Separating Regular Languages by Piecewise Testable and Unambiguous Languages

Thomas Place, Lorijn van Rooijen, Marc Zeitoun

Separating regular languages by piecewise testable and unambiguous languages

Thomas Place, Lorijn van Rooijen and Marc Zeitoun

LaBRI, Universités de 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 a smaller 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 set 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 subclass Sep of $\mathcal{C}$. The problem asks whether two disjoint elements L_1, L_2 of $\mathcal{C}$ can always be separated by an element of the subclass 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 [11].

Separating regular languages. In this paper, we look at separation problems for the class $\mathcal{C}$ of regular languages. These separation problems generalize the task of finding decidable characterizations for subclasses Sep of $\mathcal{C}$ which are closed under complement: a separation algorithm for a subclass 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.

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 already 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 [15], which states that a language is definable in first-order logic if and only if its syntactic monoid is aperiodic, which is easily decidable.

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, that is, 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 $\exists^*$. A well-known result about piecewise testable languages is Simon's Theorem [17] that 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. Stern has refined this procedure into a polynomial time algorithm [19], of which the complexity has been improved by Trahtman [22].

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 [20]. For example, these are the $\mathrm{FO}^2(<)$-definable languages, *i.e.*, 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 [12] for $\Delta_2(<)$, and in [21] 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 this 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.

A rough analysis yields a 3-NEXPTIME upper bound for separation by unambiguous languages. For separability by piecewise testable languages, we obtain a better bound starting from NFAs: we show that two languages are separable if and only if the corresponding automata do not contain certain forbidden patterns of the same type, and we prove that the presence of such patterns can be decided in polynomial time wrt. the size of the automata and of the alphabet. This yields a PTIME algorithm for deciding separation by a piecewise testable language.

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 [4,14], piecewise testable languages [3,2], star-free languages [10,9], 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, which recognizes the closure of the original language. This can be performed in polynomial time wrt. the size of the original automaton. Since these automata admit the usual construction for intersection and can be checked for emptiness in NLOGSPACE, this gives a polynomial time algorithm wrt. the size of the original automata. The construction was presented for deterministic automata but also works for nondeterministic ones. One should mention that the extended alphabet is 2^A (where A is the original alphabet). Therefore, these results give an algorithm which, from two NFAs, decides separability by piecewise testable languages 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, but a significant improvement is that we show that non-separability is witnessed by both automata admitting a path of the same shape. This allows us to present an algorithm that provides better complexity as 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 [23]. Furthermore, we show how to compute from the input languages, an index that suffices to separate them. Recently, Martens et. al. [6] also provided a PTIME algorithm for deciding separability by piecewise testable languages, using different proofs but do not provide the computation of such an index.

Finally, for separation by unambiguous languages, the positive decidability result of this paper is new, up to the authors' knowledge. It is equivalent to the decidability of computing the 2-pointlike sets for the class DA.

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 $bb\underline{\boldsymbol{a}}cc\underline{\boldsymbol{b}}a$. The *size* of a piece is its number of letters. A language $L \subseteq A^*$ is *piecewise testable* if there exists a $\kappa \in \mathbb{N}$ 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 an equivalence relation, and it has finite index (since there are finitely many pieces of size κ or less). 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 those definable by $\mathcal{B}\Sigma_1(<)$ formulas. $\mathcal{B}\Sigma_1(<)$ formulas are 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, because removing one word from a PT-language yields again a PT-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 integer 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 [16] that the class of unambiguous languages is closed under boolean operations. Moreover, languages of the form $A^* b_1 A^* \cdots A^* b_k A^*$ are unambiguous, witnessed by the product $(A \setminus \{b_1\})^* b_1 (A \setminus \{b_2\})^* \cdots (A \setminus \{b_k\})^* b_k A^*$. Therefore, piecewise testable languages form a subclass of the class of unambiguous ones.

Many equivalent characterizations for unambiguous languages have been found [20]. From a logical point of view, unambiguous languages are exactly the languages definable by an $\mathrm{FO}^2(<)$ formula [21]. $\mathrm{FO}^2(<)$ is 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 [12] 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' satisfy the same unambiguous products of size κ or less. We denote by $UL[\kappa]$ the class of all languages that are unions of equivalence classes of $\cong_\kappa$, and we let $UL = \bigcup_\kappa UL[\kappa]$. Observe that because 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

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

In general again, 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 $\mathcal{A} \restriction_B \stackrel{\text{def}}{=} (Q, B, I, F, \delta \cap (Q \times B \times Q))$.

Monoids. Let L be a language and M be a monoid. We say that L *is recognized by* M if there exists a monoid morphism $\alpha : A^* \to M$ together with a subset $F \subseteq M$ such that $L = \alpha^{-1}(F)$. It is well known that a language is accepted by an NFA if and only if it can be recognized by a *finite monoid*. Further, one can compute from any NFA a finite monoid recognizing its accepted language.

3 Separation by piecewise testable languages

Since $PT[\kappa] \subset PT$, $PT[\kappa]$-separability implies PT-separability. Further, 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 $\boldsymbol{B} = (B_1, \ldots, B_p)$. We call $(\boldsymbol{u}, \boldsymbol{B})$ a *factorization pair*. 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.

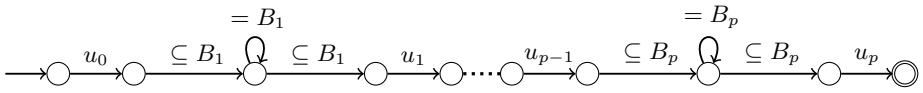

Fig. 1. A $(\boldsymbol{u}, \boldsymbol{B})$-path

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 $\mathcal{A}_2$, we say that a factorization 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, $\mathcal{A}_1$ and $\mathcal{A}_2$ pictured in Fig. 2 have a witness of non PT-separability, namely the factorization pair $(\boldsymbol{u}, \boldsymbol{B})$ with $\boldsymbol{u} = (\varepsilon, c, \varepsilon)$ and $\boldsymbol{B} = (\{a, b\}, \{a\})$.

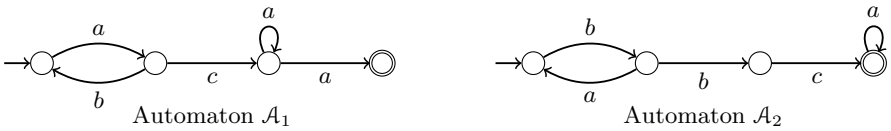

Fig. 2. A witness of non PT-separability for $(\mathcal{A}_1, \mathcal{A}_2)$: $\boldsymbol{u} = (\varepsilon, c, \varepsilon)$, $\boldsymbol{B} = (\{a, b\}, \{a\})$

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 exists 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. 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 p_1 and p_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 to get $C_n = C_{n+1}$ 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 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 $p_i \xrightarrow{\subseteq B} q_i \xrightarrow{=B} q_i \xrightarrow{\subseteq B} r_i$ 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 factorization 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)$\Longleftrightarrow$(2) $\Longrightarrow$ (1) are obvious. To show (1) $\Longrightarrow$ (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 of length ℓ, $T = t_1, \ldots, t_\ell$, such that every t_i is either a letter a or a subset B 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, $abccbbcbdaaaa = a.(\underline{\boldsymbol{bc}}c\underline{\boldsymbol{b}}b\underline{\boldsymbol{c}}b).d.(\underline{\boldsymbol{aa}}aa)$ is a 2-implementation of the 4-template $T = a, \{b, c\}, d, \{a\}$, since $\underline{\boldsymbol{bc}}c\underline{\boldsymbol{b}}b\underline{\boldsymbol{c}}b$ is a $(2, \{b, c\})$-pattern and $\underline{\boldsymbol{aa}}aa$ is a $(2, \{a\})$-pattern.

We now use p-implementations to prove (1) $\Longrightarrow$ (2). The proof is divided in two steps. First, we prove that there exists p such that if two words are p-implementations of the same ℓ-template for some ℓ, then they can be pumped into words containing the same pieces of size k for any k, while keeping membership

in the regular languages. We will then prove that if two words contain the same pieces for a large enough size, they are both p-implementations of a common unambiguous ℓ-template. We begin with the first step in the following lemma.

Lemma 4. *Let $w_1 \in L_1$ and $w_2 \in L_2$. From L_1, L_2, we can compute $p \in \mathbb{N}$ such that whenever w_1, w_2 are both p-implementations of an ℓ-template T for some ℓ, then for every $\kappa \in \mathbb{N}$, there exist $w_1' \in L_1$ and $w_2' \in L_2$ such that $w_1' \sim_\kappa w_2'$.*

Proof. This is a pumping argument. Let k_1, k_2 be the number of states of automata recognizing L_1 and L_2 and set $p = \max(k_1, k_2)$. Set w_1, w_2 and $T = t_1, \dots, t_\ell$ as in the statement of the lemma. Fix $\kappa \in \mathbb{N}$. Whenever t_i is a set B, the corresponding factors in w_1, w_2 are (p, B)-patterns. By choice of p, it follows from a pumping argument that these factors can be pumped into (κ, B)-patterns in L_1 and L_2. It is then easy to check that the resulting words have the same pieces of size κ. □

We now move to our second step. We prove that there exists a number κ such that two words having the same pieces of size up to κ must both be p-implementations of a common unambiguous ℓ-template (where p is the number introduced in Lemma 4). Again, 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, for this value of N_A, from every complete graph of size N_A with edges labeled over $2^{|A|}$ colors, there must be some complete monochromatic subgraph of size $p|A| + 1$ (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 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 $w' = w_{i_1+1} \cdots w_{i_{p|A|+1}}$, which is the concatenation of the $p|A|$ words, $w_{i_j+1} \cdots w_{i_{j+1}}$, for $j < p|A|$. By construction, these words have 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 shown in App. A proves that once ℓ and p are fixed, given w it is possible to describe by pieces the unambiguous ℓ-templates that w p-implements.

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. Assume that L_i is recognized by an NFA with k_i states. Let $p = \max(k_1, k_2)$ be as introduced in Lemma 4, N_A as introduced in Lemma 5 for $p+1$, and $\kappa = |A|(p+1)N_A$ be as introduced in Lemma 6. Fix $\kappa' > \kappa$ 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 get that there exists $w_1' \in L_1$ and $w_2' \in L_2$ such that $w_1' \sim_{\kappa'} w_2'$.

The implication (1) $\Longrightarrow$ (4) of Theorem 1 is easy and shown by contraposition, see [23, Lemma 2] and Appendix. The remaining implication (4) $\Longrightarrow$ (1) can be shown using Lemma 6 (see Appendix). For a direct proof, see [23, 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 forests [18].

4 Separation by unambiguous languages

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

In this case, we were not able to prove equivalence between *UL*-separability and the existence of some common witness inside the automata of both input languages. Because of this, we have a much higher complexity. A rough analysis of the problem, which can be found in the Appendix, gives an algorithm that runs in nondeterministic 3-exponential time in κ. It is likely that this can be improved. We now state the main theorem of this section.

Theorem 7. *Let M_1 and M_2 be monoids recognizing $L_1, L_2 \subseteq A^*$. Let $\kappa = (2|M_1||M_2|+1)(|A|+1)^2$. 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.*

As in the previous section, Condition (2) yields an algorithm for testing whether two languages are separable. Indeed, the algorithm simply computes all languages in $UL[\kappa]$ and checks for each of them whether it is a separator.

Corollary 8. *It is decidable whether two regular languages can be separated by a unambiguous language.*

Observe that in contrast to Theorem 1, the bound κ of Theorem 7 is stated in terms of monoids rather than in terms of automata which means an exponential blow-up. This is necessary for our proof technique to work.

Another remark, is 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 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 making minor adjustements to the proof of Theorem 7.

The proof of Theorem 7 is inspired from techniques used in [13] and relies heavily on the notion of (B, p)-patterns. The full proof, which works by induction on the size of the alphabet, can be found in the Appendix. We actually prove a proposition that is basically a rephrasing of Theorem 7 as a pumping-like property. In the remainder of this section, we state this proposition and explain why it implies Theorem 7.

Fix two regular languages L_1, L_2 over A, as well as their syntactic monoids M_1, M_2, and the corresponding morphisms α_1, α_2.

Proposition 9. *Let* $\kappa = (2|M_1||M_2| + 1)(|A| + 1)^2$. *For all pairs of words* $w_1 \cong_\kappa w_2$ *and all* $\kappa' > \kappa$, *there exists* $w_1' \cong_{\kappa'} w_2'$ *such that* $\alpha_1(w_1) = \alpha_1(w_1')$ *and* $\alpha_2(w_2) = \alpha_2(w_2')$.

Let us briefly explain why Proposition 9 implies Theorem 7. We explain why the direction (1) $\Longrightarrow$ (3) holds, since (3) $\Longrightarrow$ (2) and (2) $\Longrightarrow$ (1) are trivial. We prove the contrapositive: assume that $[L_1]_{\cong_\kappa}$ is not a separator. Hence, by definition of $[L_1]_{\cong_\kappa}$, there exist $w_1 \in L_1$ and $w_2 \in L_2$ such that $w_1 \cong_\kappa w_2$. Now, let $\kappa' > \kappa$. From Proposition 9, it follows that there exist $w_1' \in L_1$ and $w_2' \in L_2$ such that $w_1' \cong_{\kappa'} w_2'$. This means that L_1 and L_2 cannot be separated by a language of $UL[\kappa']$. Since this holds for all $\kappa' > \kappa$, it follows that L_1 and L_2 are not UL-separable.

5 Conclusion

We proved separation results for both piecewise testable and unambiguous languages. Both results provide a way 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 [9]. However, this approach is difficult to understand, it yields a costly algorithm, it only provides a yes/no answer, and no insight about a possible separator. Another question is to get tight complexity bounds. For unambiguous languages for instance, one can show a 3Nexptime bound using translation to automata, but in this case, and even for piecewise testable languages, we do not know any tight bounds on the size of separators.

Another 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 *ICALP*, 2013.
7. K. Etessami, M. Y. Vardi, and T. Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2), 2002.
8. P. Gastin and D. Oddoux. LTL with past and two-way very-weak alternating automata. In *MFCS*, 2003.
9. K. Henckell. Pointlike sets: the finest aperiodic cover of a finite semigroup. *J. Pure Appl. Algebra*, 55(1-2):85–126, 1988.
10. K. Henckell, J. Rhodes, and B. Steinberg. Aperiodic pointlikes and beyond. *IJAC*, 20(2):287–305, 2010.
11. H. B. Hunt, III. Decidability of grammar problems. *J. ACM*, 29(2):429–447, 1982.
12. J.-E. Pin and P. Weil. Polynomial closure and unambiguous product. *Theory of Computing Systems*, 30(4):383–422, 1997.
13. T. Place and L. Segoufin. Deciding definability in $\mathrm{FO}_2(<_h, <_v)$ on trees. Journal version, to appear, 2013.
14. L. Ribes and P. A. Zalesskiĭ. On the profinite topology on a free group. *Bull. London Math. Soc.*, 25:37–43, 1993.
15. M. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.
16. M. Schützenberger. Sur le produit de concaténation non ambigu. *Semigroup Forum*, 13:47–75, 1976.
17. I. Simon. Piecewise testable events. In *Proc. of the 2nd GI Conf. on Automata Theory and Formal Languages*, pages 214–222. Springer, 1975.
18. I. Simon. Factorization forests of finite height. *Th.. Comp. Sci.*, 72(1):65 – 94, 1990.
19. J. Stern. Complexity of some problems from the theory of automata. *Information and Control*, 66(3):163–176, 1985.
20. P. Tesson and D. Therien. Diamonds are forever: The variety DA. In *Semigroups, Algorithms, Automata and Languages*, pages 475–500. World Scientific, 2002.
21. 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.
22. A. N. Trahtman. Piecewise and local threshold testability of DFA. In *Proc. FCT'01*, pages 347–358. Springer, 2001.
23. L. van Rooijen and M. Zeitoun. The separation problem for regular languages by piecewise testable languages. http://arxiv.org/abs/1303.2143, 2013.

Appendix

A: Proofs of Section 3

Proof of Condition (4) Theorem 1

We prove Condition (4). We prove (4) $\implies$ (1) and (3) $\implies$ (4).

(4) $\implies$ (1). We proceed by contraposition. Assume that L_1, L_2 are not PT-separable. Recall that $\mathcal{A}_1$ and $\mathcal{A}_2$ are NFAs for L_1, L_2 and that k_1, k_2 are their sizes. Set $p = \max(k_1, k_2) + 1$ and $\ell = 2^{2^{|A|}|A|(p|A|+1)}$. We prove that L_1, L_2 both contain p-implementations of some N_A-template T and use T to construct a witness of non PT-separability in $(\mathcal{A}_1, \mathcal{A}_2)$.

Let κ be as defined in Lemma 6 from ℓ and p. Because L_1, L_2 are not PT-separable, there exist $w_1 \in L_1$ and $w_2 \in L_2$ such that $w_1 \sim_\kappa w_2$. By choice of ℓ, p and Lemma 5, w_1 must be the p-implementation of some unambiguous ℓ-template T. Applying Lemma 6, we obtain that w_1, w_2 are both $(\max(k_1, k_2))$-implementations of T.

Let $\boldsymbol{B} = (B_1, \ldots, B_n)$ be the subsequence of elements T that are sets. Let $\boldsymbol{u} = (u_0, \ldots, u_n)$, where u_i is the word obtained by concatenating the letters that are between B_i and B_{i+1} in T. By definition $(\boldsymbol{u}, \boldsymbol{B})$ is a factorization pair.

Because w_1 is a $(\max(k_1, k_2))$-implementation, the path used to read w_1 in $\mathcal{A}_1$ must traverse loops labeled by each of the B_i, and clearly this is a $(\boldsymbol{u}, \boldsymbol{B})$-path. Using the same argument we get that the path of w_2 in $\mathcal{A}_2$ is also a $(\boldsymbol{u}, \boldsymbol{B})$-path. Therefore $(\boldsymbol{u}, \boldsymbol{B})$ is a witness of non PT-separability.

(3) $\implies$ (4). Again, we proceed by contraposition. Set κ as in Theorem 1 and assume that there exists a factorization pair $(\boldsymbol{u}, \boldsymbol{B})$ that is a witness of non PT-separabilty. We prove that $[L_1]_{\sim\kappa}$ is not a separator.

Set $\boldsymbol{B} = (B_1, \ldots, B_n)$ and $\boldsymbol{u} = (u_0, \ldots, u_n)$. By definition, this means that there exists $w_1 \in L_1$ and $w_2 \in L_2$ of the form

$$u_0 v_i u_1 v_1 \cdots v_n u_n,$$

where the words such that $\mathsf{alph}(v_i) = B_i$ and v_i contains a $(B_i, \kappa) - pattern$. It is straightforward to see that $w_1 \sim_\kappa w_2$. Therefore, $w_2 \in L_2 \cap [L_1]_{\sim\kappa}$ and $[L_1]_{\sim\kappa}$ is not a separator.

Proof of Lemma 6

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.*

Proof. We prove that the lemma holds for $\kappa = |A|p\ell$. Let $w \sim_\kappa w'$ and let $T = \{t_1, t_2, \ldots, t_\ell\}$ be an unambiguous ℓ-template such that w is a $p+1$-implementation of T. We begin by giving a decomposition of w' and prove that it indeed witnesses the fact that w' is a p-implementation of T. We define

$w'_1 \cdots w'_\ell = w'$ inductively as follows: assume that the factors are defined up to w'_i and let u be such that $w' = w_1 \cdots w'_i \cdot u$. If t_{i+1} is a letter then w_{i+1} is just the first letter of u, otherwise $t_{i+1} = B \subseteq A$, and in that case w_{i+1} is the largest prefix of u which contains only letters of B. We will prove that $w'_1 \cdots w'_\ell$ witnesses that w' is a p-implementation of T. The proof relies on a subresult that we state and prove below.

To every unambiguous ℓ-template $T = \{t_1, t_2, \ldots, t_\ell\}$ and $p \in \mathbb{N}$, we associate a piece $v_{T,p} = v_1 \cdots v_\ell$, such that for all i:

$$v_i = \begin{cases} a & \text{if } t_i = a \in A \\ (b_1 \cdots b_n)^p & \text{if } t_i = \{b_1, \ldots, b_n\} \subseteq A \end{cases}$$

By definition, if w is a p-implementation of T then $v_{T,p}$ is a piece of w. Consider $v_{T,1} = v_1 \cdots v_\ell$. A piece v is *incompatible* with T when v is of the following form: $v = v_1 \cdots v_i \cdot u \cdot v_{i+1} \cdots v_\ell$ such that if t_i (resp. t_{i+1}) is a set the first letter (resp. last letter) of u is not in t_i (resp. t_{i+1}).

Claim 1 *If w is a 1-implementation of some unambiguous ℓ-template T, then there is no piece of w that is incompatible with T.*

Proof. This is a consequence of the fact that T is unambiguous. □

We now prove that for all i, $w'_i = a$ if t_i is the letter a or w'_i is a word over some $B \subseteq A$ containing a (p, B)-pattern if $t_i = B$. We proceed by induction, assume that this is true up to w'_i and consider w'_{i+1}. Set $v_{T,1} = v_1 \cdots v_\ell$ and $v_{T,p+1} = v_1 \cdots v_\ell$. By induction hypothesis and by choice of κ, we know that $v_{i+1} \cdots v_\ell$ and $v_{i+1} \cdots v_\ell$ are pieces of $w_{i+1} \cdots w_\ell$. We distinguish two cases depending on the nature of t_{i+1}.

Case 1: t_{i+1} is some letter a. We have to prove that $w'_{i+1} = a$. Assume that $w'_{i+1} = b \neq a$. Then $v_1 \cdots v_i \cdot b \cdot v_{i+1} \cdots v_\ell$ must be a piece of w' and therefore a piece of w ($w \sim_\kappa w'$). We prove that this piece is incompatible with T, which contradicts Claim 1. Observe that by definition of w_i, if t_i is a set then $b \notin t_i$ (otherwise it would have been included in t_i). Therefore $v_1 \cdots v_i \cdot b \cdot v_{i+1} \cdots v_\ell$ is incompatible with T and we are done for this case.

Case 2: t_{i+1} is a set $B = \{b_1, \ldots, b_n\}$. By construction, w_{i+1} contains only letters in B. Therefore, we have to prove that it contains a (p, B)-pattern. Assume that it does not. By contruction, the first letter of w_{i+2} is some letter $c \notin B$. If $w_{i+1} = \varepsilon$, then $v_1 \cdots v_i \cdot c \cdot v_{i+1} \cdots v_\ell$ must be a piece of w'. Using a similar argument to the one of the previous case, we can prove that this piece is incompatible with T, which contradicts Claim 1.

Otherwise, let b be the first letter of w_{i+1}. Recall that by definition, r_{i+1} contains a $(p+1, B)$-pattern and that $r_{i+1} \cdots r_\ell$ is a piece of $w_{i+1} \cdots w_\ell$. Therefore, since w_{i+1} does not contain a (p, B)-pattern, the last suffix of r_{i+1} containing a $(1, B)$-pattern must fall in w_{i+2} and consequently, $v_{i+1} \cdots v_\ell$ must be a piece of $w_{i+2} \cdots w_\ell$. It follows that $v_1 \cdots v_i \cdot b \cdot c \cdot v_{i+1} \cdots v_\ell$ is a piece of w' and of w ($w \sim_\kappa w'$). By definition, $c \notin t_{i+1}$. Moreover, by construction, if t_i is a set, then $b \notin t_i$. Therefore, $v_1 \cdots v_i \cdot b \cdot c \cdot v_{i+1} \cdots v_\ell$ is incompatible with T which contradicts Claim 1 since it is also a piece of w. □

B: Proofs of Section 4

Complexity Analysis

We briefly explain how *UL*-separability can be checked in 3NEXPTIME. This is a very rough analysis and it is likely that this can be improved. We use the connection between *UL* and $\mathrm{FO}^2(<)$ to prove that any language in $UL[\kappa]$ is recognized by a deterministic automaton that is 3-exponential in κ. In order to check separability it is then enough to non-deterministically guess an automaton of size 3-exponential in κ, check if the automata recognizes an unambiguous language and check if it is a separator. This can be done in non-deterministic 3-exponential time.

It is straightforward to see that any unambiguous product can be described by an $\mathrm{FO}^2(<)$ formula whose nesting depth of quantifiers is polynomial in the size κ of the product. One can then construct an equivalent unary temporal logic (**UTL**) formula that is exponential in size [7]. From this **UTL** formula, it is then possible to construct an equivalent deterministic automaton that is double exponential in size [8]. This automaton is 3-exponential in κ.

Proof of Theorem 7

We prove Proposition 9. As we explained, our proof relies heavily on the notion of (B, p)-patterns. However, in this case we will only need to use (B, p)-patterns where B is the whole alphabet A. For this reason, we simply write p-pattern for (A, p)-pattern. We begin by giving a brief outline of the proof. We fix a large enough l, intuitively l needs to be large enough so that we can apply our pumping arguments to words of length l. Then, we show that κ is large enough in order to ensure that w_1, w_2 are both a $2l$-pattern or neither of them are. In both cases, we are then able to decompose w_1, w_2 into sequences of factors that are pairwise equivalent and use a smaller alphabet. We then apply induction on these factors. In the second case, it then suffices to reconcatenate the factors to obtain the desired result. In the first case, a pumping argument depending on l will also be needed. Before providing the lemmas involved in the construction, we first give some additional definitions concerning k-patterns that will come in conveniently.

Decompositions for k-patterns. Recall that by definition, a word w is a k-pattern iff $w \in (A^*a_1 \cdots A^*a_nA^*)^k$. This means that w can be decomposed into a sequence of factors witnessing its membership in $(A^*a_1 \cdots A^*a_nA^*)^k$:

$$w = \prod_{i=1}^{kn} (w_i \cdot a_{i \bmod n}) \cdot w_{kn+1}.$$

To each word w we associate a unique such decomposition for the largest integer k such that w is a k-pattern. Let $w \in A^*$, we say that w *admits a k-decomposition* iff w is a k-pattern but not a $k+1$-pattern. It is straightforward to see that if w admits

a k-decomposition then there exists an integer l such that $kn < l < (k+1)n$, and

$$w = \prod_{i=1}^{l} (w_i \cdot a_{i \bmod m}) \cdot w_{l+1}, \tag{1}$$

such that for all i, $a_{i \bmod m} \notin w_i$. The integer l is called the *length* of the decomposition. We give an example of a word that admits a 1-decomposition (of length 5) in Figure (3).

$$\underbrace{\mathrm{b}}_{w_1}\underbrace{\mathbf{a}\mathrm{c}}_{w_2}\underbrace{\mathbf{b}\mathrm{b}}_{w_3}\underbrace{\mathbf{c}\mathrm{c}\mathrm{c}}_{w_4}\underbrace{\mathbf{a}\mathrm{c}\mathrm{c}}_{w_5}\underbrace{\mathbf{b}\mathrm{a}\mathrm{a}}_{w_6}$$

Fig. 3. $w = bcacbbcccaccbaa$ over $A = \{a, b, c\}$ admits a 1-decomposition

We now state two lemmas that are needed for our construction. Both lemmas link k-decompositions to unambiguous languages. The first one states that k-decompositions can be detected using an unambiguous product with large enough size. Moreover, using products of the same size, one can also describe the products that are satisfied by the factors in the k-decomposition.

Lemma 7. *Let $k, \tilde{\kappa} \in \mathbb{N}$ such that $\kappa > \tilde{\kappa} + k(|A| + 1)$. Then for every pair of words $u \cong_\kappa v$ and all $h \leqslant k$, u admits an h-decomposition iff v admits an h-decomposition. Moreover, the associated decompositions, as described in (1), are of the same length l:*

$$u = \textstyle\prod_{i=1}^{l} (u_i \cdot a_{i \bmod m}) \cdot u_{l+1}$$
$$v = \textstyle\prod_{i=1}^{l} (v_i \cdot a_{i \bmod m}) \cdot v_{l+1}$$

and for all i, $u_i \cong_{\tilde{\kappa}} v_i$.

Proof. For the sake of readability, for all $a \in A$ we will write A_a for the alphabet $A \setminus \{a\}$ and for all number i, we will write b_i for $a_{i \bmod m}$. Assume that u admits an h-decomposition and consider the associated decomposition:

$$u = \prod_{i=1}^{l} (u_i \cdot b_i) \cdot u_{l+1}.$$

Consider the following unambiguous product:

$$P = \prod_{i=1}^{l} (A_{b_i} \cdot b_i) \cdot A_{b_{l+1}}.$$

By construction, P is of size $l \leqslant k(|A| + 1) < \kappa$ and $u \in P$. Therefore, $v \in P$ and v admits an h-decomposition together with the associated decomposition:

$$v = \prod_{i=1}^{l} (v_i \cdot b_i) \cdot v_{l+1}.$$

It remains to prove that for all i, $u_i \cong_{\tilde{\kappa}} v_i$. Assume that u_i belongs to some unambiguous product P' of size $\tilde{\kappa}$. This means that u is in the unambiguous product:

$$P'' = \prod_{j=1}^{i-1} (A_{b_j} \cdot b_j) \cdot (P' \cdot b_i) \cdot \prod_{j=i+1}^{l} (A_{b_j} \cdot b_j) \cdot A_{b_{l+1}}.$$

Because P'' is of size at most $\tilde{\kappa} + k(|A| + 1)$, we have $v \in P''$. By assumption on the decomposition of v, this means that $v_i \in P'$ and we are done. □

Our second lemma states that if a word has both a prefix and a suffix that are large enough k-patterns, then the middle part has no influence on membership in an unambiguous product.

Lemma 8. *Let $\kappa' \in \mathbb{N}$, and let u, v be two words that are both κ'-patterns. Then for all words w_1, w_2, the following equivalence holds:*

$$u \cdot w_1 \cdot v \cong_{\kappa'} u \cdot w_2 \cdot v.$$

Proof. We begin by observing a simple property of unambiguous products.

Remark 1 *Let $B_0^* a_1 B_1^* \cdots B_{\kappa'-1}^* a_{\kappa'} B_{\kappa'}^*$ be an unambiguous product. There can be at most one set B_i such that $B_i = A$.*

Let $P = B_0^* a_1 B_1^* \cdots B_{\kappa'-1}^* a_{\kappa'} B_{\kappa'}^*$ an unambiguous product of size κ' and assume that $uw_1v \in P$. We prove that $uw_2v \in P$. Since $uw_1v \in P$, there exists some decomposition

$$uw_1v = x_0 a_1 x_1 \cdots x_{\kappa'-1} a_{\kappa'} x_{\kappa'}$$

that is a witness. If no set B_i is the whole alphabet A, then uw_1v is at most a κ'-pattern, which is impossible since it is by definition a $2\kappa'$-pattern. Therefore, by Remark 1 there is exactly one set B_i such that $B_i = A$. It follows that the words $x_0 a_1 x_1 \cdots x_{i-1}$ and $a_{i+1} \cdots x_{\kappa'-1} a_{\kappa'} x_{\kappa'}$ are at most $\kappa' - 1$-patterns. Therefore, they are respectively a prefix of u and a suffix of v (which are κ'-patterns). It follows that there exists some word y_i such that

$$uw_2v = x_0 a_1 x_1 \cdots x_{i-1} y_i a_{i+1} \cdots x_{\kappa'-1} a_{\kappa'} x_{\kappa'}.$$

Such a decomposition for uw_2v is witness for membership in P. We conclude that $uw_2v \in P$.

□

We can now finish our construction by using Lemma 7 and Lemma 8 and prove Proposition 9. The proof goes by induction on the size of the alphabet.

We begin by fixing the size of the patterns we are going to look for in w_1, w_2. Set $k = |M_1||M_2|$. A pigeonhole principle argument proves that for $s_0, \dots, s_k \in M_1$ and $r_0, \dots, r_k \in M_2$, there exist $i < j$ such that both $s_0 \cdots s_{i-1} = s_0 \cdots s_j$ and $r_0 \cdots r_{i-1} = r_0 \cdots r_j$. We will look for k-patterns.

Observe that $\kappa = (2k+1)(m+1)^2$ (recall that $m = |A|$). We prove Proposition 9 by induction on m. Let $\kappa' > \kappa$, $w_1 \cong_\kappa w_2$ and set $\tilde{\kappa} = (2k+1)m^2$. By Lemma 7, either both w_1, w_2 admit h-decompositions for some $h < 2k$ or both w_1, w_2 do not admit h-decomposition for $h \leqslant 2l$. We treat these cases separately.

Case 1: both w_1 and w_2 admit h-decompositions for $h < 2k$. Observe that $\tilde{\kappa} = (2k+1)m^2$ implies that $\kappa > \tilde{\kappa} + 2k(m+1)$. Therefore, we can apply the second part of Lemma 7 to w_1 and w_2. This yields the following decompositions:

$$w_1 = \prod_{i=1}^{l}(u_i \cdot a_{i \bmod m}) \cdot u_{l+1}$$
$$w_2 = \prod_{i=1}^{l}(v_i \cdot a_{i \bmod m}) \cdot v_{l+1}$$

such that for all i, $u_i \cong_{\tilde{\kappa}} v_i$. Also observe that by definition of these decompositions u_i, v_i use a strict subalphabet of A. Therefore, the induction hypothesis can be used and for all i we get words u'_i, v'_i such that $u'_i \cong_{\kappa'} v'_i$, $\alpha_1(u_i) = \alpha_1(u'_i)$ and $\alpha_2(v_i) = \alpha_2(v'_i)$. Now, consider the words:

$$w'_1 = \prod_{i=1}^{l}(u'_i \cdot a_{i \bmod m}) \cdot u'_{l+1}$$
$$w'_2 = \prod_{i=1}^{l}(v'_i \cdot a_{i \bmod m}) \cdot v'_{l+1}$$

By construction, we have $w'_1 \cong_{\kappa'} w'_2$, $\alpha_1(w_1) = \alpha_1(w'_1)$ and $\alpha_2(w_2) = \alpha_2(w'_2)$ and we are done.

Case 2: both w_1 and w_2 do not admit h-decompositions for $h \leqslant 2k$. This means that w_1, w_2 are both $2k$-patterns. A simple argument as in the proof of Lemma 7 shows that w_1, w_2 can be decomposed into three factors:

$$w_1 = u_{\mathfrak{l}} \cdot u_{\mathfrak{c}} \cdot u_{\mathfrak{r}}$$
$$w_2 = v_{\mathfrak{l}} \cdot v_{\mathfrak{c}} \cdot v_{\mathfrak{r}}$$

such that $u_{\mathfrak{l}} \cong_{\kappa-k(m+1)} v_{\mathfrak{l}}$, $u_{\mathfrak{r}} \cong_{\kappa-k(m+1)} v_{\mathfrak{r}}$, $u_{\mathfrak{c}} \cong_{\kappa-k(m+1)} v_{\mathfrak{c}}$ and $u_{\mathfrak{l}}, v_{\mathfrak{l}}, u_{\mathfrak{r}}, v_{\mathfrak{r}}$ admit k-decompositions. We prove that it is possible to construct $u'_{\mathfrak{l}}, v'_{\mathfrak{l}}, u'_{\mathfrak{r}}, v'_{\mathfrak{r}}$ such that:

(1) $u'_{\mathfrak{l}} \cong_{\kappa'} v'_{\mathfrak{l}}$ and $u'_{\mathfrak{r}} \cong_{\kappa'} v'_{\mathfrak{r}}$.
(2) $\alpha_1(u_{\mathfrak{l}}) = \alpha_1(u'_{\mathfrak{l}})$, $\alpha_1(u_{\mathfrak{r}}) = \alpha_1(u'_{\mathfrak{r}})$, $\alpha_2(v_{\mathfrak{l}}) = \alpha_2(v'_{\mathfrak{l}})$ and $\alpha_2(v_{\mathfrak{r}}) = \alpha_2(v'_{\mathfrak{r}})$.
(3) $u'_{\mathfrak{l}}, v'_{\mathfrak{l}}, u'_{\mathfrak{r}}, v'_{\mathfrak{r}}$ are all κ'-patterns.

We only give the proof for $u'_{\mathfrak{l}}, v'_{\mathfrak{l}}$, as the proof for $u'_{\mathfrak{r}}, v'_{\mathfrak{r}}$ is identical. Observe that $\kappa - k(m+1) > \tilde{\kappa} + k(m+1)$. Therefore, we can apply Lemma 7 to $u_{\mathfrak{l}}$ and $v_{\mathfrak{l}}$. This yields the following decompositions:

$$u_{\mathfrak{l}} = \prod_{i=1}^{l}(u_i \cdot a_{i \bmod m}) \cdot u_{l+1}$$
$$v_{\mathfrak{l}} = \prod_{i=1}^{l}(v_i \cdot a_{i \bmod m}) \cdot v_{l+1}$$

such that for all i, $u_i \cong_{\tilde{\kappa}} v_i$. Moreover the length l of the decompositions is $l > km$. Also observe that by definition of these decompositions, u_i, v_i use a strict subalphabet of A. Therefore, the induction hypothesis can be used and for all i we get words u'_i, v'_i such that $u'_i \cong_{\kappa'} v'_i$, $\alpha_1(u_i) = \alpha_1(u'_i)$ and $\alpha_2(v_i) = \alpha_2(v'_i)$. We now use the fact that our choice of k allows us to pump the sequences of factors into large sequences: there exist numbers $0 \leqslant j_1 < j_2 \leqslant k$ such that

$$\begin{aligned}\textstyle\prod_{i=1}^{(j_1-1)n} \alpha_1(u'_i \cdot a_{i \bmod m}) &= \textstyle\prod_{i=1}^{j_2 n} \alpha_1(u'_i \cdot a_{i \bmod m}) \\ \textstyle\prod_{i=1}^{(j_1-1)n} \alpha_2(v'_i \cdot a_{i \bmod m}) &= \textstyle\prod_{i=1}^{j_2 n} \alpha_1(u'_i \cdot a_{i \bmod m})\end{aligned}$$

Now set,

$$\begin{aligned}e_1 &= \textstyle\prod_{i=(j_1-1)n+1}^{j_2 n} u'_i \cdot a_{i \bmod m} \\ e_2 &= \textstyle\prod_{i=(j_1-1)n+1}^{j_2 n} v'_i \cdot a_{i \bmod m}\end{aligned}$$

$$\begin{aligned}u'_{\mathfrak{l}} &= \textstyle\prod_{i=1}^{j_1 n}(u'_i \cdot a_{i \bmod m}) \cdot (e_1)^{\kappa'} \cdot \prod_{i=j_2 n+1}^{l}(u'_i \cdot a_{i \bmod m}) \cdot u'_{l+1} \\ v'_{\mathfrak{l}} &= \textstyle\prod_{i=1}^{j_1 n}(v'_i \cdot a_{i \bmod m}) \cdot (e_2)^{\kappa'} \cdot \prod_{i=j_2 n+1}^{l}(v'_i \cdot a_{i \bmod m}) \cdot v'_{l+1}\end{aligned}$$

Observe that by construction $u'_{\mathfrak{l}} \cong_{\kappa'} v'_{\mathfrak{l}}$ (they are products of pairwise equivalent factors). Moreover, by construction of e_1, e_2, we have $\alpha_1(u_{\mathfrak{l}}) = \alpha_1(u'_{\mathfrak{l}})$ and $\alpha_2(v_{\mathfrak{l}}) = \alpha_1(v'_{\mathfrak{l}})$. Finally, since e_1, e_2 are 1-patterns, $u'_{\mathfrak{l}}, v'_{\mathfrak{l}}$ are κ'-patterns. Using the same construction for $u_{\mathfrak{r}}, v_{\mathfrak{r}}$, we obtain $u'_{\mathfrak{r}}, v'_{\mathfrak{r}}$. We can now finish our construction by giving w'_1, w'_2:

$$\begin{aligned}w'_1 &= u'_{\mathfrak{l}} \cdot u_{\mathfrak{c}} \cdot u'_{\mathfrak{r}} \\ w'_2 &= v'_{\mathfrak{l}} \cdot v_{\mathfrak{c}} \cdot v'_{\mathfrak{r}}\end{aligned}$$

By construction it is clear that $\alpha_1(w_1) = \alpha_1(w'_1)$ and $\alpha_2(w_2) = \alpha_2(w'_2)$. Moreover, because $u'_{\mathfrak{l}}, u'_{\mathfrak{r}}$ are κ'-patterns it follows from Lemma 8 that

$$u'_{\mathfrak{l}} \cdot u_{\mathfrak{c}} \cdot u'_{\mathfrak{r}} \cong_{\kappa'} u'_{\mathfrak{l}} \cdot v_{\mathfrak{c}} \cdot u'_{\mathfrak{r}}$$

Finally, since $u'_{\mathfrak{l}} \cong_{\kappa'} v'_{\mathfrak{l}}$ and $u'_{\mathfrak{r}} \cong_{\kappa'} v'_{\mathfrak{r}}$, we have

$$u'_{\mathfrak{l}} \cdot v_c \cdot u'_{\mathfrak{r}} \cong_{\kappa'} v'_{\mathfrak{l}} \cdot v_c \cdot v'_{\mathfrak{r}}$$

Combining the two equivalences we obtain $w'_1 \cong_{\kappa'} w'_2$, which concludes the proof.