Separating Regular Languages by Locally Testable and Locally Threshold Testable Languages

Thomas Place, Lorijn van Rooijen, and Marc Zeitoun

LaBRI, Bordeaux University, France, `firstname.lastname@labri.fr`.

Abstract

A separator for two languages is a third language containing the first one and disjoint from the second one. We investigate the following decision problem: given two regular input languages, decide whether there exists a locally testable (resp. a locally threshold testable) separator. In both cases, we design a decision procedure based on the occurrence of special patterns in automata accepting the input languages. We prove that the problem is computationally harder than deciding membership. The correctness proof of the algorithm yields a stronger result, namely a description of a possible separator. Finally, we discuss the same problem for context-free input languages.

Keywords and phrases Automata, Logics, Monoids, Locally testable, Separation, Context-free.

1 Introduction

Context. The strong connection between finite state devices and descriptive formalisms, such as first-order or monadic second-order logic, has been a guideline in computer science since the seminal work of Büchi, Elgot and Trakhtenbrot. This bridge has continuously been fruitful, disseminating tools and bringing a number of applications outside of its original research area. For instance, compiling logical specifications into various forms of automata has become one of the most successful methods in automatic program verification [23].

One of the challenging issues when dealing with a logical formalism is to precisely understand its expressiveness and its limitations. While solutions to *decide* such logics often use a compilation procedure from formulas to automata, capturing the expressive power amounts to the opposite translation: given a language, one wants to know whether one can reconstruct a formula that describes it. In other words, we want to solve an instance of the *membership problem*, which asks whether an input language belongs to some given class.

For regular languages of finite words, the main tool developed to capture this expressive power is the syntactic monoid [16]: this is a finite, computable, algebraic abstraction of the language, whose properties make it possible to decide membership. An emblematic example is the membership problem for the class of first-order definable languages, solved by Schützenberger [19] and McNaughton and Papert [14], which has led to the development of algebraic methods for obtaining decidable characterizations of logical or combinatorial properties.

The separation problem and its motivations. We consider here the *separation problem* as a generalization of the membership problem. Assume we are given two classes of languages $\mathcal{C}$ and $\mathcal{S}$. The question is, given *two* input languages from $\mathcal{C}$, whether we can separate them by a language from $\mathcal{S}$. Here, we say that a language *separates* K from L if it contains K and is disjoint from L. An obvious necessary condition for separability is that the input languages K, L be disjoint. A separator language *witnesses* this condition.

One strong motivation for this problem is to understand the limits of logics over finite words. Notice that membership reduces to separation when both $\mathcal{C}$ and $\mathcal{S}$ are closed under complement, because checking that a language belongs to $\mathcal{S}$ amounts to testing that it is $\mathcal{S}$-separable from its complement. Deciding $\mathcal{S}$-separation is also more difficult than deciding membership in $\mathcal{S}$, as one cannot rely on algebraic tools tailored to the membership problem.

It may also be computationally harder, as we shall see in this paper. Thus, solving the separation problem requires a deeper understanding of $\mathcal{S}$ than what is sufficient to check membership: one not only wants to decide whether $\mathcal{S}$ is powerful enough to *describe* a language, but also to decide whether it can *discriminate* between two input languages. This discriminating power provides more accurate information than the expressive power.

While our main concern is theoretical, let us mention some motivating applications. In model checking, reachable configurations of a system can be represented by a language. Separating it from the one representing bad configurations proves to be effective for verifying safety of a system. Craig interpolation is a form of separation used in this context [12, 9]. In this line of work, Leroux [10] simplified the proof that reachability in vector addition systems is decidable [11]: he proved that a recursively enumerable set of separators witnesses non-reachability. Finally, questions in database theory also motivated separation questions [4].

Although the separation problem frequently occurs, it has however not been systematically studied, even in the restricted, yet still challenging case of regular languages.

Contributions. In general, elements of $\mathcal{C}$ cannot always be separated by an element of $\mathcal{S}$ and there is no minimal separator wrt. inclusion. We are interested in the following questions:

- can we *decide* whether one can separate two given languages of $\mathcal{C}$ by a language of $\mathcal{S}$?
- what is the *complexity* of this decision problem?
- if separation is possible, can we *compute* a separator, and at which cost?

A motivating but difficult objective is to understand separation by first-order definable languages, whose decidability follows from involved algebraic methods [7, 8]. A first step is to look at easier subclasses. Thus, the question was raised and solved for separation by piecewise-testable [4, 18], and unambiguous languages [18], which sit in the lower levels of the quantifier-alternation hierarchy of first-order logic. In this paper, we look at yet another class, widely studied, and whose properties are orthogonal to those of the above classes.

We investigate the separation problem by locally and locally threshold testable languages. A language is *locally testable (LT)* if membership of a word can be tested by inspecting its prefixes, suffixes and infixes up to some length (which depends on the language). The membership problem for this class was raised by McNaughton and Papert [14], and solved independently by McNaughton and Zalcstein [25, 13] and by Brzozowski and Simon [3]. This class has several generalizations. The most studied is that of *locally threshold testable languages (LTT)*, where counting infixes is allowed up to some threshold. It also consists in languages definable in FO(+1), *i.e.*, first-order logic with the successor relation (but without the order). Again, membership is decidable [22], and can actually be tested in Ptime [17].

Our results are as follows: we show that separability of regular languages by LT and LTT languages is decidable, first for a fixed threshold, by reduction to fixed parameters: we provide a bound on the length of infixes that define a possible separator. This reduces the problem to a finite number of candidate separators, and hence entails decidability. For LTT-separability, we also provide a bound for the threshold. We further get an equivalent formulation on automata in terms of forbidden patterns for the languages to be separable, which yields an Nexptime algorithm. We also obtain lower complexity bounds: even starting from DFAs, the problem is Np-hard for LT and LTT (while membership to LTT is in Ptime). Finally, with discuss the separation problem starting from context-free input languages rather than regular ones. Due to lack of space, some proofs are presented in Appendix.

The main arguments rely on pumping in monoids or automata. The core of our proof is generic: we show that if one can find two words, one in each input language, that are close enough wrt. the class of separators, then the languages are not separable. Here, "close enough" is defined in terms of parameters of the input languages, such as the size of input NFAs.

Related work. The separation problem has been investigated in semigroup theory [1] when $\mathcal{S}$ is a variety of regular languages. The motivation is to obtain robust properties entailing decidable membership: having decidable separability for such a class makes it possible to decide membership for other classes built on top of it. In this line of work, it has been shown that LT-separability is decidable [20]. However, the approach suffers two drawbacks: it requires a deep intuition in finite semigroup theory, and it only provides yes/no answers. Finally, the separation problem for the class of piecewise-testable languages has been recently shown to be PTIME-decidable, independently and with different techniques in [4] and [18].

2 Preliminaries

Words and Languages. 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. If w is a word, we set $|w|$ as the *length*, or *size* of w. When w is nonempty, we view w as a sequence of $|w|$ positions labeled over A. We number positions from 0 (for the leftmost one) to $|w| - 1$ (for the rightmost one).

Infixes, Prefixes, Suffixes. An *infix* of a word w is a word w' such that $w = u \cdot w' \cdot v$ for some $u, v \in A^*$. Moreover, if $u = \varepsilon$ (resp. $v = \varepsilon$) we say that w' is a *prefix* (resp. *suffix*) of w.

Let $0 \leqslant x < y \leqslant |w|$. We write $w[x, y]$ for the infix of w starting at position x and ending at position $y - 1$. By convention, we set $w[x, x] = \varepsilon$. Observe that by definition, when $x \leqslant y \leqslant z$, we have $w[x, z] = w[x, y] \cdot w[y, z]$.

Profiles. For $k \in \mathbb{N}$, let $k_\ell = \lfloor k/2 \rfloor$ and $k_r = k - k_\ell$. A *k-profile* is a pair of words (w_ℓ, w_r) of lengths at most k_ℓ and k_r, respectively. Given $w \in A^*$ and x a position of w, the *k-profile of x* is the pair (w_ℓ, w_r) defined as follows: if $x \geqslant k_\ell$, then $w_\ell = w[x - k_\ell, x]$, otherwise $w_\ell = w[0, x]$. Symmetrically, if $x \leqslant |w| - k_r$ then $w_r = w[x, x + k_r]$ otherwise $w_r = w[x, |w|]$. We say that a k-profile (w_ℓ, w_r) *occurs in a word w* if there exists some position x within w whose k-profile is (w_ℓ, w_r). Similarly, if n is a natural number, we say that (w_ℓ, w_r) *occurs n times in w* if there are n distinct positions in w where (w_ℓ, w_r) occurs.

Intuitively, the k-profile is the description of the infix of w that is centered at position x. Observe in particular that the k-profiles that occur in a word determine the prefixes and suffixes of length $k - 1$ of this word. This is convenient, since we only have to consider one object instead of three in the usual presentations of the classes LT and LTT.

We denote by A_k the set of k-profiles over the alphabet A. It is of size exponential in k.

Separability. Given languages L, L_1, L_2 over A^*, 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 $\mathcal{S}$ of languages, we say that the pair (L_1, L_2) is *$\mathcal{S}$-separable* if some language $L \in \mathcal{S}$ separates L_1 from L_2. When $\mathcal{S}$ is closed under complement, (L_1, L_2) is $\mathcal{S}$-separable if and only if (L_2, L_1) is, in which case we simply say that L_1 and L_2 are $\mathcal{S}$-separable.

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. If δ is a function, then $\mathcal{A}$ is a deterministic finite automaton (DFA). We denote by $L(\mathcal{A})$ the language of words accepted by $\mathcal{A}$.

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 Locally Testable and Locally Threshold Testable Languages

In this paper, we investigate two classes of languages. Intuitively, a language is locally testable if membership of a word in the language only depends on the *set* of infixes, prefixes and suffixes up to some fixed length that occur in the word. For a locally threshold testable language, membership may also depend on the *number* of occurrences of such infixes, which may thus be counted up to some fixed threshold.

In this section we provide specific definitions for both classes. We start with the larger class of locally threshold testable languages. In the following, we say that two numbers are *equal up to threshold* d if either both numbers are equal, or both are greater than or equal to d.

Locally Threshold Testable Languages. Let L be a language, we say that L is *locally threshold testable* (LTT) if it is a boolean combination of languages of the form:

1. $uA^* = \{w \mid u \text{ is a prefix of } w\}$, for some $u \in A^*$.
2. $A^*u = \{w \mid u \text{ is a suffix of } w\}$, for some $u \in A^*$.
3. $\{w \mid w \text{ has } u \text{ as an infix at least } d \text{ times}\}$ for some $u \in A^*$ and $d \in \mathbb{N}$.

LTT languages can actually be defined in terms of first-order logic. A language is LTT if and only if it can be defined by an FO(+1) formula, *i.e.*, a first-order logic formula using predicates for the equality and next position relations, but not for the linear order.

We also define an index on LTT languages. Usually, this index is defined as the smallest size of infixes, prefixes and suffixes needed to define the language. However, since we only work with k-profiles, we directly define an index based on the size of k-profiles. Given words w, w' and natural numbers k, d, we write $w \equiv_k^d w'$ if for every k-profile (w_ℓ, w_r), the number of positions x such that (w_ℓ, w_r) is the k-profile of x is equal up to threshold d in w and w'.

For $k, d \in \mathbb{N}$ we denote by $\mathrm{LTT}[k, d]$ the set of languages that are unions of $\equiv_k^d$-classes. We have $\mathrm{LTT} = \bigcup_{k,d} \mathrm{LTT}[k, d]$. Given $L \subseteq A^*$, the smallest $\mathrm{LTT}[k, d]$-language containing L is

$$[L]_{\equiv_k^d} = \{w \in A^* \mid \exists u \in L \text{ such that } u \equiv_k^d w\}.$$

As it is often the case, there is no smallest LTT language containing a given regular language.

Locally Testable Languages. The class of locally testable languages is the restriction of LTT languages in which infixes cannot be counted. A language L is *locally testable* (LT) if it is a boolean combination of languages of the form 1, 2 and the following restriction of 3:

4. $A^*uA^* = \{w \mid w \text{ has } u \text{ as an infix}\}$ for some $u \in A^*$.

No simple description of LT in terms of first-order logic is known. However, there is a simple definition in terms of temporal logic. A language is LT if and only if it can be defined by a temporal logic formula using operators X (next), Y (yesterday), and G (globally).

Given two words w, w' and a number k, we write $w \equiv_k w'$ for $w \equiv_k^1 w'$. For all $k \in \mathbb{N}$, we denote by $\mathrm{LT}[k]$ the set of languages that are unions of $\equiv_k$-classes, and $\mathrm{LT} = \bigcup_k \mathrm{LT}[k]$. Given $L \subseteq A^*$ and $k \in \mathbb{N}$, the smallest $\mathrm{LT}[k]$-language containing L is

$$[L]_{\equiv_k} = \{w \in A^* \mid \exists u \in L \text{ such that } u \equiv_k w\}.$$

4 Separation for a Fixed Threshold

In this section, we prove that if the counting threshold d is fixed, it is decidable whether two languages can be separated by an LTT language of counting threshold d (*i.e.*, by an $\mathrm{LTT}[k, d]$ language for some k). In particular, this covers the case of LT, which is the case $d = 1$. All results in this section are for an arbitrary d. Our result is twofold.

- First, we prove a bound on the size of profiles that need to be considered in order to separate the languages. This bound only depends on the size of monoids recognizing the languages, and it can be computed. One can then use a brute-force algorithm that tests separability by all the finitely many LTT$[k, d]$ languages, where k denotes this bound.
- The second contribution is a criterion on the input languages to check separability by an LTT$[k, d]$ language for some k. This criterion can be defined equivalently on automata or monoids recognizing the input languages.

The section is organized into two subsections: our criterion is stated in the first one, and the second one is devoted to the statement of the theorem. The proof is presented partly in this subsection and in Appendix A.1.

4.1 Patterns

In this section we define our criterion that two languages must satisfy in order to be separable. The criterion can be defined equivalently on automata or monoids recognizing the languages.

Block Patterns. A *block* is a triple of words $\mathfrak{b} = (v_\ell, u, v_r)$ where v_ℓ, v_r are nonempty. Similarly, a *prefix block* is a pair of words $\mathfrak{p} = (u, v_r)$ with v_r nonempty, and a *suffix block* is a pair of words $\mathfrak{s} = (v_\ell, u)$ with v_ℓ nonempty. Let $d \in \mathbb{N}$. A *d-pattern* $\mathcal{P}$ is a either a word w or a triple $(\mathfrak{p}, f, \mathfrak{s})$ where $\mathfrak{p}$ and $\mathfrak{s}$ are respectively a prefix and a suffix block and f is a function mapping blocks to the set $\{0, \dots, d\}$, such that all but finitely many blocks are mapped to 0.

Decompositions. Let w be a word and let $\mathcal{P}$ be a d-pattern. We say that w *admits a $\mathcal{P}$-decomposition* if w admits a decomposition $w = u_0 v_1 u_1 v_2 \cdots v_n u_n$ with $n \geqslant 0$ and such that either $n = 0$ and $\mathcal{P} = u_0 = w$, or $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$ and the following conditions are verified:

1. $\mathfrak{p} = (u_0, v_1)$ and $\mathfrak{s} = (v_n, u_n)$.
2. for all blocks $\mathfrak{b}$, if $f(\mathfrak{b}) < d$, then there are exactly $f(\mathfrak{b})$ indices i such that $(v_i, u_i, v_{i+1}) = \mathfrak{b}$.
3. for all blocks $\mathfrak{b}$, if $f(\mathfrak{b}) = d$, then there are at least d indices i such that $(v_i, u_i, v_{i+1}) = \mathfrak{b}$.

Let $\alpha : A^* \to M$ be a morphism into a monoid M, and let $s \in M$. A $\mathcal{P}$-decomposition is *(M, s)-compatible* if $\alpha(w) = s$ and $\alpha(u_0 \cdots v_i) = \alpha(u_0 \cdots v_i) \cdot \alpha(v_i)$, for $1 \leqslant i \leqslant n$. Similarly, if $\mathcal{A}$ is an automaton, we say that a $\mathcal{P}$-decomposition is *$\mathcal{A}$-compatible* if there is an accepting run for w and each infix v_i labels a loop in the run, when $1 \leqslant i \leqslant n$, as pictured in Figure 1 (where edges denote sequences of transitions).

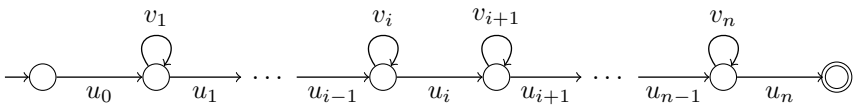

Figure 1 An $\mathcal{A}$-compatible $\mathcal{P}$-decomposition $u_0 v_1 u_1 v_2 \cdots v_n u_n$

Common Patterns. Let $d \in \mathbb{N}$ and M_1, M_2 be two monoids together with morphisms α_1, α_2, and accepting sets $F_1 \subseteq M_1$, $F_2 \subseteq M_2$. We say that M_1, M_2 have a *common d-pattern* if there exist a d-pattern $\mathcal{P}$, elements $s_1 \in F_1$, $s_2 \in F_2$, and two $\mathcal{P}$-decompositions that are respectively (M_1, s_1)-compatible and (M_2, s_2)-compatible. Similarly, if $\mathcal{A}_1, \mathcal{A}_2$ are automata, we say that $\mathcal{A}_1, \mathcal{A}_2$ have a *common d-pattern* if there exist a d-pattern $(\mathfrak{p}, f, \mathfrak{s})$ and two $\mathcal{P}$-decompositions that are respectively $\mathcal{A}_1$-compatible and $\mathcal{A}_2$-compatible. In particular, $\mathcal{A}_1, \mathcal{A}_2$ have a common 1-pattern if there are successful paths in $\mathcal{A}_1, \mathcal{A}_2$ of the form shown in Figure 1 with the same set of triples (v_i, u_i, v_{i+1}).

A useful property about common patterns is that the definitions on monoids and automata are equivalent. By this, we mean that the property of having common patterns only depends on the recognized languages, and not on the choice of $\mathcal{A}_1, \mathcal{A}_2, M_1, M_2$.

▶ **Proposition 1.** *Fix $d \in \mathbb{N}$. Let L_1, L_2 be regular languages and let $M_1, M_2, \mathcal{A}_1, \mathcal{A}_2$ be arbitrary monoids and automata recognizing L_1, L_2, respectively. Then M_1, M_2 have a common d-pattern if and only if $\mathcal{A}_1, \mathcal{A}_2$ have a common d-pattern.*

Proposition 1 is proved in the Appendix. We now state our main theorem for this section.

4.2 Separation Theorem for a Fixed Threshold

▶ **Theorem 2.** *Fix $d \in \mathbb{N}$. Let L_1, L_2 be regular languages and let $M_1, M_2, \mathcal{A}_1, \mathcal{A}_2$ be arbitrary monoids and automata recognizing L_1, L_2 respectively. Set $k = 4(|M_1||M_2| + 1)$. Then the following conditions are equivalent:*

1. *L_1 and L_2 are $\mathrm{LTT}[l, d]$-separable for some l.*
2. *L_1 and L_2 are $\mathrm{LTT}[k, d]$-separable.*
3. *The language $[L_1]_{\equiv_k^d}$ separates L_1 from L_2.*
4. *M_1, M_2 do not have a common d-pattern.*
5. *$\mathcal{A}_1, \mathcal{A}_2$ do not have a common d-pattern.*

Observe that Item 2 is essentially a *delay theorem* for separation restricted to the case of LTT: we prove that the size of profiles (*i.e.*, infixes) that a potential separator needs to consider can be bounded by a function of the size of the monoids recognizing the languages. By restricting Theorem 2 to the case $d = 1$, we get the following separation theorem for LT.

▶ **Theorem 3.** *Let L_1, L_2 be regular languages and let $M_1, M_2, \mathcal{A}_1, \mathcal{A}_2$ be arbitrary monoids and automata recognizing L_1, L_2 respectively. Let $k = 4(|M_1||M_2| + 1)$. Then the following conditions are equivalent:*

1. *L_1 and L_2 are LT-separable.*
2. *L_1 and L_2 are $\mathrm{LT}[k]$-separable.*
3. *The language $[L_1]_{\equiv_k}$ separates L_1 from L_2.*
4. *M_1, M_2 do not have a common 1-pattern.*
5. *$\mathcal{A}_1, \mathcal{A}_2$ do not have a common 1-pattern.*

Theorem 2 and Theorem 3 yield algorithms for deciding LT- and LTT-separability for a fixed threshold. Indeed, the algorithm just tests all the finitely many $\mathrm{LTT}[k, d]$ languages as potential separators. This brute-force approach yields a very costly procedure. In Appendix C, we give a deeper analysis of the complexity of the problem. In particular, a more practical algorithm will be obtained from Items 4 and 5. This yields the following corollary.

▶ **Corollary 4.** *Let $d \in \mathbb{N}$. It is decidable whether two given regular languages are $\mathrm{LTT}[l, d]$-separable for some $l \in \mathbb{N}$. In particular, it is decidable whether they are LT-separable.*

More precisely, given NFAs $\mathcal{A}_1, \mathcal{A}_2$, deciding whether $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ are LT-separable is in co-NEXPTIME. *It is* co-NP*-hard, even starting from DFAs.*

It remains to prove Theorem 2. The implications (3) ⇒ (2) ⇒ (1) are immediate by definition. We now prove (1) ⇒ (5) ⇒ (4) ⇒ (3). The implication (5) ⇒ (4) is immediate from Proposition 1. The implication (1) ⇒ (5) is a consequence of the following proposition.

▶ **Proposition 5.** *Let $d \in \mathbb{N}$ and let $\mathcal{A}_1, \mathcal{A}_2$ be NFAs. If $\mathcal{A}_1, \mathcal{A}_2$ have a common d-pattern, then, for all $k \in \mathbb{N}$, there exist w_1, w_2 accepted respectively by $\mathcal{A}_1, \mathcal{A}_2$ such that $w_1 \equiv_k^d w_2$.*

An immediate consequence of Proposition 5 is that as soon as $\mathcal{A}_1, \mathcal{A}_2$ have a common d-pattern, the recognized languages cannot be separated by an LTT$[k,d]$ language for any k. This is exactly the contrapositive of (1) $\Rightarrow$ (5). We now prove Proposition 5.

Proof. Let $\mathcal{P}$ be a common d-pattern of $\mathcal{A}_1, \mathcal{A}_2$. If $\mathcal{P} = w \in A^*$, then by definition, $w \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$, so it suffices to choose $w_1 = w_2 = w$. Otherwise, $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$ and there are w_1, w_2 having an $\mathcal{A}_1$-, respectively $\mathcal{A}_2$-compatible $\mathcal{P}$-decomposition. Let $w_1 = u_0 v_1 u_1 v_2 \cdots v_n u_n$ and $w_2 = u'_0 v'_1 u'_1 v'_2 \cdots v'_m u'_m$ be these decompositions. For $k \in \mathbb{N}$, set

$$\begin{aligned} w_1 &= u_0 (v_1)^{k(d+1)} u_1 (v_2)^{k(d+1)} \cdots (v_n)^{k(d+1)} u_n \\ w_2 &= u'_0 (v'_1)^{k(d+1)} u'_1 (v'_2)^{k(d+1)} \cdots (v'_m)^{k(d+1)} u'_m \end{aligned}$$

By definition of compatibility, $w_1 \in L(\mathcal{A}_1)$ and $w_2 \in L(\mathcal{A}_2)$. From the fact that $(\mathfrak{p}, f, \mathfrak{s})$ is a d-pattern, it then follows that $w_1 \equiv_k^d w_2$ (see appendix for details). ◀

The remaining and most difficult direction, (4) $\Rightarrow$ (3) is a consequence of the next proposition.

▶ **Proposition 6.** *Let $\alpha_1 : A^* \to M_1$ and $\alpha_2 : A^* \to M_2$ be morphisms, and $k = 4(|M_1||M_2|+1)$. Let $d \in \mathbb{N}$ and let w_1, w_2 be words such that $w_1 \equiv_k^d w_2$. Then there exists a d-pattern $\mathcal{P}$, an $(M_1, \alpha_1(w_1))$-compatible $\mathcal{P}$-decomposition, and an $(M_2, \alpha_2(w_2))$-compatible $\mathcal{P}$-decomposition.*

We defer the proof of Proposition 6 to the appendix. We now explain how to conclude the proof of Theorem 2. We prove the contrapositive of (4) $\Rightarrow$ (3). If Item 3 does not hold, then by definition there must exist $w_1 \in L_1$ and $w_2 \in L_2$ such that $w_1 \equiv_k^d w_2$. If $w_1 = w_2$, $L_1 \cap L_2 \neq \emptyset$, therefore, M_1, M_2 have a common d-pattern. Otherwise, by Proposition 6 we get a d-pattern $(\mathfrak{p}, f, \mathfrak{s})$. Since $w_1 \in L_1$ and $w_2 \in L_2$, the d-pattern $(\mathfrak{p}, f, \mathfrak{s})$ is common to both M_1 and M_2, which ends the proof.

5 Separation by LTT Languages

This section is devoted to LTT. Again our theorem actually contains several results. In the case of LTT, two parameters are involved: the size k of profiles and the counting threshold d. The first result in our theorem states that the bound on k of Theorem 2 still holds for full LTT. This means that two languages are LTT-separable if and only if there exists some counting threshold d such that they are LTT$[k,d]$-separable with the same bound k as in Theorem 2. It turns out that this already yields an algorithm for testing LTT-separability. The algorithm relies on the decidability of Presburger arithmetic and is actually adapted in a straightforward manner from an algorithm of [2] for deciding membership in LTT.

While this first result gives an algorithm for testing separability, it gives no insight about an actual separator. Indeed, the procedure does not produce the actual counting threshold d. This is what we do in the second part of our theorem: we prove that two languages are LTT-separable if and only if they are LTT$[k,d]$-separable, where k is as defined in Theorem 2, and is d bounded by a function of the size of the monoids (or automata) recognizing the input languages. Note that this result also gives another (brute-force) algorithm for testing LTT-separability. We now state our theorem. Recall that A_k denotes the set of k-profiles.

▶ **Theorem 7.** *Let L_1, L_2 be regular languages and let $M_1, M_2, \mathcal{A}_1, \mathcal{A}_2$ be arbitrary monoids and automata recognizing L_1, L_2. Set n to be either $\max(|M_1|,|M_2|)+1$ or $\max(|\mathcal{A}_1|,|\mathcal{A}_2|)+1$. Let $k = 4(|M_1||M_2|+1)$ and $d = (|A_k|n)^{|A_k|}$. Then, the following conditions are equivalent:*

1. *L_1 and L_2 are LTT-separable.*
2. *There exists $d' \in \mathbb{N}$ such that L_1 and L_2 are LTT$[k,d']$-separable.*

3. *There exists $d' \in \mathbb{N}$ such that M_1, M_2 do not have a common d'-pattern.*
4. *There exists $d' \in \mathbb{N}$ such that $\mathcal{A}_1, \mathcal{A}_2$ do not have a common d'-pattern.*
5. *L_1 and L_2 are* LTT$[k, d]$*-separable.*
6. *The language $[L_1]_{\equiv_k^d}$ separates L_1 from L_2.*

Observe that decidability of LTT-separability is immediate from Item 5 by using the usual brute-force algorithm. As it was the case for a fixed counting threshold, this algorithm is slow. It is possible to obtain a faster algorithm by using Items 3 and 4. We investigate this question in Appendix C, which is devoted to complexity. This yields the following corollary.

▶ **Corollary 8.** *It is decidable whether two given regular languages are* LTT*-separable.*

More precisely, given NFAs $\mathcal{A}_1, \mathcal{A}_2$, deciding whether $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ are LT*-separable is in* 2-Expspace*. It is* co-NP*-hard, even starting from DFAs.*

By definition, a language is LTT if it is LTT$[k, d]$ for some natural numbers k, d. Hence, the equivalence between Items 1, 2, 3 and 4 is an immediate consequence of Theorem 2. Therefore, we only need to prove Items 5 and 6, *i.e.*, the bound on the threshold d. Unfortunately, these are exactly the items we need for Corollary 8. However, we will prove that by reusing an algorithm of [2], Corollary 8 can also be derived directly from Item 2.

The section is organized into two subsections. First we explain how Corollary 8 can be derived from Item 2 without actually having to compute a bound on the counting threshold. Then, we prove our bound on the counting threshold. We shall discuss the optimality of the counting threshold in Appendix B.

5.1 Decidability of LTT-separability as a consequence of Theorem 2

As we explained, the equivalence of Item 2 to LTT-separability is immediate from Theorem 2. We explain how to combine this fact with an algorithm of [2] to obtain decidability directly.

In [2], it is proved that once k is fixed, Parikh's Theorem [15] can be used to prove that whether a language is LTT$[k, d]$ for some d can be rephrased as a computable Presburger formula. Decidability of membership in LTT can then be reduced to decidability of Presburger Arithmetic. For achieving this, two ingredients were needed: *a*) a bound on k, and *b*) the translation to Presburger arithmetic. It turns out that in [2], only the proof of *a*) was specific to membership. On the other hand, separation was already taken care of in *b*), because the intuition behind the Presburger formula was testing *separability* between the input language and its complement. In our setting, we have already replaced *a*), *i.e.*, bounding k, by Item 2. Therefore, the argument can be generalized. We explain in the remainder of this subsection how to construct the Presburger formula. The argument makes use of the notion of Parikh image, which we now recall.

Parikh Images. Let $w \in A^*$, the *Parikh image* of w, denoted by $\pi(w)$, is a vector of length $|A|$ of natural numbers counting, for every $a \in A$, how many occurrences of a there are in w. This notion can be easily generalized in order to count profiles rather than just letters. Let $k \in \mathbb{N}$. The *k-image* of w, $\pi_k(w)$, is a vector of length $|A_k|$ of numbers counting for every k-profile (w_ℓ, w_r) the number of positions in w with k-profile (w_ℓ, w_r). If L is a language, the *k-image* of L, $\pi_k(L)$ is the set $\{\pi_k(w) \mid w \in L\}$. The definition of $\equiv_k^d$ yields the following fact.

▶ **Fact 9.** *Let $w, w' \in A^*$ and $k, d \in \mathbb{N}$. Then $w \equiv_k^d w'$ if and only if $\pi_k(w)$ and $\pi_k(w')$ are equal componentwise up to threshold d.*

A well-known result about Parikh images is Parikh's Theorem [15], which states that if L is context-free (and so in particular if L is regular), then $\pi(L)$ is semilinear, *i.e.*, Presburger definable [6]. As explained in [2], Parikh's Theorem extends without difficulty to k-images.

▶ **Theorem 10.** *Let L be a context-free language and let $k \in \mathbb{N}$. Then $\pi_k(L)$ is semilinear. Moreover, a Presburger formula for this semilinear set can be computed.*

Proof. When $k = 1$, the proposition is Parikh's Theorem. When $k > 1$, consider the language L' over the alphabet A_k of k-profiles such that $w' \in L'$ if and only if there exists $w \in L$ of the same length and such that a position in w' is labeled by its k-profile in w. It is straightforward to see that L' is context-free and that the k-image of w is its Parikh image, which is semilinear by Parikh's Theorem. ◀

We can now explain how to decide LTT-separability. By Item 2 in Theorem 7, L_1, L_2 are LTT-separable if and only if they are LTT$[k, d]$-separable for $k = 4(|M_1||M_2| + 1)$ (where M_1, M_2 are monoids recognizing L_1, L_2) and some natural d. Therefore, whether L_1, L_2 are LTT-separable can be rephrased as follows: does there exist some threshold d such that there exist no words $w_1 \in L_1, w_2 \in L_2$ such that $w \equiv_k^d w'$? By Fact 9, this can be expressed in terms of k-images: does there exist a threshold d such that such that there exist no vectors of natural numbers $\bar{x}_1 \in \pi_k(L_1), \bar{x}_2 \in \pi_k(L_2)$ that are equal up to threshold d? It follows from Theorem 10 that the above question can be expressed as a computable Presburger formula. Decidability of LTT-separability then follows from decidability of Presburger Arithmetic.

5.2 Bounding the Counting Threshold

In this section we prove that Items 5 and 6 in Theorem 7 are equivalent to Item 2. By definition, it is immediate that (6) $\implies$ (5) $\implies$ (2). Therefore, it suffices to prove that (2) $\implies$ (6).

We actually prove the contrapositive. If $[L_1]_{\equiv_k^d}$ does not separate L_1 from L_2 for the values of k, d defined in the theorem, then there is no LTT$[k, d']$-separator for any d'.

Assume that $[L_1]_{\equiv_k^d}$ is not a separator. By definition, this means that there exist $w_1 \in L_1$ and $w_2 \in L_2$ such that $w_1 \equiv_k^d w_2$. Set d' some arbitrary natural, we prove that d is large enough to construct words $w_1' \in L_1, w_2' \in L_2$ such that $w_1' \equiv_k^{d'} w_2'$. By definition of $\equiv_k^{d'}$, this means that there exists no LTT$[k, d']$-separator, which finishes the proof.

Recall that $d = (|A_k|n)^{|A_k|}$ with $n = \max(|M_1|, |M_2|) + 1$ or $n = \max(|\mathcal{A}_1|, |\mathcal{A}_2|) + 1$. To simplify the writing, we only treat the case when $n = \max(|\mathcal{A}_1|, |\mathcal{A}_2|) + 1$. The other case can be proved using similar arguments. The main idea behind the construction is that by choice of d, k-profiles that occur many times in w_1, w_2 can be pumped in more than d' occurrences. This is summarized in the following lemma.

▶ **Lemma 11.** *Let $h \in \mathbb{N}$, $h' \geqslant n|A_k|h$ and $w \in L_1$ (resp. $w \in L_2$). One can construct a word $w' \equiv_k^h w$ such that $w' \in L_1$ (resp. $w' \in L_2$) and every k-profile that occurs h' or more times in w occurs d' or more times in w'.*

Proof. By symmetry, we may assume that $w \in L_1$. The intuition is that as soon as a k-profile occurs more than $|\mathcal{A}_1| + 1$ (which is the case if it occurs more than n times), there exist two occurrences of this k-profile that are labeled with the same state in the run of $\mathcal{A}_1$ on w. Therefore, pumping can be used on w to generate d' copies of the k-profile without affecting membership in L_1. The issue with this argument is that in order to enforce $w' \equiv_k^h w$, we need to be careful and avoid duplicating k-profiles that occur less than h times in w. This is why we actually need a much higher constant than n in order to achieve the pumping.

Let (w_ℓ, w_r) be some k-profile that occurs more than h' times in w (if there is none, it suffices to take $w' = w$). We explain how w can be pumped in w'' that contains more than d' copies of (w_ℓ, w_r) while enforcing $w'' \equiv_k^h w$. The construction can then be repeated to treat all k-profiles occurring more than h' times in w, in order to get the desired w'.

Let $x_1 < \cdots < x_{h'}$ be h' positions where (w_ℓ, w_r) occurs. Observe that there are at most $|A_k|(h-1)$ positions in w such that the k-profile at this position occurs strictly less than h times in w. By choice of $h' \geqslant n|A_k|h$, a simple combinatorial argument yields that there exist at least n consecutive positions in the list, say $x_i, \ldots, x_{i+(n-1)}$, such that no intermediate position between x_i and $x_{i+(n-1)}$ has a k-profile occurring less than h times in w. By choice, of n, there are two positions among $x_i, \ldots, x_{i+(n-1)}$ that are labeled with the same state in the run of $\mathcal{A}_1$ on w. Therefore, the corresponding infix can be pumped to generate d' copies of (w_ℓ, w_r) without affecting membership in L_1. Moreover, by choice of the positions $x_i, \ldots, x_{i+(n-1)}$, the pumping did not duplicate k-profiles occurring less than h times in w. Therefore, the resulting word w'' verifies $w'' \equiv_k^h w$ and $w'' \in L_1$. ◀

We now use Lemma 11 to finish the construction, which is actually achieved by applying Lemma 11 recursively to w_1, w_2. We formalize this in an inductive property.

Set $l \leqslant |A_k|$, we prove the following property $\mathcal{P}(l)$. Assume that $u_1 \in L_1$ and $u_2 \in L_2$ are words such that $u_1 \equiv_k^{(|A_k|n)^l} u_2$ and the number of k-profiles that do not occur more than d' times in both u_1 and u_2 is smaller than l. Then there exist words $u_1' \in L_1$ and $u_2' \in L_2$ such that $u_1' \equiv_k^{d'} u_2'$. Observe that w_1, w_2 verify $\mathcal{P}(|A_k|)$. Therefore, we obtain the desired $w_1' \equiv_k^{d'} w_2'$. It remains to prove that $\mathcal{P}(l)$ holds for all $l \leqslant |A_k|$. We do this by induction on l.

First, if $l = 0$, the result is obvious since by definition all k-profiles in u_1, u_2 occur more than d' times in both words and therefore $u_1 \equiv_k^{d'} u_2$.

Assume now that $l > 0$. If $u_1 \equiv_k^{d'} u_2$, then it suffices to take $u_1' = u_1$ and $u_2' = u_2$. Otherwise, since $u_1 \equiv_k^{(|A_k|n)^l} u_2$, this means that there exists at least one k-profile (w_ℓ, w_r) that occurs more than $(|A_k|n)^l$ times in both u_1 and u_2 but less than d' times in at least one of the two words. By applying Lemma 11 to both u_1 and u_2 with $h = (|A_k|n)^{l-1}$ and $h' = (|A_k|n)^l$, we get two words $u_1'' \in L_1$ and $u_2'' \in L_2$ such that $u_1'' \equiv_k^h u_2''$. Moreover, (w_ℓ, w_r) now occurs more than d' times in both u_1'' and u_2''. Therefore, the number of k-profiles that do not occur more than d' times in both u_1'' and u_2'' is smaller than $l-1$. Hence, we can apply the induction hypothesis to u_1'' and u_2'' which yields the desired u_1', u_2'.

6 The Case of Context-Free Languages

In order to prove decidability of LTT-separability for regular languages, we needed three ingredients: Parikh's Theorem, decidability of Presburger Arithmetic and Item 2 in Theorem 7. Since Parikh's Theorem holds not only for regular languages but also for context-free languages, we retain at least two of the ingredients in the context-free setting.

In particular, we can reuse the argument of Section 5 to prove that once the size k of the infixes is fixed, separability by LTT is decidable for context-free languages. For any fixed $k \in \mathbb{N}$, we write $\mathrm{LTT}[k] = \bigcup_{d \in \mathbb{N}} \mathrm{LTT}[k, d]$.

▶ **Theorem 12.** *Let L_1, L_2 be context-free languages and $k \in \mathbb{N}$. It is decidable whether L_1, L_2 are $\mathrm{LTT}[k]$-separable.*

An interesting consequence of Theorem 12 is that LTT[1]-separability of context-free languages is decidable. A language is LTT[1] if and only if it can be defined by a first-order logic formula that can only test equality between positions, but not ordering. This result is surprising since membership of a context-free language in this class is undecidable. We give a proof of this fact below, which is a simple adaptation of the proof of Greibach's Theorem (which is in particular used to prove that regularity of a context-free language is undecidable).

▶ **Theorem 13.** *Let L be a context-free language. It is undecidable to test whether $L \in \mathrm{LTT}[1]$.*

Proof. We reduce universality of context-free languages to this membership problem. Fix L a context-free language over A and let $\# \notin A$. Let $K \notin \mathrm{LTT}[1]$ be some context-free language and set $L_1 = (K \cdot \# \cdot A^*) \cup (A^* \cdot \# \cdot L)$. Clearly, a context-free grammar for L_1 can be computed from a context-free grammar for L. We show that $L = A^*$ if and only if $L_1 \in \mathrm{LTT}[1]$.

If that $L = A^*$, then $L_1 = A^* \cdot \# \cdot A^* \in \mathrm{LTT}[1]$. Conversely, assume that $L_1 \in \mathrm{LTT}[1]$, and suppose by contradiction that $L \neq A^*$. Pick $w \in A^*$ such that $w \notin L$. By definition, $K = \{u \mid u\#w \in L_1\}$. One can verify that LTT[1] is closed under right residual. Therefore, $K = L_1(\#w)^{-1} \in \mathrm{LTT}[1]$ which is a contradiction by definition of K. ◀

These two results may seem contradictory. Indeed in the setting of regular languages, membership can be reduced to separability (a language belongs to a class if the class can separate it from its complement). However, context-free languages are not closed under complement, which makes the reduction false in this larger setting.

An interesting question is whether decidability extends to full LT and LTT-separability of context-free languages. This would also be surprising since membership of a context-free language in LT or LTT are undecidable problems. Such a result would require to generalize our third ingredient, Item 2 in Theorem 7, to context-free languages. This means that we would need a method for computing a bound on the size of the infixes that a potential separator has to consider. It turns out that this is not possible.

▶ **Theorem 14.** *Let L_1, L_2 be context-free languages. It is undecidable to test whether L_1, L_2 are* LT*-separable. It is undecidable to test whether L_1, L_2 are* LTT*-separable.*

It was already known [21] that separability by a regular language is undecidable for context-free languages. The proof of Theorem 14 is essentially the same since the reduction provided in [21] actually works for any class of regular separators that contains languages of the form $K_1A^* \cup K_2$ where K_1, K_2 are finite languages. Since this is clearly the case for both LT and LTT, Theorem 14 follows. For the sake of completeness, we provide a version of this proof tailored to LT and LTT in the appendix.

7 Conclusion

We proved separation theorems for both LT and LTT. In both cases, we provide a decision procedure to test separability, running in co-NEXPTIME and 2-EXPSPACE respectively. Another contribution is a description of possible separators, given by bounds defining them.

Several questions remain open in this line of research. A first one is to obtain tight complexity bounds for both classes. While we have co-NEXPTIME and 2-EXPSPACE upper bounds for LT and LTT respectively, we have only co-NP lower bounds. The upper bounds rely on a reduction to the case $k = 1$, *i.e.*, a translation to the special case when the size of infixes is fixed to 1 (see Appendix C). This translation is exponential wrt. the size of the input automata. Improving the upper bounds would likely require improving this reduction.

Another question is to consider other fragments for separability. A natural generalization of LTT is LTT+MOD, in which infixes can now also be counted modulo constants. The most interesting fragment is of course full first-order logic. While the problem was shown decidable [7, 8], the proofs rely on involved algebraic techniques and give an algorithm that provides only a yes/no answer. Furthermore, the techniques bring no insight on the expressive power of first-order logic. It remains a challenging open problem to obtain a combinatorial proof that FO-separability is decidable, as well as a description of a separator.

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 M. Bojanczyk. A new algorithm for testing if a regular language is locally threshold testable. *Inf. Process. Lett.*, 104(3):91–94, 2007.

3 J. Brzozowski and I. Simon. Characterizations of locally testable events. *Discrete Mathematics*, 4(3):243–271, 1973.

4 W. Czerwinski, W. Martens, and T. Masopust. Efficient separability of regular languages by subsequences and suffixes. In *Proc. of ICALP'13*, pages 150–161, 2013.

5 M. Fürer. The complexity of Presburger arithmetic with bounded quantifier alternation depth. *Theoretical Computer Science*, 18(1), 1982.

6 S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.

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. *Internat. J. Algebra Comput.*, 20(2):287–305, 2010.

9 T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *Proc. of POPL'04*, pages 232–244. ACM, 2004.

10 J. Leroux. Vector addition systems reachability problem (a simpler solution). In *The Alan Turing Centenary Conference, Turing-100*, volume 10, pages 214–228, 2012.

11 E. W. Mayr. An algorithm for the general petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.

12 K. L. McMillan. Applications of Craig interpolants in model checking. In N. Halbwachs and L. D. Zuck, editors, *Proc. of TACAS'05*, pages 1–12. Springer, 2005.

13 R. McNaughton. Algebraic decision procedures for local testability. *Math. Systems Theory*, 8(1):60–76, 1974.

14 R. McNaughton and S. Papert. *Counter-free automata*. The M.I.T. Press, 1971.

15 R. J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.

16 J.-E. Pin. Syntactic semigroups. In G. Rozenberg and A. Salomaa, editors, *Handbook of language theory, Vol. I*, pages 679–746. Springer, 1997.

17 J.-E. Pin. Expressive power of existential first-order sentences of Büchi's sequential calculus. *Discrete Math.*, 291(1–3):155–174, 2005.

18 T. Place, L. van Rooijen, and M. Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *Proc. of MFCS'13*, 2013.

19 M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

20 B. Steinberg. A delay theorem for pointlikes. *Sem. Forum*, 63(3):281–304, 2001.

21 T. G. Szymanski and J. H. Williams. Noncanonical extensions of bottom-up parsing techniques. *SIAM J. Comput.*, 5(2), 1976.

22 D. Thérien and A. Weiss. Graph congruences and wreath products. *J. Pure Appl. Algebra*, 36:205–215, 1985.

23 M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. of LICS'86*, pages 332–344. IEEE Computer Society, 1986.

24 K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational Horn clauses. In *Proceedings of the 20th International Conference on Automated (CADE'05)*, 2005.

25 Y. Zalcstein. Locally testable languages. *J. Comput. Syst. Sci.*, 6(2):151–167, 1972.

A Appendix to Section 4: proofs

In this section, we provide proofs and details for propositions 1, 5 and 6 (we devote a subsection for the last one).

▶ **Proposition 1.** *Set $d \in \mathbb{N}$. Let L_1, L_2 be regular languages and let $M_1, M_2, \mathcal{A}_1, \mathcal{A}_2$ be arbitrary monoids and automata recognizing L_1, L_2. Then M_1, M_2 have a common d-pattern if and only if $\mathcal{A}_1, \mathcal{A}_2$ have a common d-pattern.*

Proof. Observe that the only direction in the proof of Proposition 1 that was needed in the proof Theorem 2 is that if M_1, M_2 have a common d-pattern, then so does $\mathcal{A}_1, \mathcal{A}_2$ have a common d-pattern. Therefore, it suffices to prove this direction, the converse one will then follow from Theorem 2.

Even for this direction, we recover some arguments from the proof of Theorem 2. Indeed, by adapting Proposition 5 to monoids, we can prove the equivalence between Items 1, 2, 3 and 4 without relying on Proposition 1. In particular this proves that as soon as M_1, M_2 have a common d-pattern, then so does any other pair of monoids recognizing L_1, L_2. This is in particular true of the transitions monoids of $\mathcal{A}_1, \mathcal{A}_2$. It is then straightforward to verify that this means that $\mathcal{A}_1, \mathcal{A}_2$ have a common d-pattern. ◀

▶ **Proposition 5.** *Let $d \in \mathbb{N}$ and let $\mathcal{A}_1, \mathcal{A}_2$ be NFAs. If $\mathcal{A}_1, \mathcal{A}_2$ have a common d-pattern, then, for all $k \in \mathbb{N}$, there exist w_1, w_2 accepted respectively by $\mathcal{A}_1, \mathcal{A}_2$ such that $w_1 \equiv_k^d w_2$.*

Proof. Let $\mathcal{P}$ be a common d-pattern of $\mathcal{A}_1, \mathcal{A}_2$. If $\mathcal{P} = w \in A^*$, then by definition, $w \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$, so it suffices to choose $w_1 = w_2 = w$. Otherwise, $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$ and there are w_1, w_2 having an $\mathcal{A}_1$-, respectively $\mathcal{A}_2$-compatible $\mathcal{P}$-decomposition. Let $w_1 = u_0 v_1 u_1 v_2 \cdots v_n u_n$ and $w_2 = u'_0 v'_1 u'_1 v'_2 \cdots v'_m u'_m$ be these decompositions. For $k \in \mathbb{N}$, set

$$\begin{aligned} w_1 &= u_0 (v_1)^{k(d+1)} u_1 (v_2)^{k(d+1)} \cdots (v_n)^{k(d+1)} u_n \\ w_2 &= u'_0 (v'_1)^{k(d+1)} u'_1 (v'_2)^{k(d+1)} \cdots (v'_m)^{k(d+1)} u'_m \end{aligned}$$

By definition of compatibility, $w_1 \in L(\mathcal{A}_1)$ and $w_2 \in L(\mathcal{A}_2)$. From the fact that $(\mathfrak{p}, f, \mathfrak{s})$ is a d-pattern, it then follows that $w_1 \equiv_k^d w_2$. Indeed, since both decompositions of w_1 and w_2 $\mathcal{P}$-compatible, we have $u_0 = u'_0$ and $v_1 = v'_1$, which implies that w_1 and w_2 have the same prefix of length $k-1$ (and actually, even of length $k(d+1)$). Similarly, they have the same suffix of length $k-1$. It remains to verify that each word of length at most k occurs the same number of times, up to threshold d, as an infix in w_1 and w_2.

Let u be an infix of w_1 of length at most k. Assume first that u occurs in some $v_i^{k(d+1)}$. Then it occurs at least d times. Since the decompositions of w_1, w_2 are $\mathcal{P}$-compatible, there exists some j such that $v'_j = v_i$. Therefore, u occurs at least d times as an infix in $(v'_j)^{k(d+1)}$ as well, hence also in w_2.

Finally, assume that u does not occur in any $v_i^{k(d+1)}$. Therefore, it overlaps with some of the u_i's, and for these indices i, it is an infix of $(v_i)^{k(d+1)} u_i (v_{i+1})^{k(d+1)}$. Since both decompositions of w_1 and w_2 are $\mathcal{P}$-compatible, the number of triples (v_i, u_i, v_{i+1}) in the decomposition of w_1 and (v'_j, u'_j, v'_{j+1}) in that of w_2 which are equal to a given triple is the same up to threshold d. Therefore, u occurs the same number of times up to threshold d in both w_1 and w_2. Therefore, $w_1 \equiv_k^d w_2$. ◀

A.1 Proof of Proposition 6

This subsection is devoted to the proof of Proposition 6.

▶ **Proposition 6.** *Let $\alpha_1 : A^* \to M_1$ and $\alpha_2 : A^* \to M_2$ be morphisms, and $k = 4(|M_1||M_2|+1)$. Let $d \in \mathbb{N}$ and let w_1, w_2 be words such that $w_1 \equiv_k^d w_2$. Then there exists a d-pattern $\mathcal{P}$, an $(M_1, \alpha_1(w_1))$-compatible $\mathcal{P}$-decomposition, and an $(M_2, \alpha_2(w_2))$-compatible $\mathcal{P}$-decomposition.*

We set w_1, w_2, k and d as in the statement of the proposition. Observe first that if $w_1 = w_2 = w$, then it suffices to take $\mathcal{P} = w$. Therefore, we suppose for the remainder of the proof that $w_1 \neq w_2$. We proceed as follows: we construct two new words w_1', w_2' from w_1, w_2 and prove that w_1' admits a $(M_1, \alpha_1(w_1))$-compatible $\mathcal{P}$-decomposition and w_2' admits a $(M_2, \alpha_2(w_2))$-compatible $\mathcal{P}$-decomposition, for some d-pattern $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$.

We begin with the construction of w_1', w_2', which amounts to duplicating infixes in w_1, w_2 verifying special properties. We first define these special infixes that we will call k-loops.

k-loops. Let $w \in A^*$, x be a position in w, and (w_ℓ, w_r) be the $\lfloor k/2 \rfloor$-profile of x. We say that x *admits a k-loop* if there exists a nonempty prefix u of w_r such that $\alpha_1(w_\ell) = \alpha_1(w_\ell \cdot u)$ and $\alpha_2(w_\ell) = \alpha_2(w_\ell \cdot u)$. In this case, we call the smallest such u *the k-loop of x*. See Fig. 2.

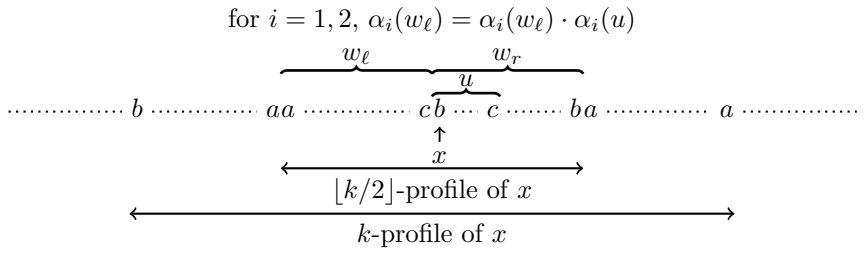

Figure 2 A position x admitting a k-loop u, that is: $\alpha_i(w_\ell) = \alpha_i(w_\ell \cdot u)$ for $i = 1, 2$

We state properties of k-loops that we will need in order to prove that our construction is correct. We begin by two simple facts that are immediate from the definition.

▶ **Fact 15.** *Let x be a position. Whether x admits a k-loop, and if so, which k-loop x admits, only depend on the $\lfloor k/2 \rfloor$-profile of x.*

▶ **Fact 16.** *Let w be a word and let x be a position within w such that x admits a k-loop u. Then for $i = 1, 2$, $\alpha_i(w[0, x]) = \alpha_i(w[0, x]) \cdot \alpha_i(u)$.*

The last property we need is that at least one of $\lfloor k/4 \rfloor$ consecutive positions must admit a k-loop. This follows from pumping arguments:

▶ **Lemma 17.** *Let w be a word and let $x_1, \ldots, x_{\lfloor k/4 \rfloor}$ be $\lfloor k/4 \rfloor$ consecutive positions in w. Then, there exists at least one position x_i with $i < \lfloor k/4 \rfloor$ that admits a k-loop.*

Proof. By choice of k, $\lfloor k/4 \rfloor = |M_1||M_2| + 1$. By the pigeonhole principle, we obtain two natural numbers $x_1 \leqslant x_i < x_j \leqslant x_{\lfloor k/4 \rfloor}$ such that $\alpha_1(w[x_1, x_i]) = \alpha_1(w[x_1, x_j])$ and $\alpha_2(w[x_1, x_i]) = \alpha_2(w[x_1, x_j])$. We prove that x_i admits a k-loop. Consider the $\lfloor k/2 \rfloor$-profile (w_ℓ, w_r) of x_i and $u = w[x_i, x_j]$. Since $|w[x_1, x_i]| < \lfloor k/4 \rfloor$, $w[x_1, x_i]$ is a suffix of w_ℓ, so $\alpha_1(w_\ell) = \alpha_1(w_\ell \cdot u)$ and $\alpha_2(w_\ell) = \alpha_2(w_\ell \cdot u)$. Since $|u| < \lfloor k/4 \rfloor$, u is a prefix of w_r. Therefore x_i admits a k-loop.

Observe that u is not necessarily *the* k-loop of x_i, as there might be a smaller word that also satisfies the definition. ◀

Construction of w'_1, w'_2. We can now construct w'_1 and w'_2. If w, u are words and x is a position of w, the *word constructed by inserting u at position x* is the word $w[0,x]\cdot u\cdot w[x,|w|]$. From w_1 (resp. w_2), we construct w'_1 (resp. w'_2) by inserting simultaneously all infixes z_x in w_1 (resp. w_2) at any position x that admits a k-loop, and where z_x is the k-loop of x.

If we had $\min(|w_1|,|w_2|) \leqslant \lfloor k/4 \rfloor$, then from $w_1 \equiv_k^d w_2$ one would obtain $w_1 = w_2$, a case already excluded. Hence, both w_1, w_2 have length at least $\lfloor k/4 \rfloor$, so by Lemma 17, at least one insertion has occurred in both w_1 and w_2. We now prove that there exists a d-pattern $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$ such that for $i = 1, 2$, w'_i admits a $(M_i, \alpha_i(w_i))$-compatible $\mathcal{P}$-decomposition.

We first define the d-pattern $(\mathfrak{p}, f, \mathfrak{s})$. By definition, w'_1 can be decomposed as $w'_1 = u_0v_1u_1v_2\cdots v_nu_n$ with $w_1 = u_0u_1\cdots u_n$ and the words v_j are the k-loops inserted in the construction. Since at least one insertion was made, we have $n \geqslant 1$ and we can set $\mathfrak{p} = (u_0, v_1)$, $\mathfrak{s} = (v_n, u_n)$. We define f as the function that maps a block (v_ℓ, u, v_r) to the number of times it occurs in the decomposition, up to threshold d. Set $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$. By definition, $u_0v_1u_1v_2\cdots v_nu_n$ is a $\mathcal{P}$-decomposition for w'_1. Moreover, it is $(M_1, \alpha_1(w_1))$-compatible by Fact 16. We now prove that w'_2 admits an $(M_2, \alpha_2(w_2))$-compatible $\mathcal{P}$-decomposition.

By definition, w'_2 can be decomposed in a similar way as w'_1, $w'_2 = u'_0v'_1u'_1v'_2\cdots v'_mu'_m$ with $w_2 = u'_0u'_1\cdots u'_m$ and the words v'_j are the k-loops inserted in the construction of w'_2. We prove that this is a $\mathcal{P}$-decomposition. It will then be $(M_2, \alpha_2(w_2))$-compatible by Fact 16.

To every position x in w_2 we associate a block, prefix block or suffix block in the following way. By definition, x must fall into a word u'_i for some i. If $i \notin \{0, m\}$, we denote by $g(x)$ the triple (v'_i, u'_i, v'_{i+1}). Similarly if $i = 0$ (resp. $i = m$), then $g(x)$ is the pair (u'_0, v'_1) (resp. (v'_m, u'_m)). The result now follows from the next lemma.

▶ **Lemma 18.** *Let x, y be distinct positions of words w_1 or w_2 with the same k-profile. Then $g(x) = g(y)$. Moreover, the number of copies of a block $\mathfrak{b}$ in the decomposition of w_1 (resp. w_2) is exactly the number of positions x in w_1 (resp. w_2) such that $g(x) = \mathfrak{b}$.*

Proof. There are several cases to treat depending on whether x, y are in w_1 or w_2 and whether there are at least $\lfloor k/4 \rfloor$ positions to the right and left of x, y or not. All cases are treated similarly. Therefore, we only treat the case when there are at least $\lfloor k/4 \rfloor$ positions to the right and left of x, y and x, y are both in w_1.

Let z be a position such that there are at least $\lfloor k/4 \rfloor$ positions to the right and left of z. Let w_ℓ, w_r be the k-profile of z and $z_\ell \leqslant z$ and $z_r > z$ be the positions admitting k-loops that are closest to z. We prove the following property: z_ℓ, z_r belong to the copy of w_ℓ, w_r at z, moreover their relative positions in this copy and the actual k-loops depend only on w_ℓ, w_r. This immediately proves by construction of w'_1, w'_2 that $g(z)$ only depends on its k-profile and hence the first part of the lemma follows. Moreover, this also proves that if $(v_\ell, u, v_r) = g(z)$, then the relative position of z within the corresponding copy of u only depends on the k-profile of z. This means that two positions with the same k-profile can only generate the same copy of a block if they are equal. This proves the second part of the lemma.

We now prove the property. Let w_ℓ, w_r be the k-profile of z. Let $z_\ell \leqslant z$ and $z_r > z$ be the positions admitting k-loops that are closest to z. Observe that by Lemma 17, $z - z_\ell \leqslant \lfloor k/4 \rfloor$ and $z_r - z \leqslant \lfloor k/4 \rfloor - 1$. By definition of profiles, the $\lfloor k/2 \rfloor$-profiles of all such positions are determined by the k-profile of z (see Figure 3). By Fact 15, this means that z_ℓ, z_r as well as their actual k-loop are determined by the k-profile of z which terminates the proof. ◀

It is immediate from Lemma 18 by taking x, y as the first positions (resp. last positions) of w_1, w_2 that $\mathfrak{p} = (u_0, v_1) = (u'_0, v'_1)$ (resp. $\mathfrak{s} = (v_n, u_n) = (v'_m, u'_m)$). We finish by proving

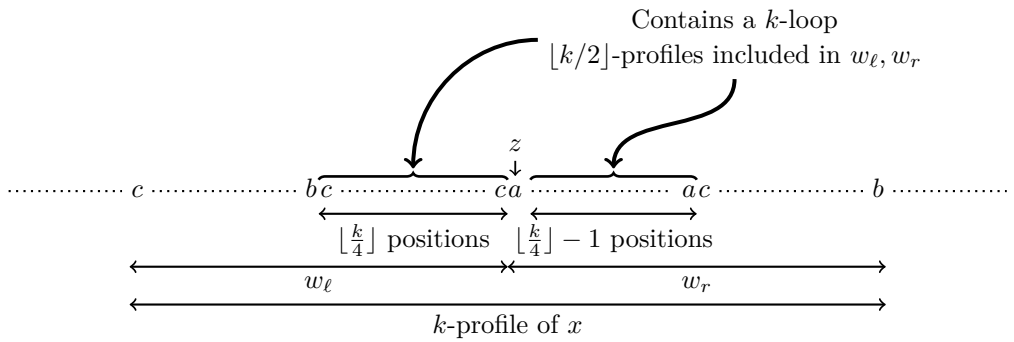

Figure 3 Construction in Lemma 18

that for every block (v_ℓ, u, v_r) the number of indices i such that $(v_\ell, u, v_r) = (v_i, u_i, v_{i+1})$ is the same in the decompositions of w'_1, w'_2 up to threshold d. By definition of $(\mathfrak{p}, f, \mathfrak{s})$ this will prove that the decomposition of w'_2 is indeed a $\mathcal{P}$-decomposition.

Let $\mathfrak{b} = (v_\ell, u, v_r)$ be a block. By Lemma 18, there exists a set P of k-profiles such that of indices i in w'_1 (resp. w'_2) such that $\mathfrak{b} = (v_i, u_i, v_{i+1})$ is exactly the number of positions in w_1 (resp. in w_2) with a k-profile in P. Because $w_1 \equiv_k^d w_2$ these numbers are then equal in w_1, w_2 up to threshold d and this finishes the proof.

B Appendix to Section 5: Optimality of the Counting Threshold

Observe that the bound we prove in Theorem 7 for the counting threshold is exponential in the size of $|A_k|$ (which is itself exponential in the size of the monoids). A relevant question is to know if this bound can be improved.

Our proof completely separates the bounding of k and d. We first provide a bound on k in Theorem 2. Then, we bound the threshold by essentially viewing our words as words over the alphabet A_k of k-profiles. This technique ignores properties of k-profiles. In particular, the k-profiles of adjacent positions are strongly connected, a fact that our proof does not exploit.

In this subsection, we prove that getting a better bound on the counting threshold would require to take these additional properties into account. More precisely, we prove that if $k = 1$ (which means the k-profile of a position is just its label), we can construct separable languages for which the separator requires a counting threshold that is exponential in $|A|$.

For convenience, we assume that the alphabet A is of even size and write $A = \{a_1, \dots, a_{2m}\}$. Consider the following languages

$$\begin{aligned} L_1 &= a_1 \cdot (a_2a_3a_3)^*(a_4a_5a_5)^* \cdots (a_{2m-2}a_{2m-1}a_{2m-1})^* \cdot (a_{2m}a_{2m}a_{2m})^* \\ L_2 &= (a_1a_2a_2)^*(a_3a_4a_4)^* \cdots (a_{2m-1}a_{2m}a_{2m})^* \end{aligned}$$

▶ **Lemma 19.** *L_1, L_2 are $\mathrm{LTT}[1, d]$-separable for some d, but not $\mathrm{LTT}[1, 2^{2m-1}]$-separable.*

Proof. We prove that L_1, L_2 are $\mathrm{LTT}[1, d]$-separable for $d = 2^{2m-1} + 1$. Consider the following language L. A word w belongs to L if and only if for all odd i either w contains more than $2^{i-1} + 1$ copies of a_i, or the number of copies of a_{i+1} in w is exactly twice the number of copies of a_i in w.

By definition $L \in \mathrm{LTT}[1, 2^{2m-1}]$, we prove that it is a separator. By definition, we have $L \supseteq L_2$. We prove that $L \cap L_1 = \emptyset$. By contradiction, assume that $w \in L \cap L_1$. Since $w \in L_1$, it contains only one copy of a_1. Then, since $w \in L$, it must contain two copies

of a_2. By iterating the argument we get that w must contain 2^{2m-1} copies of a_{2m} which is impossible since this number must be multiple of 3 by definition of L_1. It follows that $L \in \mathrm{LTT}[1, 2^{2m-1}+1]$ is a separator.

It remains to prove that L_1, L_2 are not $\mathrm{LTT}[1, 2^{2m-1}]$-separable. Consider the words

$$\begin{array}{lcl} w_1 = a_1(a_2a_3^2)^2 \cdots (a_{2m-2}a_{2m-1}^2)^{2^{2m-3}}(a_{2m})^{3\times 2^{2m-1}} & \in & L_1 \\ w_2 = a_1a_2^2(a_3a_4^2)^4 \cdots (a_{2m-1}a_{2m}^2)^{2^{2m-2}} & \in & L_2 \end{array}$$

It is clear that $w_1 \equiv_1^{2^{2m-1}} w_2$. Therefore L_1, L_2 are not $\mathrm{LTT}[1, 2^{2m-1}]$-separable. ◀

C Complexity: upper and lower bounds

In this section, we prove the lower and upper complexity bounds stated in Corollaries 4 and 8 for deciding LT- and LTT-separability.

Both the lower and upper bounds rely on the pattern criteria of Theorems 3 and 7. We are able to prove that starting from NFAs or DFAs recognizing the input languages, deciding separability can be achieved in co-NEXPTIME for LT and in 2-EXPSPACE for LTT. Moreover, we prove a co-NP lower bound for both problems.

C.1 Upper Bounds

In this subsection, we prove the complexity upper bounds of Corollaries 4 and 8 for the separation problem for both LT and LTT. The corresponding algorithms rely on the patterns criteria of Theorems 3 and 7. We prove the two following results.

▶ **Proposition 20.** *Let $\mathcal{A}_1, \mathcal{A}_2$ be NFAs. Deciding whether $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ are* LT*-separable can be achieved in* co-NEXPTIME*.*

▶ **Proposition 21.** *Let $\mathcal{A}_1, \mathcal{A}_2$ be NFAs. Deciding whether $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ are* LTT*-separable can be achieved in* 2-EXPSPACE*.*

Both algorithms work by reducing the problems to the special case of $k = 1$, *i.e.*, whether there exists an LT (resp. LTT) separator which considers only 1-profiles. The reduction is identical in both cases and the difference resides in proving that it is correct. These proofs rely on Item 5 in Theorem 3 and Item 4 in Theorem 7 respectively. The computations involved in the reduction can be done in EXPTIME and the new NFAs it outputs are of size exponential in the input NFAs. Therefore, it then suffices to give algorithms for the special case $k = 1$ which run in NP for LT and EXPSPACE for LTT.

Note that a reduction to $k = 1$ could also be done by considering the bound k on the size of profiles in Theorems 3 and 7. Indeed, once k is fixed it suffices to modify the input NFAs to work on the alphabet of k-profiles to be reduced to the case $k = 1$. However, this technique might yield NFAs that are doubly exponential in the size of the input NFAs.

We first present and prove the reduction to the case $k = 1$. Then we explain how to decide both problems in this special case.

C.1.1 Reduction to the case $k = 1$

Let $\mathcal{A}_1 = (Q_1, A_1, I_1, F_1, \delta_1)$ and $\mathcal{A}_2 = (Q_2, A_2, I_2, F_2, \delta_2)$ be NFAs. It follows from Theorem 3 (resp. Theorem 7) that to determine whether $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ are not LT-separable (resp. not LTT-separable), it suffices to verify whether they have a common 1-pattern (resp. whether

there exists d such that they have a common d-pattern). This requires verifying whether there exist a pattern $\mathcal{P}$ and $\mathcal{P}$-decompositions compatible with $\mathcal{A}_1$ and $\mathcal{A}_2$.

Observe that a $\mathcal{P}$-decomposition can be viewed as a word over the alphabet of blocks. The main idea behind the reduction is to construct new NFAs $\tilde{\mathcal{A}}_1$ and $\tilde{\mathcal{A}}_2$ which recognize words over the alphabet of blocks and represent $\mathcal{P}$-decompositions that are compatible with $\mathcal{A}_1$ or $\mathcal{A}_2$ for some $\mathcal{P}$. There are two main issues with this outline. First, the alphabet of blocks is infinite, second, a notion of compatibility has to be enforced between consecutive blocks in a word, *i.e.*, (v, u, v') needs to be followed by (v', u', v'') for some u', v''. Again, this compatibility cannot be simply encoded in the states since there are infinitely many words.

Both issues are solved using a similar argument. Recall that we are only interested in $\mathcal{P}$-decompositions that are compatible with $\mathcal{A}_1$ or $\mathcal{A}_2$. Observe that for a block (v, u, v') to appear in such a decomposition, there need to exist states q, q' with loops on q, q' labeled by v, v' respectively and a path from q to q' labeled by u. We abstract a block by the set of pairs of states verifying this property. Since there are finitely many such sets, this yields a finite alphabet. The same argument can be used for compatibility, all words v that need to be considered for compatibility need to label a loop at some state q. We abstract v by the set of states having such a loop labeled v. The information can then be encoded in the states of $\tilde{\mathcal{A}}_1$ and $\tilde{\mathcal{A}}_2$.

We now describe the formal construction. Let $R \subseteq Q_1 \cup Q_2$. We say that R can be *synchronized* if there exists $v \in A^+$ such that for all $q \in R$ there exists a loop around q labeled by v. We will encode compatibility in synchronizable sets of states.

Similarly, if $T \subseteq (Q_1)^2 \cup (Q_2)^2$ we denote by $\ell(T)$ (resp. $r(T)$) the set of states that are left (resp. right) members of pairs in T. We say that T can be *synchronized* if **a)** there exists $u \in A^*$ such that for all $(q, q') \in T$ there exists a run from q to q' labeled by u, **b)** $\ell(T)$ can be synchronized and **c)** $r(T)$ can be synchronized. In order to deal with prefixes and suffixes, we generalize the notion to these limit cases. We say that T can be *prefix synchronized* (resp. *suffix synchronized*) if **a)** and **c)** (resp. **a)** and **b)**) hold. Finally, we say that T can be *weakly synchronized* if **a)** holds. We set B_w, B_p, B_i and B_s as the sets of weakly synchronizable, prefix synchronizable, synchronizable and suffix synchronizable sets of pairs of states, respectively.

We can now define $\tilde{\mathcal{A}}_1$ and $\tilde{\mathcal{A}}_2$. Set the alphabet B as the disjoint union of B_w, B_p, B_i and B_s. The sets of states $\tilde{Q}_1$ and $\tilde{Q}_2$ are defined as follows.

$$\tilde{Q}_i = \{(r, R) \mid r \in R \subseteq Q_1 \cup Q_2, R \text{ is synchronizable}, r \in Q_i\} \cup I_i \cup F_i.$$

The initial and final states remain I_i and F_i. We now define the transitions. For all $b \in B_w$, we add a b-transition from $q_i \in I_i$ to $r_i \in F_i$ if $(q_i, r_i) \in b$. For all $b \in B_p$, we add a b-transition from $q_i \in I_i$ to (r_i, R_i) if $(q_i, r_i) \in b$ and $R_i = r(b)$. Similarly, for all $b \in B_s$, we add a b-transition from (r_i, R_i) to $q_i \in F_i$ if $(r_i, q_i) \in b$ and $R_i = \ell(b)$. Finally, for all $b \in B_i$, we add a b-transition from (r_i, R_i) to (s_i, S_i) if $(r_i, s_i) \in b$, $R_i = \ell(b)$ and $S_i = r(b)$.

This ends the construction of $\tilde{\mathcal{A}}_1$ and $\tilde{\mathcal{A}}_2$. Observe that these are of size exponential in the size of $\mathcal{A}_1, \mathcal{A}_2$. We now prove that the computation can be done in Exptime.

▶ **Lemma 22.** *Given $\mathcal{A}_1, \mathcal{A}_2$ as input, $\tilde{\mathcal{A}}_1, \tilde{\mathcal{A}}_2$ can be constructed in* Exptime.

Proof. Testing synchronizability of a set of states (resp. a set of pairs of states) can easily be reduced to checking non-empty intersection between a set of NFAs. This is known to be a Pspace-complete problem. It follows that computing the synchronizable sets can be done in Exptime. It is then clear that the remaining computations can be done in Exptime. ◀

We now prove that the construction is correct, *i.e.*, that it reduces LT- and LTT-separability to the special case $k = 1$.

▶ **Lemma 23.** *The following properties hold:*

1. *$L(\mathcal{A}_1), L(\mathcal{A}_2)$ are* LT*-separable if and only if $L(\tilde{\mathcal{A}}_1), L(\tilde{\mathcal{A}}_2)$ are* LT[1]*-separable.*
2. *$L(\mathcal{A}_1), L(\mathcal{A}_2)$ are* LTT*-separable if and only if there exists $d \in \mathbb{N}$ such that $L(\tilde{\mathcal{A}}_1), L(\tilde{\mathcal{A}}_2)$ are* LTT$[1, d]$*-separable.*

Proof. We only give the proof for LTT using Condition 4 from Theorem 7. The proof for LT is obtained similarly using Condition 5 from Theorem 3.

Suppose $L(\mathcal{A}_1), L(\mathcal{A}_2)$ are not LTT-separable. We prove that for all $d \in \mathbb{N}$, $L(\tilde{\mathcal{A}}_1), L(\tilde{\mathcal{A}}_2)$ are not LTT$[1, d]$-separable. Set $d \in \mathbb{N}$, by Condition 4 from Theorem 7, there exists a common d-pattern $\mathcal{P}$: there are two words $w_1 \in L_1, w_2 \in L_2$ such that $w_1 = u_0 v_1 u_1 \dots v_n u_n$, $w_2 = u'_0 v'_1 u'_1 \dots v'_m u'_m$ are $\mathcal{P}$-decompositions compatible with $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. There are two cases. In the first case $n = m = 0$ and $w_1 = w_2$. Therefore, $L_1 \cap L_2 \neq \emptyset$, by definition, it follows that there exist some $b \in B_w$ such that $b \in L(\tilde{\mathcal{A}}_1) \cap L(\tilde{\mathcal{A}}_2)$, which ends the proof.

In the second case $n, m > 0$. Set $\hat{w_1} = \mathfrak{p}\mathfrak{b}_1 \cdots \mathfrak{b}_{n-1}\mathfrak{s}$ such that $\mathfrak{p} = (u_0, v_1)$, $\mathfrak{s} = (u_n, v_n)$ and for all i, $\mathfrak{b}_i = (v_i, u_i, v_{i+1})$. Similarly, we define $\hat{w_2} = \mathfrak{p}'\mathfrak{b}'_1 \cdots \mathfrak{b}'_{m-1}\mathfrak{s}'$. Observe that since we started from $\mathcal{P}$-decompositions for a d-pattern $\mathcal{P}$, we have $\hat{w_1} \equiv_1^d \hat{w_2}$. We use these words to construct $\tilde{w_1}, \tilde{w_2} \in B^*$ accepted by $\tilde{\mathcal{A}}_1, \tilde{\mathcal{A}}_2$.

Let $\mathfrak{b}$ be a block appearing in $\hat{w_1}, \hat{w_2}$, we set $T_\mathfrak{b} \subseteq (Q_1)^2 \cup (Q_2)^2$ as the set of pairs of states that correspond to $\mathfrak{b}$ in the runs on w_1 and w_2. Since the $\mathcal{P}$-decompositions are compatible with their respective NFAs, by definition, $T_\mathfrak{b}$ is synchronizable. Similarly, if $\mathfrak{p}$ (resp. $\mathfrak{s}$) is a prefix (resp. suffix) block occurring in $\hat{w_1}, \hat{w_2}$, we get a prefix (resp. suffix) synchronizable set $T_\mathfrak{p}$ (resp. $T_\mathfrak{s}$). We now set $\tilde{w_1} = T_\mathfrak{p} T_{\mathfrak{b}_1} \cdots T_{\mathfrak{b}_{n-1}} T_\mathfrak{s}$ and $\tilde{w_2} = T_{\mathfrak{p}'} T_{\mathfrak{b}'_1} \cdots T_{\mathfrak{b}'_{m-1}} T_{\mathfrak{s}'}$. By definition of $\tilde{\mathcal{A}}_1, \tilde{\mathcal{A}}_2$ and the fact that the $\mathcal{P}$-decompositions are $\mathcal{A}_1$-, $\mathcal{A}_2$-compatible, $\tilde{w_1}, \tilde{w_2} \in B^*$ are accepted by $\tilde{\mathcal{A}}_1, \tilde{\mathcal{A}}_2$. Moreover, since $\hat{w_1} \equiv_1^d \hat{w_2}$, we have $\tilde{w_1} \equiv_1^d \tilde{w_2}$. We conclude that $L(\tilde{\mathcal{A}}_1), L(\tilde{\mathcal{A}}_2)$ are not LTT$[1, d]$-separable.

Conversely, assume that for all $d \in \mathbb{N}$, $L(\tilde{\mathcal{A}}_1), L(\tilde{\mathcal{A}}_2)$ are not LTT$[1, d]$-separable. We prove that for all $d \in \mathbb{N}$, $\mathcal{A}_1, \mathcal{A}_2$ have a common d-pattern $\mathcal{P}$. Set $d \in \mathbb{N}$, by hypothesis, there exist $\tilde{w_1}, \tilde{w_2} \in B^*$ accepted by $\tilde{\mathcal{A}}_1, \tilde{\mathcal{A}}_2$, such that $\tilde{w_1} \equiv_1^d \tilde{w_2}$. Again, we have two cases, by definition of $\tilde{\mathcal{A}}_1, \tilde{\mathcal{A}}_2$, either $\tilde{w_1}, \tilde{w_2} \in B_w$ or $\tilde{w_1}, \tilde{w_2} \in B_p(B_i)^* B_s$. In the first case this means that $L(\mathcal{A}_1), L(\mathcal{A}_2)$ have non-empty intersection, therefore it suffices to set $\mathcal{P}$ as a word in the intersection to conclude.

Otherwise, $\tilde{w_1} = b_p b_1 \cdots b_n b_s$ and $\tilde{w_2} = b_p b'_1 \cdots b'_m b_s$. By definition of B_p, B_i, B_s, to each label appearing in $\tilde{w_1}, \tilde{w_2}$ we can associate a unique block, prefix block or suffix block. We set $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$ as the d-pattern defined in the following way, $\mathfrak{p}$,$\mathfrak{s}$ are the prefix and suffix blocks associated to b_p and b_s respectively. Since $\tilde{w_1} \equiv_1^d \tilde{w_2}$, for all blocks $\mathfrak{b}$, the number of occurrences of labels b such that $\mathfrak{b}$ is associated to b is the same in $\tilde{w_1}, \tilde{w_2}$ up to threshold d, we set $f(\mathfrak{b})$ as this number. It is now straightforward to verify that $\mathcal{A}_1, \mathcal{A}_2$ admit compatible $\mathcal{P}$-decompositions, which terminates the proof. ◀

C.1.2 Deciding the case $k = 1$

We explain how LT- and LTT-separability can be decided when the size of profiles if fixed to 1. Observe that in this case, the 1-profile of a position is its label. We prove the two following lemmas.

▶ **Lemma 24.** *Let $\mathcal{A}_1, \mathcal{A}_2$ be NFAs. Deciding whether $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ are* LT[1]*-separable is* co-NP.

▶ **Lemma 25.** *Let $\mathcal{A}_1, \mathcal{A}_2$ be NFAs. Deciding whether there exists $d \in \mathbb{N}$ such that $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ are* LTT$[1, d]$*-separable is* Expspace.

As we explained in Section 5, decidability follows from Parikh's Theorem and decidability of Presburger arithmetic. However, applying these results naively yields high complexity. We explain here how to refine the argument in order to get co-Np and Expspace complexities.

Set $\pi(\mathcal{A}_1), \pi(\mathcal{A}_2)$ as the Parikh's images of $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$. As we explained in Section 5, non-LTT-separability is equivalent to the following Presburger property: 'for all $d \in \mathbb{N}$ there exists $\bar{i_1} \in \pi(\mathcal{A}_1)$, $\bar{i_2} \in \pi(\mathcal{A}_2)$ which are equal componentwise up to threshold d.' By [24], existential Presburger formulas for $\pi(\mathcal{A}_1), \pi(\mathcal{A}_2)$ can be computed in polynomial time. Therefore, a Presburger formula for the property can also be computed in polynomial time. Moreover, by definition, this property has exactly one quantifier alternation, it then follows from [5] that it can be decided in Expspace.

The same construction can be done for LT, however, the counting threshold d is fixed to 1. Therefore, the constructed formula is existential. It is known that existential Presburger formuals can be decided in Np (see [24]). We conclude that non LT[1]-separability is in Np.

In the case of LT the problem is actually co-Np-complete. This means that Lemma 24 cannot be improved and that improving Proposition 20 would require improving the reduction. For LTT, the situation is different, it is likely that a sharper analysis of the Presburger formula would yield a better upper bound. Indeed, while deciding Presburger formulas with only one quantifier alternation is already very costly in general, the formula we consider is very specific.

C.2 Lower Bounds

In this subsection, we prove co-Np lower bounds for both LT and LTT-separability, stated in Corollaries 4 and 8. The bounds hold when the input languages are given as NFAs or DFAs.

▶ **Proposition 26.** *Let $\mathcal{A}_1, \mathcal{A}_2$ be DFAs, the two following problems are* co-Np*-hard:*

1. *Are $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$,* LT*-separable?*
2. *Are $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$,* LTT*-separable?*

Proof. We only do the proof for LTT. The reduction is identical for the other case and only minor adjustments in the correctness proof are needed.

We prove that testing if $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ are not LTT-separable is Np-hard. The proof is by reduction of **3-SAT**. From an instance of **3-SAT**, we construct two DFAs and prove that the corresponding languages are not LTT-separable if and only if the **3-SAT** instance is satisfiable.

Let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be a set of 3-clauses over the set of variables $\{x_1, \ldots, x_n\}$. We construct DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ over the alphabet $A := \{\#, @, x_1, \ldots x_n, \neg x_1, \ldots, \neg x_n\}$. Given $w \in A^*$, we say that w *encodes an assignment of truth values* if for all $i \leqslant n$, w contains either the label x_i or the label $\neg x_i$, but not both. It is straightforward to see that an assignment of truth values for the variables can be uniquely defined from such a word. Moreover, by definition of LTT we have the following fact.

▶ **Fact 27.** *The language of correct assignments is* LTT*.*

Intuitively, we want our DFAs $\mathcal{A}_1, \mathcal{A}_2$ to verify the following three properties:

1. all words accepted by $\mathcal{A}_1$ encode assignments, and for each assignment of variables, a word coding that assignment is accepted by $\mathcal{A}_1$.

2. all assignments accepted by $\mathcal{A}_2$ satisfy $\mathcal{C}$, and for each assignment of variables satisfying $\mathcal{C}$, a word coding that assignment is accepted by $\mathcal{A}_2$.
3. if there exist $w_1, w_2 \in A^*$ that are accepted by $\mathcal{A}_1, \mathcal{A}_2$ and encode the same assignment of variables, then for all $k, d \in \mathbb{N}$, w_1, w_2 can be chosen such that $w_1 \equiv_k^d w_2$.

Conditions 1 and 2 are simple to enforce. Indeed, for Condition 1, it suffices to construct a DFA recognizing the words $t_1 \dots t_n$ such that for all $i \leqslant n$, $t_i = x_i$ or $t_i = \neg x_i$. For Condition 2 it suffices to construct a DFA recognizing the words $t_1 \dots t_m u$ where for all $i \leqslant m$, t_i is a literal of C_i and u is an arbitrary word (u is necessary since some variables might not appear in the prefix, preventing the word from coding an assignment). The problem with these two template DFAs is that they do not verify Condition 3. To solve this issue, we add loops on their states in order to generate as many copies of infixes as necessary to make two words coding the same assignment indistinguishable by LTT.

We begin by giving $\mathcal{A}_1$, of which a graphical representation is shown in Figure 4.

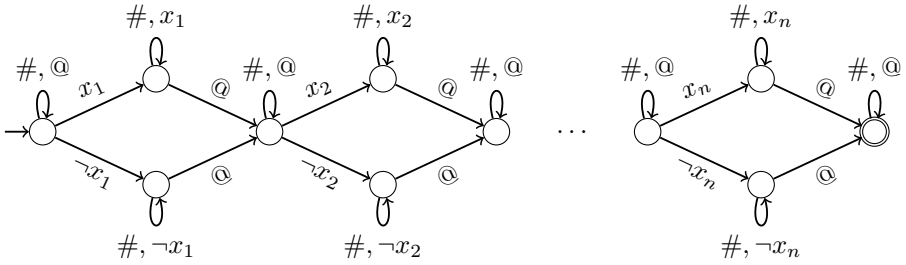

Figure 4 Representation of $\mathcal{A}_1$

By definition, $\mathcal{A}_1$ verifies Item 1. We now define $\mathcal{A}_2$ as a sequence of m subautomata, each one corresponding to a clause C in $\mathcal{C}$. Intuitively, if $C = x_i \vee x_j \vee \neg x_k$, the subautomata selects a label within $\{x_i, x_j, \neg x_k\}$. This means that if this word encodes an assignment, it must satisfy all clauses in $\mathcal{C}$. We give a graphical representation of $\mathcal{A}_2$ in Figure 5.

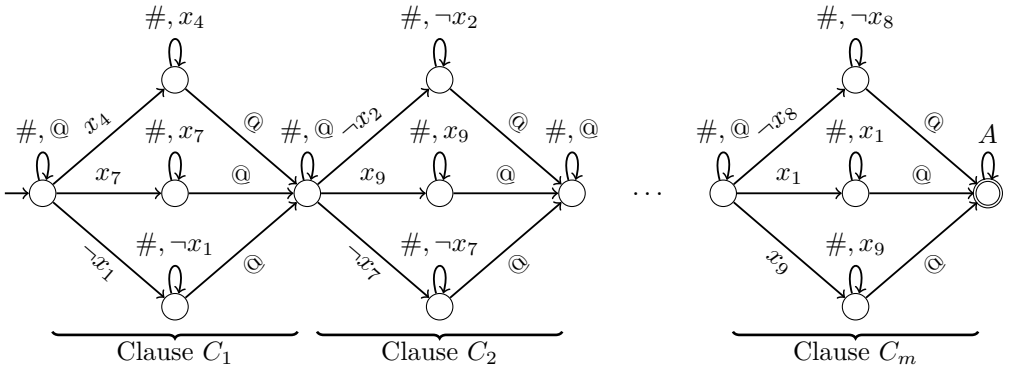

Figure 5 Representation of $\mathcal{A}_2$, for $C_1 = x_4 \vee x_7 \vee \neg x_1$, $C_2 = \neg x_2 \vee x_9 \vee \neg x_7$, $C_m = \neg x_8 \vee x_1 \vee x_9$

By definition, $\mathcal{A}_2$ verifies Item 2. It remains to verify that Item 3 holds.

▶ **Lemma 28.** *If there are words w_1, w_2 accepted by $\mathcal{A}_1, \mathcal{A}_2$ that encode the same assignment, then for all $k, d \in \mathbb{N}$, they can be chosen such that $w_1 \equiv_k^d w_2$.*

Proof. This is done by using the loops in $\mathcal{A}_1, \mathcal{A}_2$ to generate as many copies of the k-profiles in w_1, w_2 as needed in order to get words that are $\equiv_k^d$-equivalent. ◀

We finish by proving that $\mathcal{C}$ is satisfiable if and only if $L(\mathcal{A}_1), L(\mathcal{A}_2)$ are not LTT-separable. Assume first that $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ are not LTT-separable. By Fact 27, this means that there exist $w_1 \in L(\mathcal{A}_1)$ and $w_2 \in L(\mathcal{A}_2)$ sharing the same alphabet. By Item 1, w_1 encodes an assignment. Therefore, w_2 (which has the same alphabet) encodes the same assignment which satisfies $\mathcal{C}$ by Item 2. Hence $\mathcal{C}$ is satisfiable.

Conversely, assume that $\mathcal{C}$ is satisfiable. We prove that for all $k, d \in \mathbb{N}$, $L(\mathcal{A}_1), L(\mathcal{A}_2)$ are not LTT$[k, d]$-separable. Set $k, d \in \mathbb{N}$ and consider an assignment of truth values satisfying $\mathcal{C}$. By Items 1 and 2, there must exist w_1, w_2 accepted by $\mathcal{A}_1, \mathcal{A}_2$ that both encode this assignment. It follows from Lemma 28 that w_1, w_2 can be chosen such that $w_1 \equiv_k^d w_2$. Therefore, $L(\mathcal{A}_1), L(\mathcal{A}_2)$ are not LTT$[k, d]$-separable, which terminates the proof. ◀

D Appendix to Section 6: separation of context-free languages

In this appendix, we prove Theorem 14.

▶ **Theorem 14.** *Let L_1, L_2 be context-free languages. It is undecidable to test whether L_1, L_2 are* LT*-separable. It is undecidable to test whether L_1, L_2 are* LTT*-separable.*

Proof. The proof is done by reduction of the halting problem on Turing machines to LT-separability and LTT-separability. The reduction is the same for both LT and LTT and is essentially a rewriting of a proof of [21].

Consider a deterministic Turing machine M. We prove that it is possible to compute context-free languages L_1, L_2 from M such that M halts on empty input if and only if L_1, L_2 are LT-separable, if and only if L_1, L_2 are LTT-separable.

Let A be the alphabet of M, let Q be its set of states, and let $B = A \cup (A \times Q) \cup \{\#, \gamma\}$ where $\#, \gamma \notin A$. It is clear that any possible configuration of M can be encoded as a word in $A^* \cdot (A \times Q) \cdot A^* \subseteq B^*$. Finally, if $w \in B^*$, we denote by w^R the mirror image of w. We can now define context-free languages L_1, L_2 over B. The language L_1 contains all words of the form:

$$c_1 \# c_2^R \# c_3 \# c_4^R \cdots \# c_{2k-1} \# c_{2k}^R \# \gamma^k$$

such that $c_1, \ldots, c_{2k}$ are encodings of configurations of M, and for all $i \leqslant k$, $c_{2i-1} \vdash_M c_{2i}$ (*i.e.*, c_{2i} is the configuration of M reached after one computation step from configuration c_{2i-1}). Similarly, L_2 contains all words of the form:

$$c_1 \# c_2^R \# c_3 \# c_4^R \cdots \# c_{2k-1} \# c_{2k}^R \# \gamma^{2k}$$

such that $c_1, \ldots, c_{2k}$ are encodings of configurations of M, c_1 is the intial configuration of M starting with an empty input and for all $i \leqslant k-1$, $c_{2i} \vdash_M c_{2i+1}$. It can be shown using classical arguments for context-free languages that L_1, L_2 are indeed context-free and that grammars for L_1, L_2 can be computed from M. We now make a simple observation about prefixes that are common to both languages. Let $c_1, c_2, c_3, c_4, \cdots, c_{i-1}, c_i$ be a sequence of configurations and w be the word

$$\begin{array}{rcll} w & = & c_1 \# c_2^R \# c_3 \# c_4^R \cdots \# c_{2k-1} \# c_{2k}^R \# \in B^* & (\text{if } i = 2k) \\ w & = & c_1 \# c_2^R \# c_3 \# c_4^R \cdots \# c_{2k}^R \# c_{2k+1} \# \in B^* & (\text{if } i = 2k+1) \end{array}$$

By definition of L_1, L_2, we have the following fact:

▶ **Fact 29.** *If w is both a prefix of some word in L_1 and some word in L_2, then $c_1, c_2, \ldots, c_i$ are the first i configurations of the run of M starting from the empty input. Moreover, if $i = 2k$ and*

$$c_1\#c_2^R\#c_3\#c_4^R\cdots\#c_{2k-1}\#c_{2k}^R\#c\#$$

is a prefix of a word in L_2, then c is configuration $i+1$ in the run. Symetrically, if $i = 2k+1$ and

$$c_1\#c_2^R\#c_3\#c_4^R\cdots\#c_{2k}^R\#c_{2k+1}\#c\#$$

is a prefix of a word in L_1, then c is configuration $i+1$ in the run.

It remains to prove that this is indeed a reduction, *i.e.*, that M halts on empty input if and only if L_1, L_2 are LT-separable if and only if L_1, L_2 are LTT-separable. Assume first that M does not halt on empty input. This means that the run of M is an infinite sequence of configurations $c_1, c_2, c_3, \ldots$. By definition of L_1, L_2, for all $k \in \mathbb{N}$:

$$\begin{array}{ll} c_1\#c_2^R\#c_3\#c_4^R\cdots\#c_{2k-1}\#c_{2k}^R\#\gamma^k & \in L_1 \\ c_1\#c_2^R\#c_3\#c_4^R\cdots\#c_{2k-1}\#c_{2k}^R\#\gamma^{2k} & \in L_2 \end{array}$$

It is then easy to deduce that L_1, L_2 cannot be separated by a LT or LTT language (actually not even by a regular language).

Conversely, assume that M halts on empty input and let l be the number of steps it takes M to halt. We define an LT language L such that $L_1 \subseteq L$ and $L \cap L_2 = \emptyset$. The definition of L is a consequence of the following lemma:

▶ **Lemma 30.** *Let $w_1, w_2 \in L_1, L_2$ of length greater than $(l+1)^2 + 2(l+1)$ and u_1, u_2 the prefixes of length $l(l+1)+2l+1$ of w_1, w_2. Then $u_1 \neq u_2$.*

Proof. We proceed by contradiction. Assume $u_1 = u_2$ and let u be the largest prefix of $u_1 = u_2$ of the form

$$\begin{array}{c} u = c_1\#c_2^R\#c_3\#c_4^R\cdots\#c_{i-1}\#c_i^R\# \\ \textit{or} \\ u = c_1\#c_2^R\#c_3\#c_4^R\cdots\#c_{i-1}^R\#c_i\# \end{array}$$

By definition $u_1 = u_2 = u \cdot v$, where v can be of the form γ^j with $j \leqslant i$ or is the prefix of c or c^R for some configuration c of M.

Assume first that $v = \gamma^j$ with $j \leqslant i$. By Fact 29, $c_1, c_2, c_3, c_4, \cdots, c_{i-1}, c_i$ are the first i configurations of the run of M starting from the empty input. Since M halts in l steps, this means that $i \leqslant l$ and that each configuration c_i is of length at most $\leqslant l+1$. It follows that u is of length at most $l(l+1)+l$. Therefore, $u_1 = u_2$ is of length at most $l(l+1)+l+j \leqslant l(l+1)+2l$ which is a contradiction.

If v is the prefix of c or c^R for some configuration c of M. By Fact 29, c is so that $c_1, c_2, c_3, c_4, \cdots, c_{i-1}, c_i, c$ are the first $i+1$ configurations of the run of M starting from the empty input. Since M halts in l steps, this means that $i+1 \leqslant l$ and that each configuration is of length at most $\leqslant l+1$. It follows that $u_1 = u_2$ is of length at most $l(l+1)+l$ which is again a contradiction. ◀

Set K_1 be the language of words of L_1 of length less than $l(l+1)+2l+1$. Similarly let K_2 be the set of prefixes of length $l(l+1)+2l+1$ of words in L_1. Finally, set $L = K_1 \cup K_2 \cdot B^*$. By definition, K_1, K_2 are finite languages, hence L is clearly LT (and therefore LTT). It remains to prove that L is a separator.

Observe that by definition, we have $L_1 \subseteq L$. We prove that $L \cap L_2 = \emptyset$. We proceed by contradiction, assume that there exists $w \in L \cap L_2$. If $w \in K_1$, then there is a contradiction because w must be in of the form

$$c_1 \# c_2^R \# c_3 \# c_4^R \cdots \# c_{2k-1} \# c_{2k}^R \# \gamma^k$$

and being in L_2 requires twice as many letters γ at the end of the word. If $w \in K_2 \cdot B^*$, then, by definition of K_2 there exists some word of length $l(l+1)+2l+1$ that is both a prefix of a word in L_1 and a prefix of a word in L_2. This contradicts Lemma 30.

We deduce that L_1, L_2 are LT-separable, and therefore also LTT-separable, which concludes the proof of Theorem 14. ◀