mathcal{B}} = D_1^{\mathcal{B}} \dots D_k^{\mathcal{B}}$, where $D_i^{\mathcal{A}}$ and $D_i^{\mathcal{B}}$ are synchronized in one step, for all $1 \leq i \leq k$. Define the *$n$-th canonical word* of a singleton language as its unique word, and of a cycle language $v_{\mathtt{pref}}(v_{\mathtt{mid}})^* v_{\mathtt{suff}}$ as the word $v_{\mathtt{pref}}(v_{\mathtt{mid}})^n v_{\mathtt{suff}}$. Let $N$ be the maximum number of states of automata $\mathcal{A}$ and $\mathcal{B}$, and let $w_i^{\mathcal{A}}$ and $w_i^{\mathcal{B}}$ be the $N$-th canonical words of languages $D_i^{\mathcal{A}}$ and $D_i^{\mathcal{B}}$, respectively, for $1 \leq i \leq k$. Let $w^{\mathcal{A}} = w_1^{\mathcal{A}} \cdots w_k^{\mathcal{A}}$ and $w^{\mathcal{B}} = w_1^{\mathcal{B}} \cdots w_k^{\mathcal{B}}$. Notice that $w^{\mathcal{A}} \in K^{\mathcal{A}} \subseteq L(\mathcal{A})$ and $w^{\mathcal{B}} \in K^{\mathcal{B}} \subseteq L(\mathcal{B})$. Consider some of the accepting runs of $\mathcal{A}$ on $w^{\mathcal{A}}$ and of $\mathcal{B}$ on $w^{\mathcal{B}}$, respectively,

$$q_1^{\mathcal{A}} \xrightarrow{w_1^{\mathcal{A}}} q_2^{\mathcal{A}} \xrightarrow{w_2^{\mathcal{A}}} \dots \xrightarrow{w_{k-1}^{\mathcal{A}}} q_k^{\mathcal{A}} \xrightarrow{w_k^{\mathcal{A}}} q_{k+1}^{\mathcal{A}}$$

and

$$q_1^{\mathcal{B}} \xrightarrow{w_1^{\mathcal{B}}} q_2^{\mathcal{B}} \xrightarrow{w_2^{\mathcal{B}}} \dots \xrightarrow{w_{k-1}^{\mathcal{B}}} q_k^{\mathcal{B}} \xrightarrow{w_k^{\mathcal{B}}} q_{k+1}^{\mathcal{B}} .$$

By definition of run, states $q_1^{\mathcal{A}}$ and $q_1^{\mathcal{B}}$ are initial respectively in $\mathcal{A}$ and $\mathcal{B}$, and states $q_{k+1}^{\mathcal{A}}$ and $q_{k+1}^{\mathcal{B}}$ are accepting in $\mathcal{A}$ and $\mathcal{B}$, respectively. Thus, to show that $\mathcal{A}$ and $\mathcal{B}$ are synchronizable, it is sufficient to show that pairs $(q_i^{\mathcal{A}}, q_i^{\mathcal{B}})$ and $(q_{i+1}^{\mathcal{A}}, q_{i+1}^{\mathcal{B}})$ are synchronizable, for all $1 \leq i \leq k$. Fix some $1 \leq i \leq k$, then there are two cases. Either both $D_i^{\mathcal{A}}$ and $D_i^{\mathcal{B}}$ are singletons, or they are cycle languages. Consider first the situation when they are singletons. Then we have $w_i^{\mathcal{A}} = w_i^{\mathcal{B}} \in D_i^{\mathcal{A}} = D_i^{\mathcal{B}}$ and pairs $(q_i^{\mathcal{A}}, q_i^{\mathcal{B}})$ and $(q_{i+1}^{\mathcal{A}}, q_{i+1}^{\mathcal{B}})$ are clearly synchronizable.

Focus now on the situation where $D_i^{\mathcal{A}}$ and $D_i^{\mathcal{B}}$ are cycle languages. In this case,

$$D_i^{\mathcal{A}} = v_{\mathtt{pref}}^{\mathcal{A}} (v_{\mathtt{mid}}^{\mathcal{A}})^* v_{\mathtt{suff}}^{\mathcal{A}} \qquad \text{and} \qquad D_i^{\mathcal{B}} = v_{\mathtt{pref}}^{\mathcal{B}} (v_{\mathtt{mid}}^{\mathcal{B}})^* v_{\mathtt{suff}}^{\mathcal{B}} ,$$

for some $v_{\mathtt{pref}}^{\mathcal{A}}, v_{\mathtt{mid}}^{\mathcal{A}}, v_{\mathtt{suff}}^{\mathcal{A}}, v_{\mathtt{pref}}^{\mathcal{B}}, v_{\mathtt{mid}}^{\mathcal{B}}, v_{\mathtt{suff}}^{\mathcal{B}} \in \Sigma^*$; and

$$w_i^{\mathcal{A}} = v_{\mathtt{pref}}^{\mathcal{A}} (v_{\mathtt{mid}}^{\mathcal{A}})^N v_{\mathtt{suff}}^{\mathcal{A}} \qquad \text{and} \qquad w_i^{\mathcal{B}} = v_{\mathtt{pref}}^{\mathcal{B}} (v_{\mathtt{mid}}^{\mathcal{B}})^N v_{\mathtt{suff}}^{\mathcal{B}} .$$

Consider a run of $\mathcal{A}$ on $w_i^{\mathcal{A}}$ from $q_i^{\mathcal{A}}$ to $q_{i+1}^{\mathcal{A}}$. It is of the form

$$q_i^{\mathcal{A}} \xrightarrow{v_{\mathtt{pref}}^{\mathcal{A}}} m_0^{\mathcal{A}} \xrightarrow{v_{\mathtt{mid}}^{\mathcal{A}}} m_1^{\mathcal{A}} \xrightarrow{v_{\mathtt{mid}}^{\mathcal{A}}} \dots \xrightarrow{v_{\mathtt{mid}}^{\mathcal{A}}} m_{N-1}^{\mathcal{A}} \xrightarrow{v_{\mathtt{mid}}^{\mathcal{A}}} m_N^{\mathcal{A}} \xrightarrow{v_{\mathtt{suff}}^{\mathcal{A}}} q_{i+1}^{\mathcal{A}} ,$$

for some states $m_j^{\mathcal{A}}$, for $0 \leq j \leq N$. Notice that at least two among states $m_0^{\mathcal{A}}, \dots, m_N^{\mathcal{A}}$ necessarily coincide, as automaton $\mathcal{A}$ has no more than $N$ states. Assume thus that for some $0 \leq k < \ell \leq N$ we have $m_k^{\mathcal{A}} = m_\ell^{\mathcal{A}} = m^{\mathcal{A}}$. Then

$$q_i^{\mathcal{A}} \xrightarrow{v_{\mathtt{pref}}^{\mathcal{A}} (v_{\mathtt{mid}}^{\mathcal{A}})^k} m^{\mathcal{A}} \xrightarrow{(v_{\mathtt{mid}}^{\mathcal{A}})^{\ell-k}} m^{\mathcal{A}} \xrightarrow{(v_{\mathtt{mid}}^{\mathcal{A}})^{N-\ell} v_{\mathtt{suff}}^{\mathcal{A}}} q_{i+1}^{\mathcal{A}} ,$$

which shows that states $q_i^{\mathcal{A}}$ and $q_{i+1}^{\mathcal{A}}$ are $\mathtt{Alph}(v_{\mathtt{mid}}^{\mathcal{A}})$-connected in $\mathcal{A}$, since we have $\mathtt{Alph}(v_{\mathtt{pref}}^{\mathcal{A}}) \cup \mathtt{Alph}(v_{\mathtt{suff}}^{\mathcal{A}}) \subseteq \mathtt{Alph}(v_{\mathtt{mid}}^{\mathcal{A}})$ by definition of languages synchronized in one step. Similarly we can show that $q_i^{\mathcal{B}}$ and $q_{i+1}^{\mathcal{B}}$ are $\mathtt{Alph}(v_{\mathtt{mid}}^{\mathcal{B}})$-connected in $\mathcal{B}$. However, by definition of synchronization in one step, the cycle alphabets of $D_i^{\mathcal{A}}$ and $D_i^{\mathcal{B}}$ are the same, so $\mathtt{Alph}(v_{\mathtt{mid}}^{\mathcal{A}}) = \mathtt{Alph}(v_{\mathtt{mid}}^{\mathcal{A}})$. This shows that the pairs $(q_i^{\mathcal{A}}, q_i^{\mathcal{B}})$ and $(q_{i+1}^{\mathcal{A}}, q_{i+1}^{\mathcal{B}})$ are synchronizable and completes the proof. □

**Theorem 11.** *Let $\mathcal{A}$ and $\mathcal{B}$ be two NFAs. Then the languages $L(\mathcal{A})$ and $L(\mathcal{B})$ are separable by piecewise testable languages if and only if the automata $\mathcal{A}$ and $\mathcal{B}$ are not synchronizable.*

*Proof.* This theorem follows from the previous results. Namely, by Theorem 3, the languages $L(\mathcal{A})$ and $L(\mathcal{B})$ are separable by piecewise testable languages if and only if there is no infinite zigzag between them. Lemma 9 shows that the existence of a zigzag is equivalent to the existence of two synchronized sublanguages $K^{\mathcal{A}} \subseteq L(\mathcal{A})$ and $K^{\mathcal{B}} \subseteq L(\mathcal{B})$. Finally, by Lemma 10, the existence of two synchronized sublanguages is equivalent to the fact that the automata $\mathcal{A}$ and $\mathcal{B}$ are synchronizable, which concludes the proof. □

**Theorem 12.** *Given two NFAs $\mathcal{A}$ and $\mathcal{B}$, it is possible to test in polynomial time whether $L(\mathcal{A})$ and $L(\mathcal{B})$ can be separated by a piecewise testable language.*

*Proof.* By Theorem 11 it is enough to check whether $\mathcal{A}$ and $\mathcal{B}$ are synchronizable. Let $\mathcal{A} = (Q^{\mathcal{A}}, \Sigma, \delta^{\mathcal{A}}, q_0^{\mathcal{A}}, F^{\mathcal{A}})$ and $\mathcal{B} = (Q^{\mathcal{B}}, \Sigma, \delta^{\mathcal{B}}, q_0^{\mathcal{B}}, F^{\mathcal{B}})$. We will consider the graph SYNCH for which the vertices are pairs of states of $Q^{\mathcal{A}} \times Q^{\mathcal{B}}$ and the edges correspond to pairs of vertices synchronizable in one step. Specifically, there is an edge $(p^{\mathcal{A}}, p^{\mathcal{B}}) \to (q^{\mathcal{A}}, q^{\mathcal{B}})$ in SYNCH if and only if $(p^{\mathcal{A}}, p^{\mathcal{B}})$ and $(q^{\mathcal{A}}, q^{\mathcal{B}})$ are synchronizable in one step. Thus, $\mathcal{A}$ and $\mathcal{B}$ are synchronizable if and only if a vertex consisting of accepting states is reachable in SYNCH from the pair of initial states $(q_0^{\mathcal{A}}, q_0^{\mathcal{B}})$. Since reachability is testable in PTIME, it is thus sufficient to show how we compute the edges of SYNCH.

The definition of synchronizability in one step (page 8) consists of two cases. We refer to the first case as *symbol synchronization* and to the second case as *cycle synchronization*.

For two symbol-synchronizable pairs of states $(p^{\mathcal{A}}, p^{\mathcal{B}})$, $(q^{\mathcal{A}}, q^{\mathcal{B}})$, there should be an edge $(p^{\mathcal{A}}, p^{\mathcal{B}}) \to (q^{\mathcal{A}}, q^{\mathcal{B}})$ in SYNCH if there exists an $a \in \Sigma$ such that $p^{\mathcal{A}} \xrightarrow{a} q^{\mathcal{A}}$ and $p^{\mathcal{B}} \xrightarrow{a} q^{\mathcal{B}}$. Since it is easy to find all these pairs in polynomial time, these edges in SYNCH can be easily constructed.

We now show how to construct the edges for cycle-synchronized states. For two pairs $(p^{\mathcal{A}}, p^{\mathcal{B}})$ and $(q^{\mathcal{A}}, q^{\mathcal{B}})$ to be cycle-synchronizable, we require that $p^{\mathcal{A}}$ and $q^{\mathcal{A}}$ are $\Sigma_0$-connected in $\mathcal{A}$, and $p^{\mathcal{B}}$ and $q^{\mathcal{B}}$ are $\Sigma_0$-connected in $\mathcal{B}$, for (the same) $\Sigma_0 \subseteq \Sigma$. We now rephrase this definition using other notions that will be useful in the algorithm.

A pair of states $(p^{\mathcal{A}}, p^{\mathcal{B}}) \in Q^{\mathcal{A}} \times Q^{\mathcal{B}}$ has a *saturated* $\Sigma_0$*-cycle* if there exist two words $v^{\mathcal{A}}, v^{\mathcal{B}}$ satisfying

1. $\mathtt{Alph}(v^{\mathcal{A}}) = \mathtt{Alph}(v^{\mathcal{B}}) = \Sigma_0$;
2. $p^{\mathcal{A}} \xrightarrow{v^{\mathcal{A}}} p^{\mathcal{A}}$ in $\mathcal{A}$; and
3. $p^{\mathcal{B}} \xrightarrow{v^{\mathcal{B}}} p^{\mathcal{B}}$ in $\mathcal{B}$.

We say that there is a $\Sigma_0$*-route* from $(p^{\mathcal{A}}, p^{\mathcal{B}}) \in Q^{\mathcal{A}} \times Q^{\mathcal{B}}$ to $(q^{\mathcal{A}}, q^{\mathcal{B}}) \in Q^{\mathcal{A}} \times Q^{\mathcal{B}}$ if there exist words $v^{\mathcal{A}}$ and $v^{\mathcal{B}}$ in $\Sigma_0^*$ such that $p^{\mathcal{A}} \xrightarrow{v^{\mathcal{A}}} q^{\mathcal{A}}$ and $p^{\mathcal{B}} \xrightarrow{v^{\mathcal{B}}} q^{\mathcal{B}}$. So, in contrast to saturated $\Sigma_0$-cycles, here we do not require that the alphabets $\mathtt{Alph}(v^{\mathcal{A}})$ and $\mathtt{Alph}(v^{\mathcal{B}})$ are equal to $\Sigma_0$.

Note that if a pair $V = (q^{\mathcal{A}}, q^{\mathcal{B}})$ has a saturated $\Sigma_0$-cycle and a saturated $\Sigma_1$-cycle, then it also has a saturated $(\Sigma_0 \cup \Sigma_1)$-cycle (obtained by the concatenation of the two cycles). Thus, for every pair $V = (q^{\mathcal{A}}, q^{\mathcal{B}})$, there exists a unique maximal alphabet $\Sigma_0 \subseteq \Sigma$ such that it has a saturated $\Sigma_0$-cycle. (This unique maximal alphabet can be empty if no such saturated cycle exists.) We call this alphabet the *saturated cycle alphabet* of $V$ and denote it by $\Sigma_0^V$. This means that $V_p = (p^{\mathcal{A}}, p^{\mathcal{B}})$ and $V_q = (q^{\mathcal{A}}, q^{\mathcal{B}})$ are cycle synchronizable if and only if there is a $V$ such that there are $\Sigma_0^V$-routes from $V_p$ to $V$ and from $V$ to $V_q$.

To find all the cycle synchronizable pairs we can first compute, for every $V = (p^{\mathcal{A}}, p^{\mathcal{B}})$, the saturated cycle alphabet $\Sigma_0^V$. This can be done in polynomial time in the following manner. Let $C_0^{\mathcal{A}}$ and $C_0^{\mathcal{B}}$ be the strongly connected components of $\mathcal{A}$ and $\mathcal{B}$ containing $p^{\mathcal{A}}$ and $p^{\mathcal{B}}$, respectively. For a strongly connected component $C$, let $\mathtt{Alph}(C)$ be the union of all symbols $a$ of $\Sigma$ that label transitions of the form $p \xrightarrow{a} q$, where both $p$ and $q$ belong to $C$. If $\mathtt{Alph}(C_0^{\mathcal{A}}) = \mathtt{Alph}(C_0^{\mathcal{B}})$, then

$\Sigma_0^V$ equals $\texttt{Alph}(C_0^{\mathcal{A}})$. Otherwise, set $\Sigma_1 = \texttt{Alph}(C_0^{\mathcal{A}}) \cap \texttt{Alph}(C_0^{\mathcal{B}})$ and consider automata $\mathcal{A}_1$ and $\mathcal{B}_1$ obtained from $\mathcal{A}$ and $\mathcal{B}$ by removing all transitions labeled by symbols from $\Sigma \setminus \Sigma_1$. Consider the strongly connected components $C_1^{\mathcal{A}}$ and $C_1^{\mathcal{B}}$ of $\mathcal{A}_1$ and $\mathcal{B}_1$ containing $p^{\mathcal{A}}$ and $p^{\mathcal{B}}$, respectively, and proceed in the same way as before. Continuing this procedure we obtain a sequence of decreasing alphabets $\Sigma_1 \supsetneq \Sigma_2 \supsetneq \ldots$, hence we perform at most $|\Sigma|$ iterations. If we arrive at the empty alphabet then we say $\Sigma_0^V = \emptyset$.

We argue that we compute $\Sigma_0^V$ correctly. Clearly, if the algorithm returns a set $\Sigma'$, then $\Sigma' \subseteq \Sigma_0^V$. Conversely, we have that $\Sigma_0^V \subseteq \Sigma'$ because, at each point in the algorithm, the alphabet under consideration contains $\Sigma_0^V$. (In the first iteration, $\Sigma_0^V \subseteq \texttt{Alph}(C_0^{\mathcal{A}})$ and $\Sigma_0^V \subseteq \texttt{Alph}(C_0^{\mathcal{B}})$. Furthermore, at each iteration $i$, if $\Sigma_0^V \subseteq \texttt{Alph}(C_i^{\mathcal{A}})$ and $\Sigma_0^V \subseteq \texttt{Alph}(C_i^{\mathcal{B}})$, then $\Sigma_0^V \subseteq (\texttt{Alph}(C_i^{\mathcal{A}}) \cap \texttt{Alph}(C_i^{\mathcal{B}}))$.)

Once we know, for each pair $V = (q^{\mathcal{A}}, q^{\mathcal{B}})$, its saturated cycle alphabet $\Sigma_0^V$, we can find all vertices $V_p$ such that there is a $\Sigma_0^V$-route from $V_p$ to $V$ and all vertices $V_q$ such that there is a $\Sigma_0^V$-route from $V$ to $V_q$, and add edges $V_p \to V_q$ to the graph `SYNCH`. This concludes the construction of `SYNCH` and the presentation of the algorithm. We note that our algorithm is clearly not yet time-optimal. □

# Proofs of Section 5

The goal is to prove the following Theorem.

**Theorem 13.** *For $O \in \{\preceq, \preceq_s\}$ and $C$ being one of single, unions, or boolean combinations, we have that the complexity of the separation problem by $\mathcal{F}(O, C)$ is as indicated in Table 1.*

## The Subsequence Order Cases

**Lemma 24.** *The separation problem by $\mathcal{F}(\preceq, single)$ is NP-complete.*

*Proof.* Let $K$ and $L$ be two regular languages over $\Sigma$ given by NFAs. The problem is to find a word $w$ in $\Sigma^*$ such that $K \subseteq \text{closure}^{\preceq}(w)$ and $L \cap \text{closure}^{\preceq}(w) = \emptyset$. By definition of the subsequence order $\preceq$, the maximal length of such a word $w$ is equal to the length of a shortest word of $K$. Therefore, such a $w$ cannot be longer than the size of the automaton for $K$. An NP algorithm can guess such a word $w$ of length at most the size of the automaton and computes the minimal DFA for $\text{closure}^{\preceq}(w)$. This minimal DFA corresponds to a "greedy" procedure for embedding $w$ in a given string. That is, the states of this DFA correspond to the maximal prefix of $w$ that can be embedded in the currently read string. It can be computed in polynomial time from $w$. Verifying if $K \subseteq \text{closure}^{\preceq}(w)$ and $L \cap \text{closure}^{\preceq}(w) = \emptyset$ then reduces to standard automata constructions that can be done in polynomial time.

To show NP-hardness, we use a simple reduction of the longest common subsequence problem, which is well known to be NP-hard [17]. A word $w$ is a *longest common subsequence* of words $(w_i)_{i=1}^n$ if $w \preceq w_i$ for all $1 \leq i \leq n$ and there is no longer word with this property. This word $w$ is not necessarily unique (the longest common subsequence for $ab$ and $ba$ could be $a$ or $b$). By [17], to determine whether the length of the longest common subsequences of words $(w_i)_{i=1}^n$ is longer than a given $k$ is NP-hard with respect to $\sum_{i=1}^n |w_i|$ and $k$.

Consider the DFA $\mathcal{A}$ that accepts the finite language $K = \{w_1, \ldots, w_n\}$ and the DFA $\mathcal{B}$ that accepts the language $L$ of all words up to length $k-1$. Then we have that the existence of a common subsequence of $(w_i)_{i=1}^n$ longer than $k$ is then equivalent to the possibility to separate $K$ and $L$ by $\mathcal{F}(\preceq, \text{single})$. Furthermore, we can construct $\mathcal{A}$ in time $O(\sum_{i=1}^n w_i)$ and $\mathcal{B}$ in time $O(k \cdot \sum_{i=1}^n w_i)$. Since both $\mathcal{A}$ and $\mathcal{B}$ are DFAs, we have shown that the problem even remains NP-hard if the input is given as DFAs instead of NFAs. □

Actually, using the proof of Lemma 24 we can prove the same result for union-free languages.

**Lemma 25.** *The separation problem by union-free languages is NP-complete.*

*Proof.* The proof of NP-hardness of the proof of Lemma 24 also applies to union-free languages since the language $\text{closure}^{\preceq}(a_1 a_2 \ldots a_n) = \Sigma^* a_1 \Sigma^* a_2 \Sigma^* \ldots \Sigma^* a_n \Sigma^*$ is union free. Indeed, any regular expression $(b_1 + b_2 + \ldots + b_m)^* = (b_1^* \ldots b_m^*)^*$ is union free. The NP algorithm guesses a word $w$ as above and the positions and scopes of star operators. □

We now turn to separation by $\mathcal{F}(\preceq, \text{unions})$.

**Lemma 26.** *A language $K$ is separable from a language $L$ by $\mathcal{F}(\preceq, \text{unions})$ if and only if there exist no words $w \in K$ and $w' \in L$ such that $w \preceq w'$.*

*Proof.* If there exist $w \in K$ and $w' \in L$ with $w \preceq w'$, then any $\preceq$-closed language containing $w$ also contains $w'$. Since unions of $\preceq$-closed languages are also $\preceq$-closed, we have that $K$ is not separable from $L$ by $\mathcal{F}_{\text{u}}^{\preceq}$.

The opposite implication follows directly from Claim 17. Observe that in this case $\text{layer}(K, L) = K$ and every $\preceq$-closed language is a finite union of languages $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$ due to Higman's lemma. □

The words $w$ and $w'$ from the statement of Lemma 26 exist iff $\text{closure}^{\preceq}(K) \cap L = \emptyset$. An NFA for $\text{closure}^{\preceq}(K)$ is obtained by adding self loops under all symbols in $\Sigma$ to all states of the automaton for $K$. Emptiness of intersection is then decidable in polynomial time by standard methods. This gives the following lemma.

**Lemma 27.** *The separation problem by $\mathcal{F}(\preceq, \text{unions})$ is in polynomial time.*

## The Suffix Order Cases

It remains to prove the cases for the suffix order $\preceq_s$. Let $\text{lcs}(L)$ denote the longest common suffix of all words of language $L$.

**Lemma 28.** *A language $K$ is separable from a language $L$ by $\mathcal{F}(\preceq_s, \text{single})$ if and only if there is no word $w' \in L$ such that $\text{lcs}(K) \preceq_s w'$.*

*Proof.* The separation problem asks to check the existence of a word $w \in \Sigma^+$ such that $K \subseteq \Sigma^* w$ and $\Sigma^* w \cap L = \emptyset$. Obviously, if such a word exits, it must be a common suffix of all words from $K$. Assume that there is no $w' \in L$ such that $\text{lcs}(K) \preceq_s w'$. Then $K$ is separable from $L$ by the language $\Sigma^* \text{lcs}(K)$. To show the opposite implication, assume that there exists a $w' \in L$ such that $\text{lcs}(K) \preceq_s w'$. Then, for any common suffix $w$ of $K$, it holds that $w \preceq_s w'$, which means that $K$ is not separable from $L$ by a language from $\mathcal{F}(\preceq_s, \text{single})$. □

The word $\mathrm{lcs}(K)$ can be computed from the automaton for $K$ in polynomial time by inspecting paths that end up in accepting states. The length of $\mathrm{lcs}(K)$ is not larger than the length of the shortest word in $K$, hence linear with respect to the size of the automaton. To check whether there exists $w' \in L$ such that $\mathrm{lcs}(K) \preceq_s w'$ can be done in polynomial time by testing non-emptiness of the language $\Sigma^* \mathrm{lcs}(K) \cap L$.

**Lemma 29.** *The separation problem by $\mathcal{F}(\preceq_s, \mathit{single})$ is in polynomial time.*

**Lemma 30.** *A language $K$ is separable from a language $L$ by $\mathcal{F}(\preceq_s, \mathit{unions})$ if and only if the following two conditions are satisfied:*

1. *there exist no words $w \in K$ and $w' \in L$ such that $w \preceq_s w'$,*
2. *there exists a natural number $k \geq 0$ such that no words $w \in K$ and $w' \in L$ have a common suffix of length $k$.*

*Proof.* From left to right. Assume that $K$ is separable from $L$ by a language $S = \bigcup_{i=1}^{n} \Sigma^* w_i$. If $w \in K$ and $w \in S$ then there is no $w' \in L$ such that $w \preceq_s w'$ since $S$ contains all words that are longer than $w$ in $\preceq_s$ and $S \cap L = \emptyset$. Assume that for every number $k$ there are words $w \in K$ and $w' \in L$ with a common suffix of length $k$. Then, in particular, there are words $w \in K$ and $w' \in L$ with a common suffix of length $\max(|w_1|, \ldots, |w_n|) + 1$. However, these words are either both inside $S$ or both outside $S$, which contradicts that $K$ is separable from $L$ by $S$. This concludes the proof from left to right.

For the other direction, assume that $K$ and $L$ satisfy conditions 1 and 2. Let $M = \{w \in \Sigma^* \mid |w| \leq k \text{ and there is no } w' \in L \text{ such that } w \preceq_s w'\}$ and define $S = \bigcup_{w \in M} \Sigma^* w$. By definition, $S \cap L = \emptyset$ and $S$ is a finite union of suffix languages, i.e., a finite union of $\preceq_s$-closures of words. We show that $K \subseteq S$. Indeed, let $w \in K$. If $|w| \geq k$ and $v$ is a suffix of $w$ of length $k$ then $v$ belongs to $M$, which implies that $w \in \Sigma^* v \subseteq S$. If $|w| < k$ then $w \in M$ since there is no $w' \in L$ such that $w \preceq_s w'$. Thus, $w \in S$, which completes the proof. □

We now argue that the two conditions in Lemma 30 can be tested in polynomial time, given NFAs for $K$ and $L$. To check the first condition we test in polynomial time whether $(\Sigma^* K) \cap L$ is nonempty. To decide the second condition we compute the reversals $\mathrm{rev}(K)$ and $\mathrm{rev}(L)$ of languages $K$ and $L$, respectively. This is done by reversing transitions in the corresponding automata and swapping the role of initial and accepting states. We note that this step may require an NFA to have more than one initial state, but NFAs are known to be sufficiently robust to allow this. Common suffixes of words from $K$ and $L$ are common prefixes of words from $\mathrm{rev}(K)$ and $\mathrm{rev}(L)$. We then compute the language of all prefixes of words from $\mathrm{rev}(K)$ and $\mathrm{rev}(L)$ by making all the states accepting, thereby obtaining languages $\mathrm{pref}(\mathrm{rev}(K))$ and $\mathrm{pref}(\mathrm{rev}(L))$, respectively. The intersection $I = \mathrm{pref}(\mathrm{rev}(K)) \cap \mathrm{pref}(\mathrm{rev}(L))$ is the set of all words $v \in \Sigma^*$ such that there are words $w \in K$ and $w' \in L$ with $v \preceq_s w$ and $v \preceq_s w'$. To check the condition it is sufficient to test whether the language $I$ is infinite, which can also be done in polynomial time. This leads to the following lemma.

**Lemma 31.** *The separation problem by $\mathcal{F}(\preceq_s, \mathit{union})$ is in polynomial time.*

**Lemma 32.** *A language $K$ is separable from a language $L$ by $\mathcal{F}(\preceq_s, bc)$ if and only if there exists a natural number $k \geq 0$ such that no words $w \in K$ and $w' \in L$ have a common suffix of length $k$.*

*Proof.* Assume that $K$ is separable from $L$ by a finite boolean combination of languages $\Sigma^* w_1, \ldots, \Sigma^* w_n$. Let $k = \max(|w_1|, \ldots, |w_n|)+1$. Note that, for all words $wv$ and $w'v$ with $|v| \geq k$ and all $1 \leq i \leq n$, it holds that $wv \in \Sigma^* w_i$ if and only if $w'v \in \Sigma^* w_i$. Thus, any words with a common suffix of length at least $k$ cannot be separated by the considered set of languages, which means that there are no words $w \in K$ and $w' \in L$ with a common suffix of length $k$.

To show the opposite implication, assume that there exists a natural number $k$ satisfying the condition. Then, for every $w \in K$, if $|w| < k$, we can cover word $w$ by the language $\{w\} = \Sigma^* w \setminus \bigcup_{a \in \Sigma} \Sigma^* aw$. If $|w| \geq k$, then $w \in \Sigma^* v$, where $v$ is a suffix of $w$ of length $k$. By the assumption that no words of $K$ and $L$ have a common suffix of length $k$ we have that $\Sigma^* v \cap L = \emptyset$, which completes the proof. □

It therefore follows that separability by $\mathcal{F}(\preceq_s, \mathrm{bc})$ can be done with a simplified version of the procedure for $\mathcal{F}(\preceq_s, \mathrm{unions})$. Since the latter was already in polynomial time according to Lemma 31, we have the following lemma.

**Lemma 33.** *The separation problem by $\mathcal{F}(\preceq_s, bc)$ is in polynomial time.*