Separating regular languages by piecewise testable and unambiguous languages*

Thomas Place, Lorijn van Rooijen and Marc Zeitoun

LaBRI, Univ. Bordeaux & CNRS UMR 5800.
351 cours de la Libération, 33405 Talence Cedex, France.
tplace@labri.fr, lvanrooi@labri.fr, mz@labri.fr

Abstract. Separation is a classical problem asking whether, given two sets belonging to some class, it is possible to separate them by a set from another class. We discuss the separation problem for regular languages. We give a PTIME algorithm to check whether two given regular languages are separable by a piecewise testable language, that is, whether a $\mathcal{B}\Sigma_1(<)$ sentence can witness that the languages are disjoint. The proof refines an algebraic argument from Almeida and the third author. When separation is possible, we also express a separator by saturating one of the original languages by a suitable congruence. Following the same line, we show that one can as well decide whether two regular languages can be separated by an unambiguous language, albeit with a higher complexity.

1 Introduction

Separation is a classical notion in mathematics and computer science. In general, one says that two structures L_1, L_2 from a class $\mathcal{C}$ are *separable* by a structure L if $L_1 \subseteq L$ and $L_2 \cap L = \varnothing$. In this case, L is called a *separator*. In separation problems, the separator L is required to belong to a given class Sep. The problem asks whether two disjoint elements L_1, L_2 of $\mathcal{C}$ can always be separated by an element of the class Sep. In the case that disjoint elements of $\mathcal{C}$ cannot always be separated by an element of Sep, several natural questions arise:

(1) given elements L_1, L_2 in $\mathcal{C}$, can we decide whether a separator exists in Sep?
(2) if so, what is the complexity of this decision problem?
(3) can we, in addition, compute a separator, and what is the complexity?

In this context, it is known for example that separation of two context-free languages by a regular one is undecidable [9].

Separating regular languages. This paper looks at separation problems for the class $\mathcal{C}$ of regular languages, and for classes Sep closed under complement. Under this last condition, a separation algorithm for Sep entails an algorithm for deciding membership in Sep, *i.e.*, membership reduces to separability. Indeed, membership in Sep can be checked by testing whether the input language is Sep-separable from its complement.

* Supported by the Agence Nationale de la Recherche ANR 2010 BLAN 0202 01 FREC.

Conversely, while finding a decidable characterization for Sep already requires a deep understanding of the subclass, the search for separation algorithms is intrinsically more difficult. Indeed, powerful tools are available to decide membership in Sep: one normally makes use of a recognizing device of the input language, *viz.* its syntactic monoid. A famous result along these lines is Schützenberger's Theorem [14], which states that a language is definable in first-order logic if and only if its syntactic monoid is aperiodic, a property one can easily decide.

Now for a separation algorithm, the question is whether the input languages are *sufficiently different*, from the point of view of the subclass Sep, to allow this to be witnessed by an element of Sep. Note that we cannot use standard methods on the recognizing devices, as was the case for the membership problem. We now have to decide whether there *exists* a recognition device of the given type that separates the input: we do not have it in hand, nor its syntactic monoid. An even harder question then is to actually construct the so-called separator in Sep.

Contributions. In this paper, we study this problem for two subclasses of the regular languages: piecewise testable languages and unambiguous languages.

Piecewise testable languages are languages that can be described by the presence or absence of scattered subwords up to a certain size within the words. Equivalently, these are the languages definable using $\mathcal{B}\Sigma_1(<)$ formulas, *i.e.*, first-order logic formulas that are boolean combinations of $\Sigma_1(<)$ formulas. A $\Sigma_1(<)$ formula is a first-order formula with a quantifier prefix $\exists^*$, followed by a quantifier-free formula. A well-known result about piecewise testable languages is Simon's Theorem [16], which states that a regular language is piecewise testable if and only if its syntactic monoid is $\mathcal{J}$-trivial. This property yields a decision procedure to check whether a language is piecewise testable, refined by Stern into a PTIME algorithm [18], of which the complexity has been improved by Trahtman [21].

The second class that we consider is the class of unambiguous languages, *i.e.*, languages defined by unambiguous products. This class has been given many equivalent characterizations [19]. For example, these are the $\mathrm{FO}^2(<)$-definable languages, that is, languages that can be defined in first-order logic using only two variables. Equivalently, this is the class $\Delta_2(<)$ of languages that are definable by a first-order formula with a quantifier prefix $\exists^*\forall^*$ and simultaneously by a first-order formula with a quantifier prefix $\forall^*\exists^*$. Note that consequently, all piecewise testable languages are $\mathrm{FO}^2(<)$-definable. It has been shown in [10] for $\Delta_2(<)$, and in [20] for $\mathrm{FO}^2(<)$ that these are exactly the languages whose syntactic monoid belongs to the decidable class DA.

There is a common difficulty in the separation problems for these two classes. *A priori*, it is not known up to which level one should proceed in refining the candidate separators to be able to answer the question of separability. For piecewise testable languages, this refinement basically means increasing the size of the considered subwords. For unambiguous languages, it means increasing the size of the unambiguous products. For both of these classes, we are able to compute, from the two input languages, a number that suffices for this purpose. This entails decidability of the separability problem for both classes.

In both cases, we obtain a better complexity bound to answer the decision problem starting from NFAs: we show that two languages are separable if and only if the corresponding automata contain certain forbidden patterns of the same type. We prove that for piecewise testable languages this property can be decided in polynomial time wrt. the size of the automata and of the alphabet. For unambiguous languages this can be done in exponential space.

Related work. The classes of piecewise testable and unambiguous languages are varieties of regular languages. For such varieties, there is a generic connection found by Almeida [1] between profinite semigroup theory and the separation problem: Almeida has shown that two regular languages over A are separable by a language of a variety $A^*\mathcal{V}$ if and only if the topological closures of these two languages inside a profinite semigroup, depending only on $A^*\mathcal{V}$, intersect. Note that this theory does not give any information about how to actually *construct* the separator, in case two languages are separable. To turn Almeida's result into an algorithm deciding separability, we should compute representations of these topological closures, and test for emptiness of intersections of such closures.

So far, these problems have no generic answer and have been studied in an algebraic context for a small number of specific varieties. Deciding whether the closures of two regular languages intersect is equivalent to computing the so-called 2-pointlike sets of a finite semigroup wrt. the considered variety, see [1]. This question has been answered positively for the varieties of finite group languages [4,12], piecewise testable languages [3,2], star-free languages [8,7], and a few other varieties, but it was left open for unambiguous languages.

A general issue is that the topological closures may not be describable by a finite device. However, for piecewise testable languages, the approach of [3] builds an automaton over an extended alphabet, of exponential size wrt. the original alphabet, recognizing the closure of a regular language. The algorithm is polynomial wrt. the size of the original automaton (the construction was presented for deterministic automata but also works for nondeterministic ones). These automata admit the usual construction for intersection and can be checked for emptiness in NLOGSPACE. This yields an algorithm which, from two NFAs, decides separability by a piecewise testable language in time polynomial in the number of states of the NFAs, and exponential in the size of the original alphabet.

Our proof for separability by piecewise testable languages follows the same pattern as the method described above. A significant improvement is that we show that non-separability is witnessed by paths of the same shape in both automata, which yields an algorithm providing better complexity: it runs in polynomial time in both the size of the automata *and* in the size of the alphabet. Also, we do not make use of the theory of profinite semigroups: we work only with elementary concepts. We have described this algorithm in [13]. Furthermore, we show how to compute from the input languages an index that suffices to separate them. We use the same technique for unambiguous languages. Recently, Czerwinski et. al. [6] also provided a PTIME algorithm for deciding separability by piecewise testable languages, but do not provide the computation of such an index.

Due to space constraints, some proofs only appear in the full version of this paper.

2 Preliminaries

We fix a finite alphabet $A = \{a_1, \dots, a_m\}$. We denote by A^* the free monoid over A. The empty word is denoted by ε. For a word $u \in A^*$, the smallest $B \subseteq A$ such that $u \in B^*$ is called the *alphabet* of u and is denoted by $\mathsf{alph}(u)$.

Separability. Given languages L, L_1, L_2, we say that L *separates* L_1 from L_2 if

$$L_1 \subseteq L \text{ and } L_2 \cap L = \varnothing.$$

Given a class Sep of languages, we say that the pair (L_1, L_2) is Sep-*separable* if some language $L \in \mathsf{Sep}$ separates L_1 from L_2. Since all classes we consider are closed under complement, (L_1, L_2) is Sep-separable if and only if (L_2, L_1) is, in which case we simply say that L_1 and L_2 are Sep-separable.

We are interested in two classes Sep of separators: the class of piecewise testable languages, and the class of unambiguous languages.

Piecewise Testable Languages. We say that a word u is a *piece* of v, if

$$u = b_1 \cdots b_k, \text{ where } b_1, \dots, b_k \in A, \quad \text{and } v \in A^* b_1 A^* \cdots A^* b_k A^*.$$

For instance, ab is a piece of $bb\underline{a}cc\underline{b}a$. The *size* of a piece is its number of letters. A language $L \subseteq A^*$ is *piecewise testable* if there exists $\kappa \in \mathbb{N}$ such that membership of w in L only depends on the pieces of size up to κ occurring in w. We write $w \sim_\kappa w'$ when w and w' have the same pieces of size up to κ. Clearly, $\sim_\kappa$ is a congruence of finite index. Therefore, a language is piecewise testable if and only if it is a union of $\sim_\kappa$-classes for some $\kappa \in \mathbb{N}$. In this case, the language is said to be of *index* κ. It is easy to see that a language is piecewise testable if and only if it is a finite boolean combination of languages of the form $A^* b_1 A^* \cdots A^* b_k A^*$.

Piecewise testable languages are languages definable by $\mathcal{B}\Sigma_1(<)$ formulas, that is, boolean combinations of first-order formulas of the form:

$$\exists x_1 \dots \exists x_n \ \varphi(x_1, \dots, x_n),$$

where φ is quantifier-free. For instance, $A^* b_1 A^* \cdots A^* b_k A^*$ is defined by the formula $\exists x_1 \dots \exists x_k \big[\bigwedge_{i<k}(x_i < x_{i+1}) \wedge \bigwedge_{i \leqslant k} b_i(x_i)\big]$, where the first-order variables $x_1, \dots, x_k$ are interpreted as positions, and where $b(x)$ is the predicate testing that position x carries letter b.

We denote by $PT[\kappa]$ the class of all piecewise testable languages of index κ or less, and by $PT = \bigcup_\kappa PT[\kappa]$ the class of all piecewise testable languages. Given $L \subseteq A^*$ and $\kappa \in \mathbb{N}$, the smallest $PT[\kappa]$-language containing L is

$$[L]_{\sim_\kappa} = \{w \in A^* \mid \exists u \in L \text{ and } u \sim_\kappa w\}.$$

In general however, there is no smallest PT-language containing a given language.

Unambiguous Languages. A product $L = B_0^* a_1 B_1^* \cdots B_{k-1}^* a_k B_k^*$ is called *unambiguous* if every word of L admits exactly one factorization witnessing its membership in L. The number k is called the *size* of the product. An *unambiguous*

language is a finite disjoint union of unambiguous products. Observe that unambiguous languages are connected to piecewise testable languages. Indeed, it was proved in [15] that the class of unambiguous languages is closed under boolean operations. Moreover, languages of the form $A^*b_1A^* \cdots A^*b_kA^*$ are unambiguous, witnessed by the product $(A \setminus \{b_1\})^*b_1(A \setminus \{b_2\})^* \cdots (A \setminus \{b_k\})^*b_kA^*$. Therefore, piecewise testable languages form a subclass of the unambiguous languages.

Many equivalent characterizations for unambiguous languages have been found [19]. From a logical point of view, unambiguous languages are exactly the languages definable by an $\mathrm{FO}^2(<)$ formula [20]. Here, $\mathrm{FO}^2(<)$ denotes the two-variable restriction of first-order logic. Another logical characterization which further illustrates the link with piecewise testable languages (*i.e.*, $\mathcal{B}\Sigma_1(<)$-definable languages) is $\Delta_2(<)$. A $\Sigma_2(<)$ formula is a first-order formula of the form:

$$\exists x_1 \ldots \exists x_n \ \forall y_1 \ldots \forall y_m \ \varphi(x_1, \ldots, x_n, y_1, \ldots, y_m),$$

where φ is quantifier-free. A language is $\Delta_2(<)$-*definable* if it can be defined both by a $\Sigma_2(<)$ formula and the negation of a $\Sigma_2(<)$ formula. It has been proven in [10] that a language is unambiguous if and only if it is $\Delta_2(<)$-definable.

For two words w, w', we write, $w \cong_\kappa w'$ if w, w' belong to the same unambiguous products of size κ or less. We denote by $UL[\kappa]$ the class of all languages that are unions of $\cong_\kappa$-classes, and we let $UL = \bigcup_\kappa UL[\kappa]$. Since unambiguous languages are closed under boolean operations, UL is the class of all unambiguous languages. Given $L \subseteq A^*$ and $\kappa \in \mathbb{N}$, the smallest $UL[\kappa]$-language containing L is

$$[L]_{\cong\kappa} = \{w \in A^* \mid \exists u \in L \text{ and } u \cong_\kappa w\}.$$

Again, in general there is no smallest UL-language containing a given language.

Automata. A *nondeterministic finite automaton* (NFA) over A is denoted by a tuple $\mathcal{A} = (Q, A, I, F, \delta)$, where Q is the set of states, $I \subseteq Q$ the set of initial states, $F \subseteq Q$ the set of final states and $\delta \subseteq Q \times A \times Q$ the transition relation. The *size* of an automaton is its number of states plus its number of transitions. We denote by $L(\mathcal{A})$ the language of words accepted by $\mathcal{A}$. Given a word $u \in A^*$, a subset B of A and two states p, q of $\mathcal{A}$, we denote

- by $p \xrightarrow{u} q$ a path from state p to state q labeled u,
- by $p \xrightarrow{\subseteq B} q$ a path from p to q of which all transitions are labeled over B,
- by $p \xrightarrow{=B} q$ a path from p to q of which all transitions are labeled over B, with the additional demand that every letter of B occurs at least once along it.

Given a state p, we denote by $\mathsf{scc}(p, \mathcal{A})$ the strongly connected component of p in $\mathcal{A}$ (that is, the set of states that are reachable from p and from which p can be reached), and by $\mathsf{alph\_scc}(p, \mathcal{A})$ the set of labels of all transitions occurring in this strongly connected component. Finally, we define the restriction of $\mathcal{A}$ to a subalphabet $B \subseteq A$ by $\mathcal{A} \restriction_B \stackrel{\text{def}}{=} (Q, B, I, F, \delta \cap (Q \times B \times Q))$.

3 Separation by piecewise testable languages

Since $PT[\kappa] \subset PT$, $PT[\kappa]$-separability implies PT-separability. Furthermore, for a fixed κ, it is obviously decidable whether two languages L_1 and L_2 are $PT[\kappa]$-

separable: there is a finite number of $PT[\kappa]$ languages over A, and for each of them, one can test whether it separates L_1 and L_2. The difficulty for deciding whether L_1 and L_2 are PT-separable is to effectively compute a witness $\kappa = \kappa(L_1, L_2)$, *i.e.*, such that L_1 and L_2 are PT-separable if and only if they are $PT[\kappa]$-separable. Actually, we show that PT-separability is decidable, by different arguments:

(1.a) We give a necessary and sufficient condition on NFAs recognizing L_1 and L_2, in terms of forbidden patterns, to test whether L_1 and L_2 are PT-separable.

(1.b) We give a polynomial time algorithm to check this condition.

(2) We compute $\kappa \in \mathbb{N}$ from L_1, L_2, such that PT-separability and $PT[\kappa]$-separability are equivalent for L_1 and L_2. Hence, if the PTIME algorithm answers that L_1 and L_2 are PT-separable, then $[L_1]_{\sim\kappa}$ is a valid PT-separator.

Let us first introduce some terminology to explain the necessary and sufficient condition on NFAs. Let $\mathcal{A}$ be an NFA over A. For $u_0, \ldots, u_p \in A^*$ and nonempty subalphabets $B_1, \ldots, B_p \subseteq A$, let $\boldsymbol{u} = (u_0, \ldots, u_p)$ and $\boldsymbol{B} = (B_1, \ldots, B_p)$. A $(\boldsymbol{u}, \boldsymbol{B})$-*path* in $\mathcal{A}$ is a successful path (leading from the initial state to a final state of $\mathcal{A}$), of the form shown in Fig. 1.

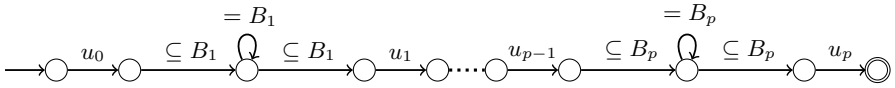

Fig. 1. A $(\boldsymbol{u}, \boldsymbol{B})$-path

Recall that edges denote sequences of transitions (see section *Automata*, p. 5). Therefore, if $\mathcal{A}$ has a $(\boldsymbol{u}, \boldsymbol{B})$-path, then $L(\mathcal{A})$ contains a language of the form $u_0(x_1 y_1^* z_1)u_1 \cdots u_{p-1}(x_p y_p^* z_p)u_p$, where $\mathsf{alph}(x_i) \cup \mathsf{alph}(z_i) \subseteq \mathsf{alph}(y_i) = B_i$.

Given NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$, a pair $(\boldsymbol{u}, \boldsymbol{B})$ is a *witness of non PT-separability* for $(\mathcal{A}_1, \mathcal{A}_2)$ if there is a $(\boldsymbol{u}, \boldsymbol{B})$-path in both $\mathcal{A}_1$ and $\mathcal{A}_2$. For instance in Fig. 2, $\boldsymbol{u} = (\varepsilon, c, \varepsilon)$ and $\boldsymbol{B} = (\{a, b\}, \{a\})$ define such a witness of non PT-separability.

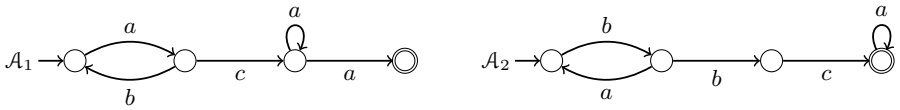

Fig. 2. A witness of non PT-separability for $(\mathcal{A}_1, \mathcal{A}_2)$: $\boldsymbol{u} = (\varepsilon, c, \varepsilon)$, $\boldsymbol{B} = (\{a, b\}, \{a\})$

We are now ready to state our main result regarding PT-separability.

Theorem 1. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two NFAs over A. Let $L_1 = L(\mathcal{A}_1)$ and $L_2 = L(\mathcal{A}_2)$. Let k_1, k_2 be the number of states of $\mathcal{A}_1$ resp. $\mathcal{A}_2$. Define $p = \max(k_1, k_2)+1$ and $\kappa = p|A|2^{2^{|A|}|A|(p|A|+1)}$. Then the following conditions are equivalent:*

(1) *L_1 and L_2 are PT-separable.*
(2) *L_1 and L_2 are $PT[\kappa]$-separable.*
(3) *The language $[L_1]_{\sim_\kappa}$ separates L_1 from L_2.*
(4) *There is no witness of non PT-separability in $(\mathcal{A}_1, \mathcal{A}_2)$.*

Condition (2) yields an algorithm to test PT-separability of regular languages. Indeed, one can effectively compute all piecewise testable languages of index κ (of which there are finitely many), and for each of them, one can test whether it separates L_1 and L_2. Before proving Theorem 1, we show that Condition (4) can be tested in polynomial time (and hence, PT-separability is PTIME decidable).

Proposition 2. *Given two NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$, one can determine whether there exists a witness of non PT-separability in $(\mathcal{A}_1, \mathcal{A}_2)$ in polynomial time wrt. the sizes of $\mathcal{A}_1$ and $\mathcal{A}_2$, and the size of the alphabet.*

Proof. Let us first show that the following problem is in PTIME: given states p_1, q_1, r_1 of $\mathcal{A}_1$ and p_2, q_2, r_2 of $\mathcal{A}_2$, determine whether there exist a nonempty $B \subseteq A$ and paths $p_i \xrightarrow{\subseteq B} q_i \xrightarrow{(=B)} q_i \xrightarrow{\subseteq B} r_i$ in $\mathcal{A}_i$ for both $i = 1, 2$.

To do so, we compute a decreasing sequence $(C_i)_i$ of alphabets overapproximating the greatest alphabet B that can be chosen for labeling the loops around q_1 and q_2. Note that if there exists such an alphabet B, it should be contained in

$$C_1 \stackrel{\text{def}}{=} \mathsf{alph\_scc}(q_1, \mathcal{A}_1) \cap \mathsf{alph\_scc}(q_2, \mathcal{A}_2).$$

Using Tarjan's algorithm to compute strongly connected components in linear time, one can compute C_1 in linear time as well. Then, we restrict the automata to alphabet C_1, and we repeat the process to obtain the sequence $(C_i)_i$:

$$C_{i+1} \stackrel{\text{def}}{=} \mathsf{alph\_scc}(q_1, \mathcal{A}_1 \restriction_{C_i}) \cap \mathsf{alph\_scc}(q_2, \mathcal{A}_2 \restriction_{C_i}).$$

After a finite number n of iterations, we obtain $C_n = C_{n+1}$. Note that $n \leqslant |\mathsf{alph}(\mathcal{A}_1) \cap \mathsf{alph}(\mathcal{A}_2)| \leqslant |A|$. If $C_n = \varnothing$, then there exists no nonempty B for which there is an $(= B)$-loop around both q_1 and q_2. If $C_n \neq \varnothing$, then it is the maximal nonempty alphabet B such that there are $(= B)$-loops around q_1 in $\mathcal{A}_1$ and q_2 in $\mathcal{A}_2$. It then remains to determine whether there exist paths $p_1 \xrightarrow{\subseteq B} q_1 \xrightarrow{\subseteq B} r_1$ and $p_2 \xrightarrow{\subseteq B} q_2 \xrightarrow{\subseteq B} r_2$, which can be performed in linear time.

To sum up, since the number n of iterations such that $C_n = C_{n+1}$ is bounded by $|A|$, and since each computation is linear wrt. the size of $\mathcal{A}_1$ and $\mathcal{A}_2$, one can decide in PTIME wrt. to both $|A|$ and these sizes whether such a pair of paths occurs.

Now we build from $\mathcal{A}_1$ and $\mathcal{A}_2$ two new automata $\tilde{\mathcal{A}}_1$ and $\tilde{\mathcal{A}}_2$ as follows. The procedure first initializes $\tilde{\mathcal{A}}_i$ as a copy of $\mathcal{A}_i$. Denote by Q_i the state set of $\mathcal{A}_i$. For each 4-uple $\tau = (p_1, r_1, p_2, r_2) \in Q_1^2 \times Q_2^2$ such that there exist $B \neq \varnothing$, two states $q_1 \in Q_1, q_2 \in Q_2$ and paths $p_i \xrightarrow{\subseteq B} q_i \xrightarrow{=B} q_i \xrightarrow{\subseteq B} r_i$ in $\mathcal{A}_i$ both for $i = 1$ and $i = 2$, we add in both $\tilde{\mathcal{A}}_1$ and $\tilde{\mathcal{A}}_2$ a new letter a_τ to the alphabet, and "summary" transitions $p_1 \xrightarrow{a_\tau} r_1$ and $p_2 \xrightarrow{a_\tau} r_2$. Since there is a polynomial number of tuples

$(p_1, q_1, r_1, p_2, q_2, r_2)$, the above shows that computing these new transitions can be performed in PTIME. So, computing $\tilde{\mathcal{A}_1}$ and $\tilde{\mathcal{A}_2}$ can be done in PTIME.

By construction, there exists some pair $(\boldsymbol{u}, \boldsymbol{B})$ such that $\mathcal{A}_1$ and $\mathcal{A}_2$ both have a $(\boldsymbol{u}, \boldsymbol{B})$-path if and only if $L(\tilde{\mathcal{A}_1}) \cap L(\tilde{\mathcal{A}_2}) \neq \varnothing$. Since both $\tilde{\mathcal{A}_1}$ and $\tilde{\mathcal{A}_2}$ can be built in PTIME, this can be decided in polynomial time as well. □

The following is an immediate consequence of Theorem 1 and Proposition 2.

Corollary 3. *Given two NFAs, one can determine in polynomial time, with respect to the number of states and the size of the alphabet, whether the languages recognized by these NFAs are PT-separable.* □

In the rest of the section, we sketch the proof of Theorem 1. The implications (3)$\iff$(2) $\implies$ (1) are obvious. To show (1) $\implies$ (2), we introduce some terminology. Let us fix an arbitrary order $a_1 < \cdots < a_m$ on A.

(p, B)-patterns. Let $B = \{b_1, \ldots, b_r\} \subseteq A$ with $b_1 < \cdots < b_r$, and let $p \in \mathbb{N}$. We say that a word $w \in A^*$ is a *(p, B)-pattern* if $w \in (B^* b_1 B^* \cdots B^* b_r B^*)^p$. The number p is called the *power* of w. For example, set $B = \{a, b, c\}$ with $a < b < c$. The word $bb\underline{\boldsymbol{a}}a\underline{\boldsymbol{b}}ab\underline{\boldsymbol{c}}c\underline{\boldsymbol{a}}c\underline{\boldsymbol{b}}aba\underline{\boldsymbol{c}}a$ is a $(2, B)$-pattern but not a $(3, B)$-pattern.

ℓ-templates. An *ℓ-template* is a sequence $T = t_1, \ldots, t_\ell$ of length ℓ, such that every t_i is either a letter or a nonempty subset of the alphabet A. The main idea behind ℓ-templates is that they yield decompositions of words that can be detected using pieces and provide a suitable decomposition for pumping. Unfortunately, not all ℓ-templates are actually detectable. Because of this we restrict ourselves to a special case of ℓ-templates. An ℓ-template is said to be *unambiguous* if all pairs t_i, t_{i+1} are either two letters, two incomparable sets or a set and a letter that is not included in the set. For example, $T = a, \{b, c\}, d, \{a\}$ is unambiguous, while $T' = \underline{\boldsymbol{b}}, \{\underline{\boldsymbol{b}}, c\}, d, \{a\}$ and $T'' = a, \{b, \underline{\boldsymbol{c}}\}, \{\underline{\boldsymbol{c}}\}, \{a\}$ are not.

p-implementations. A word $w \in A^*$ is a *p-implementation* of an ℓ-template $T = t_1, \ldots, t_\ell$ if $w = w_1 \cdots w_\ell$ and for all i either $t_i = w_i \in A$ or $t_i = B \subseteq A$, $w_i \in B^*$ and w_i is a (p, B)-pattern. For example, $abccbbcbdaaaa = a.(\underline{\boldsymbol{b}}\boldsymbol{c}c\underline{\boldsymbol{b}}b\underline{\boldsymbol{c}}b).d.(\underline{\boldsymbol{a}}\boldsymbol{a}aa)$ is a 2-implementation of the 4-template $T = a, \{b, c\}, d, \{a\}$, since $\underline{\boldsymbol{b}}\boldsymbol{c}c\underline{\boldsymbol{b}}b\underline{\boldsymbol{c}}b$ is a $(2, \{b, c\})$-pattern and $\underline{\boldsymbol{a}}\boldsymbol{a}aa$ is a $(2, \{a\})$-pattern.

We now prove (1) $\implies$ (2) by contraposition: we show that if $w_1 \in L_1, w_2 \in L_2$ are such that $w_1 \sim_\kappa w_2$, then for any h, one can build $v_1 \in L_1$ and $v_2 \in L_2$ such that $v_1 \sim_h v_2$. Therefore, non-$PT[\kappa]$-separability entails non-PT-separability.

Lemma 4. *From regular languages L_1, L_2, we can compute $p \in \mathbb{N}$ such that whenever L_1 and L_2 both contain p-implementations of the same ℓ-template T, then L_1 and L_2 are not PT-separable.*

Proof. Let p be greater than the number of states of NFAs recognizing L_1, L_2. Let w_1, w_2 be p-implementations of an ℓ-template $T = t_1, \ldots, t_\ell$. Fix $h \in \mathbb{N}$. Whenever t_i is a set B, the corresponding factors in w_1, w_2 are (p, B)-patterns. By choice of p, these factors can be pumped into (h, B)-patterns in $v_1 \in L_1$ and $v_2 \in L_2$, respectively. It is then easy to check that $v_1 \sim_h v_2$. Hence, L_1 and L_2 are not $PT[h]$-separable. Since h is arbitrary, L_1, L_2 are not PT-separable. □

It remains to prove that if two words contain the same pieces of a large enough size κ, they are both p-implementations of a common unambiguous ℓ-template, where p is the number introduced in Lemma 4. We split the proof in two parts. We begin by proving that it is enough to look for ℓ-templates for a bounded ℓ.

Lemma 5. *Let $p \in \mathbb{N}$. Every word is the p-implementation of some unambiguous N_A-template, for $N_A = 2^{2^{|A|}|A|(p|A|+1)}$.*

Proof. We first get rid of the unambiguity condition. Any ambiguous ℓ-template T can be reduced to an unambiguous ℓ'-template T' with $\ell' < \ell$ by merging the ambiguities. It is then straightforward to reduce any p-implementation of T into a p-implementation of T'. Therefore, it suffices to prove that every word is the p-implementation of some (possibly ambiguous) N_A-template.

The choice of N_A comes from Erdös-Szekeres' upper bound of Ramsey numbers. Indeed, a complete graph with edges labeled over $c = 2^{|A|}$ colors, there exists a complete monochromatic subgraph of size $m = p|A| + 1$ provided the graph has at least 2^{mc} vertices (see [5] for a short proof that this bound suffices).

Observe that a word is always the p-implementation of the ℓ-template which is just the sequence of its letters. Therefore, in order to complete our proof, it suffices to prove that if a word is the p-implementation of some ℓ-template T with $\ell > N_A$, then it is also the p-implementation of an ℓ'-template with $\ell' < \ell$.

Fix a word w, and assume that w is the p-implementation of some ℓ-template $T = t_1, \dots, t_\ell$ with $\ell > N_A$. By definition, we get a decomposition $w = w_1 \cdots w_\ell$. We construct a complete graph Γ with vertices $\{0, \dots, \ell\}$ and edges labeled by subsets of A. For all $i < j$, we set $\mathsf{alph}(w_{i+1} \cdots w_j)$ as the label of the edge (i, j). Since Γ has more than $\ell > N_A$ vertices, by definition of N_A there exists a complete monochromatic subgraph with $p|A| + 1$ vertices $\{i_1, \dots, i_{p|A|+1}\}$. Let B be the color of the edges of this monochromatic subgraph. Let $w' = w_{i_1+1} \cdots w_{i_{p|A|+1}}$. By construction, w' is the concatenation of $p|A| \geqslant p$ words with alphabet exactly B. Hence w' is a (p, B)-pattern. It follows that w is a p-implementation of the ℓ'-template $t_1, \dots, t_{i_1}, B, t_{i_{p|A|+2}}, \dots, t_\ell$ with $\ell' = \ell - p|A| + 1$. Hence $\ell' < \ell$ (except for the trivial case $p = |A| = 1$). □

The next lemma proves that once ℓ and p are fixed, given w it is possible to describe by pieces ℓ-templates that w p-implements, as long as they are unambiguous.

Lemma 6. *Let $\ell, p \in \mathbb{N}$. From p and ℓ, we can compute κ such that for every pair of words $w \sim_\kappa w'$ and every unambiguous ℓ-template T, w' is a p-implementation of T whenever w is a $(p+1)$-implementation of T.* □

We finish the proof of the implication (1) $\Longrightarrow$ (2) by assembling the results. Let p be greater than the number of states of NFAs recognizing L_1 and L_2, as introduced in the proof of Lemma 4. Let N_A be as introduced in Lemma 5 for $p + 1$, and let $\kappa = |A|(p + 1)N_A$ be as introduced in Lemma 6. Fix $h > \kappa$ and assume that we have $w_1 \in L_1$ and $w_2 \in L_2$ such that $w_1 \sim_\kappa w_2$. By Lemma 5, w_1 is the $(p + 1)$-implementation of some unambiguous N_A-template T. Moreover,

it follows from Lemma 6 that w_2 is a p-implementation of T. By Lemma 4, we finally obtain that L_1 and L_2 are not PT-separable.

The implication (1) $\Longrightarrow$ (4) of Theorem 1 is easy to show by contraposition, see [13, Lemma 2]. The remaining implication (4) $\Longrightarrow$ (1) can be shown using Lemma 6. For a direct proof, see [13, Lemma 3], where the key for getting a forbidden pattern out of two non-separable languages is to extract a suitable p-implementation using Simon's Factorization Forest Theorem [17].

4 Separation by unambiguous languages

This section is devoted to proving that UL-separability is a decidable property. Again, the result is twofold. Using an argument that is analogous to property (2) of Theorem 1 in Section 3, we prove that given L_1, L_2, it is possible to compute a number κ such that L_1, L_2 are UL-separable if and only if they are $UL[\kappa]$-separable. It is then possible to test separability by using a brute-force approach that tests all languages in $UL[\kappa]$.

The second part of our theorem is an algorithm providing only a 'yes/no' answer, but running in exponential space. This algorithm is more complicated than the one of Section 3. In this case, we cannot search for a witness of non-separability directly on the NFAs of the languages. A precomputation is needed. We present the algorithm before stating our main theorem.

***UL*-intersection.** Let $\mathcal{A}_1 = (Q_1, A, I_1, F_1, \delta_1)$, $\mathcal{A}_2 = (Q_2, A, I_2, F_2, \delta_2)$ be NFAs. The purpose of our precomputation is to associate to all 4-uples $(q_1, r_1, q_2, r_2) \in Q_1^2 \times Q_2^2$ a set $\alpha(q_1, r_1, q_2, r_2)$ of subalphabets. Intuitively, $B \in \alpha(q_1, r_1, q_2, r_2)$ if, for all $\kappa \in \mathbb{N}$, there are two words w_1, w_2 such that

(1) $B = \mathsf{alph}(w_1) = \mathsf{alph}(w_2)$,
(2) $q_1 \xrightarrow{w_1} r_1$, and $q_2 \xrightarrow{w_2} r_2$,
(3) $w_1 \cong_\kappa w_2$.

The precomputation of $\alpha : Q_1^2 \times Q_2^2 \to 2^{2^A}$ is performed via a fixpoint algorithm. For all $(q_1, r_1, q_2, r_2) \in Q_1^2 \times Q_2^2$, we initially set $\alpha(q_1, r_1, q_2, r_2) = \{\{a\} \mid q_1 \xrightarrow{a} r_1 \text{ and } q_2 \xrightarrow{a} r_2\}$. The sets are then saturated with the following two operations:

(1) When $\alpha(p_1, q_1, p_2, q_2) = B$ and $\alpha(q_1, r_1, q_2, r_2) = C$, then add $B \cup C$ to $\alpha(p_1, r_1, p_2, r_2)$.
(2) When $B \in \alpha(q_1, q_1, q_2, q_2) \cap \alpha(r_1, r_1, r_2, r_2)$ and there exist words $w_1, w_2 \in B^*$ such that $q_1 \xrightarrow{w_1} r_1$ and $q_2 \xrightarrow{w_2} r_2$ then add B to $\alpha(q_1, r_1, q_2, r_2)$.

Since every set $\alpha(q_1, r_1, q_2, r_2)$ only grows with respect to inclusion, and is bounded from above by 2^A, the computation terminates. It is straightforward to see that α can be computed in EXPSPACE using a fixpoint algorithm. Finally, we say that L_1, L_2 have *empty UL-intersection* if $\alpha(q_1, r_1, q_2, r_2) = \varnothing$ for all $q_1, q_2 \in I_1, I_2$ and $r_1, r_2 \in F_1, F_2$. We now state the main theorem of this section.

Theorem 7. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two NFAs over alphabet A. Let $L_1 = L(\mathcal{A}_1)$ and $L_2 = L(\mathcal{A}_2)$. Let k_1, k_2 be the number of states of $\mathcal{A}_1$, resp. $\mathcal{A}_2$. Define $\kappa = (2k_1k_2 + 1)(|A| + 1)^2$. Then the following conditions are equivalent:*

(1) L_1 *and* L_2 *are UL-separable.*
(2) L_1 *and* L_2 *are* $UL[\kappa]$*-separable.*
(3) *The language* $[L_1]_{\cong_\kappa}$ *separates* L_1 *from* L_2*.*
(4) L_1, L_2 *have empty UL-intersection.*

As in the previous section, Conditions (2) and (4) yield algorithms for testing whether two languages are separable. Moreover, it can be shown that empty *UL*-intersection can be tested in PSPACE from α. Therefore, we get the following corollary.

Corollary 8. *It is decidable whether two regular languages can be separated by an unambiguous language. Moreover, this can be done in* EXPSPACE *in the size of the NFAs recognizing the languages.*

Observe that by definition of $UL[\kappa]$, the bound κ is defined in terms of unambiguous products. A rephrasing of the theorem would be: there exists a separator iff there exists one defined by a boolean combination of unambiguous products of size κ. It turns out that the same κ also works for $\mathrm{FO}^2(<)$, *i.e.*, there exists a separator iff there exists one defined by an $\mathrm{FO}^2(<)$-formula of quantifier rank κ. This can be proved by minor adjustements to the proof of Theorem 7.

The proof of Theorem 7 is inspired from techniques used in [11] and relies heavily on the notion of (p, B)-patterns. It works by induction on the size of the alphabet. There are two non-trivial implications: (1) $\Longrightarrow$ (4) and (4) $\Longrightarrow$ (3). We now provide an insight into the most difficult one, *i.e.*, (4) $\Longrightarrow$ (3). The following proposition is used to prove this.

Proposition 9. *Let* $B \subseteq A$ *and* $\kappa = (2k_1k_2+1)(|B|+1)^2$*. For all pairs of words* $w_1 \cong_\kappa w_2$ *such that* $B = \mathsf{alph}(w_1) = \mathsf{alph}(w_2)$ *and all pairs of states* $(q_1, r_1) \in Q_1^2$ *and* $(q_2, r_2) \in Q_2^2$ *such that* $q_1 \xrightarrow{w_1} r_1$ *and* $q_2 \xrightarrow{w_2} r_2$*, we have* $B \in \alpha(q_1, r_1, q_2, r_2)$*.*

Observe that a consequence of Proposition 9 is that as soon as there exists $w_1 \in L_1$,$w_2 \in L_2$ such that $w_1 \cong_\kappa w_2$ (*i.e.*, $[L_1]_{\cong_\kappa}$ is not a separator), there exists a witness of nonempty *UL*-intersection. This is the contrapositive of (4) $\Longrightarrow$ (3).

5 Conclusion

We proved separation results for both piecewise testable and unambiguous languages. Both results provide a means to decide separability. In the *PT* case, we even prove that this can be done in PTIME. Moreover, in both cases we give an insight on the actual separator by providing a bound on its size, should it exist.

There remain several interesting questions in this field. First, one could consider other subclasses of regular languages, the most interesting one being full first-order logic. Separability by first-order logic has already been proven to be decidable using semigroup theory [7]. However, this approach is difficult to understand, and it yields a costly algorithm that only provides a yes/no answer, without insight about a possible separator. Another question is to get tight

complexity bounds. For unambiguous languages for instance, it is likely that our EXPSPACE upper bound can be improved, and even for piecewise testable languages, we do not know any tight bounds.

A final observation is that right now, we have no general approach and are bound to use *ad-hoc* techniques for each subclass. An interesting direction would be to invent a general framework that is suitable for this problem in the same way that monoids are a suitable framework for decidable characterizations.

References

1. J. Almeida. Some algorithmic problems for pseudovarieties. *Publ. Math. Debrecen*, 54(suppl.):531–552, 1999. Automata and formal languages, VIII (Salgótarján, 1996).
2. J. Almeida, J. C. Costa, and M. Zeitoun. Pointlike sets with respect to R and J. *J. Pure Appl. Algebra*, 212(3):486–499, 2008.
3. J. Almeida and M. Zeitoun. The pseudovariety **J** is hyperdecidable. *RAIRO Inform. Théor. Appl.*, 31(5):457–482, 1997.
4. C. J. Ash. Inevitable graphs: a proof of the type II conjecture and some related decision procedures. *Internat. J. Algebra Comput.*, 1:127–146, 1991.
5. R. Bacher. An easy upper bound for Ramsey numbers. HAL, 00763927.
6. W. Czerwinski, W. Martens, and T. Masopust. Efficient separability of regular languages by subsequences and suffixes. In *Proc. of ICALP'13*, 2013.
7. K. Henckell. Pointlike sets: the finest aperiodic cover of a finite semigroup. *J. Pure Appl. Algebra*, 55(1-2):85–126, 1988.
8. K. Henckell, J. Rhodes, and B. Steinberg. Aperiodic pointlikes and beyond. *IJAC*, 20(2):287–305, 2010.
9. H. B. Hunt, III. Decidability of grammar problems. *J. ACM*, 29(2):429–447, 1982.
10. J.-E. Pin and P. Weil. Polynomial closure and unambiguous product. *Theory of Computing Systems*, 30(4):383–422, 1997.
11. T. Place and L. Segoufin. Deciding definability in $\mathrm{FO}_2(<_h, <_v)$ on trees. Journal version, to appear, 2013.
12. L. Ribes and P. A. Zalesskiĭ. On the profinite topology on a free group. *Bull. London Math. Soc.*, 25:37–43, 1993.
13. L. van Rooijen and M. Zeitoun. The separation problem for regular languages by piecewise testable languages. http://arxiv.org/abs/1303.2143, 2013.
14. M. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.
15. M. Schützenberger. Sur le produit de concaténation non ambigu. *Semigroup Forum*, 13:47–75, 1976.
16. I. Simon. Piecewise testable events. In *Proc. of the 2nd GI Conf. on Automata Theory and Formal Languages*, pages 214–222. Springer, 1975.
17. I. Simon. Factorization forests of finite height. *Th.. Comp. Sci.*, 72(1):65 – 94, 1990.
18. J. Stern. Complexity of some problems from the theory of automata. *Information and Control*, 66(3):163–176, 1985.
19. P. Tesson and D. Therien. Diamonds are forever: The variety DA. In *Semigroups, Algorithms, Automata and Languages*, pages 475–500. World Scientific, 2002.
20. D. Thérien and T. Wilke. Over words, two variables are as powerful as one quantifier alternation. In *Proc. of STOC'98*, pages 234–240. ACM, 1998.
21. A. N. Trahtman. Piecewise and local threshold testability of DFA. In *Proc. FCT'01*, pages 347–358. Springer, 2001.