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.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.3 Formal Languages

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

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.363

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 [26].

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

[*] Work supported by Agence Nationale de la Recherche ANR 2010 BLAN 0202 01 FREC.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 363–375
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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 $\mathcal{C}$ is 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 this from the language 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 [5].

Although the separation problem frequently occurs, it has 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. Indeed, the question was raised and solved for separation by piecewise-testable languages [5, 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 widely studied class, 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 [27, 13] and by Brzozowski and Simon [4]. This class has several generalizations. The most studied one is that of *locally threshold testable languages (LTT)*, where counting infixes is allowed up to some threshold. These are the languages definable in FO(+1), *i.e.*, first-order logic with the successor relation (but without the order). Again, membership is decidable [24], 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, we discuss the separation problem starting from context-free input languages rather than regular ones. Due to lack of space, several proofs only appear in the journal version of the paper.

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. In the context of semigroup theory, it has been proven [1] that the separation problem can be rephrased in purely algebraic terms. Solving the separation problem for a class $\mathcal{S}$ amounts to computing the so-called *pointlike sets* for the algebraic variety corresponding to $\mathcal{S}$. While it has been shown that the varieties corresponding to both LT and LTT have computable pointlike sets [2, 20, 21], this approach suffers two drawbacks. First, being purely algebraic, the proofs provide no insight on the underlying class $\mathcal{S}$. In particular, they provide only yes/no answers without giving any description of what an actual separator might be.

Finally, the separation problem for the class of piecewise-testable languages has recently been shown Ptime-decidable, independently and with different techniques in [5] and [18].

2 Preliminaries

Words and Languages. We fix a finite alphabet A. 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: $w_\ell = w[\max(0, x - k_\ell), x]$ and $w_r = w[x, \min(x + k_r, |w|)]$ (see Figure 1). 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 $n \in \mathbb{N}$, 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.

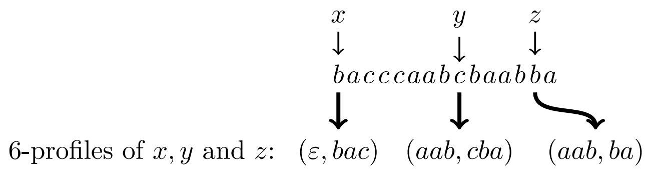

Figure 1 Illustration of the notion of k-profile for $k = 6$.

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 [2, 25], *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'. One can verify that for all $k, d \in \mathbb{N}$, $\equiv_k^d$ is an equivalence relation of finite index.

For $k, d \in \mathbb{N}$ we denote by $\mathrm{LTT}[k, d]$ the set of the finitely many 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 corresponds to $d = 1$. All results in this section are for an arbitrary d. Our result is twofold.

- First, we establish 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 $\mathrm{LTT}[k, d]$ languages, where k denotes this bound.
- The second contribution is a criterion on the input languages to check separability by an $\mathrm{LTT}[k, d]$ language for some k. This criterion can be defined equivalently on automata or monoids recognizing the input languages, in terms of the absence of common patterns.

The section is organized into three subsections: our criterion is stated in the first one, and the second and last ones are devoted to the statement and proof of the theorem.

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.

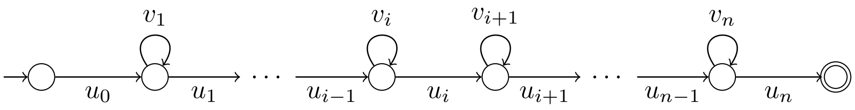

Figure 2 An $\mathcal{A}$-compatible $\mathcal{P}$-decomposition $u_0v_1u_1v_2\cdots v_nu_n$.

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 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 $\{0, \ldots, d\}$, 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 $w = u_0v_1u_1v_2\cdots v_nu_n$ 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 $(v_i, u_i, v_{i+1}) = \mathfrak{b}$.

Sometimes, we just say $\mathcal{P}$-decomposition to mean $\mathcal{P}$-decomposition *of some word*. Let $\alpha : A^* \to M$ be a morphism into a monoid M, and let $s \in M$. A $\mathcal{P}$-decomposition is (α, 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, for $1 \leqslant i \leqslant n$, as pictured in Figure 2 (where edges denote sequences of transitions).

Common Patterns. Let $d \in \mathbb{N}$ and M_1, M_2 be two monoids together with morphisms $\alpha_1 : A^* \to M_1$ and $\alpha_2 : A^* \to M_2$ 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 $\mathcal{P}$, two elements $s_1 \in F_1$, $s_2 \in F_2$, and two $\mathcal{P}$-decompositions of (possibly different) words that are respectively (α_1, s_1)-compatible and (α_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 $\mathcal{P}$ and two $\mathcal{P}$-decompositions of words that are respectively $\mathcal{A}_1$-compatible and $\mathcal{A}_2$-compatible. In particular, by the very definition, $\mathcal{A}_1$ and $\mathcal{A}_2$ have a common 1-pattern if there are successful paths in $\mathcal{A}_1, \mathcal{A}_2$ of the form shown in Figure 2 with the same *set* of triples (v_i, u_i, v_{i+1}).

A useful property about common patterns is that whether such a pattern exists 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.*

4.2 Separation Theorem for a Fixed Threshold

We can now state our main theorem for this section.

▶ **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 3b is essentially a *delay theorem* [22] 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. It turns out that a better algorithm can be obtained from Items 3d and 3e (the proof is available in the full version of the paper). 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 (3c) ⇒ (3b) ⇒ (3a) are immediate by definition. We now prove (3a) ⇒ (3e) ⇒ (3d) ⇒ (3c). The implication (3e) ⇒ (3d) is immediate from Proposition 1. The implication (3a) ⇒ (3e) 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 $\mathrm{LTT}[k,d]$ language for any k. This is exactly the contrapositive of (3a) ⇒ (3e). We now prove Proposition 5.

Proof of Prop. 5. 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

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

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 $w_1 \equiv_k^d w_2$. ◀

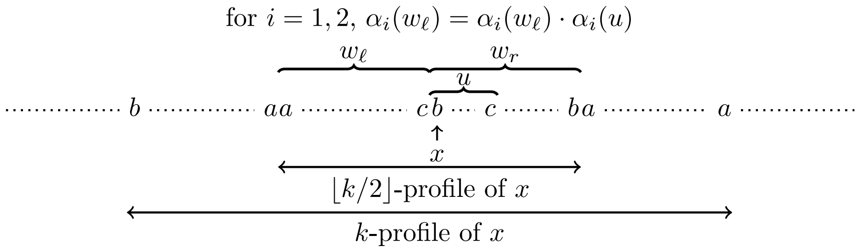
for $i = 1, 2$, $\alpha_i(w_\ell) = \alpha_i(w_\ell) \cdot \alpha_i(u)$
w_ℓ
w_r
u
b
aa
cb
c
ba
a
x
$\lfloor k/2 \rfloor$-profile of x
k-profile of x

Figure 3 A position x admitting a k-loop u: $\alpha_i(w_\ell) = \alpha_i(w_\ell \cdot u)$, for $i = 1, 2$.

The remaining and most difficult direction, (3d) ⇒ (3c) is a consequence of the next proposition whose proof is outlined in the next subsection.

▶ **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 $(\alpha_1, \alpha_1(w_1))$-compatible $\mathcal{P}$-decomposition, and an $(\alpha_2, \alpha_2(w_2))$-compatible $\mathcal{P}$-decomposition.*

Before explaining how to show Proposition 6, let us explain how to conclude the proof of Theorem 2. We prove the contrapositive of (3d) ⇒ (3c). If Item 3c 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.

4.3 Proof of Proposition 6

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 admitting respectively a $(\alpha_1, \alpha_1(w_1))$-compatible $\mathcal{P}$-decomposition and a $(\alpha_2, \alpha_2(w_2))$-compatible $\mathcal{P}$-decomposition, for some d-pattern $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$. In this outline, we only provide the construction, the proof of correctness is available in the full version of the paper.

The construction of w_1', w_2', 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 Figure 3.

For our construction to work, we need k-loops to have three properties that we state now. The first two are simple facts that are immediate from the definition: k-loops are determined by profiles and can be duplicated without modifying the image of the word under α.

▶ **Fact 7.** *Let x be a position. Whether x admits a k-loop, and if so, which k-loop x admits, only depends on the $\lfloor k/2 \rfloor$-profile of x.*

▶ **Fact 8.** *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 k-loops occur frequently in words, *i.e.*, at least one of $\lfloor k/4 \rfloor$ consecutive positions must admit a k-loop. This follows from pumping arguments:

▶ **Lemma 9.** *Let w be a word and let $x_1, \dots, x_{\lfloor k/4 \rfloor}$ 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.*

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 $(u_x)^{k'}$ in w_1 (resp. w_2) at any position x that admits a k-loop, and where u_x is the k-loop of x. Using Fact 7, Fact 8 and Lemma 9 one can then verify that w'_1 admits a $(\alpha_1, \alpha_1(w_1))$-compatible $\mathcal{P}$-decomposition and w'_2 admits a $(\alpha_2, \alpha_2(w_2))$-compatible $\mathcal{P}$-decomposition, for some d-pattern $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$. The proof is available in the full version of the paper.

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 [3] 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 obtained 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 d is 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 10.** *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 3e by using the usual brute-force algorithm. As it was the case for a fixed counting threshold, this algorithm is slow. In the full version of the paper, we obtain a faster algorithm by using Items 3c and 3d.

▶ **Corollary 11.** *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* LTT*-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 3a, 3b, 3c and 3d is an immediate consequence of Theorem 2.

Therefore, we only need to prove Items 3e and 3f, *i.e.*, the bound on the threshold d. Unfortunately, these are exactly the items we need for Corollary 11. However, we will prove that by reusing an algorithm of [3], Corollary 11 can also be derived directly from Item 3b.

We now explain how to derive the first part of Corollary 11 from Item 3b without relying on the actual bound on the counting threshold. The bound itself is proved in the full version.

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

As we explained, the equivalence of Item 3b to LTT-separability is immediate from Theorem 2. We explain how to combine this fact with an algorithm of [3] to obtain decidability directly.

In [3], 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 [3], 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 3b. 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 commutative image, which we now recall.

The *commutative image* of a word $w \in A^*$, denoted $\pi(w)$, is a vector of length $|A|$ of natural numbers counting, for all $a \in A$, 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 $\{\pi_k(w) \mid w \in L\} \subseteq \mathbb{N}^{A_k}$. The definition of $\equiv_k^d$ yields the following fact.

► **Fact 12.** *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 commutative images is Parikh's Theorem [15], which states that if L is context-free (so in particular if L is regular), then $\pi(L)$ is semilinear, *i.e.*, Presburger definable [6]. As explained in [3], Parikh's Theorem extends without difficulty to k-images.

► **Theorem 13.** *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. For $k = 1$, this is Parikh's Theorem. When $k > 1$, let $L' \subseteq A_k^*$ such that $w' \in L'$ if there exists $w \in L$ of the same length, and such that a position in w' is labeled by the k-profile of the same position in w. One can verify that L' is (effectively) context-free, and that the k-image of L is the commutative image of L', which is semilinear by Parikh's Theorem. ◄

We can now explain how to decide LTT-separability. By Item 3b in Theorem 10, 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 number 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 12, this can be expressed in terms of k-images: does there exist a threshold d 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 13 that the above question can be expressed as a computable Presburger formula. Decidability of LTT-separability then follows from decidability of Presburger Arithmetic.

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 3b in Theorem 10. 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 profiles 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 14.** *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 14 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 15.** *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 $L = A^*$ iff $L_1 \in \mathrm{LTT}[1]$.

If $L = A^*$, 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. ◀

Theorems 14 and 15 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 is undecidable. Such a result would require to generalize our third ingredient, Item 3b in Theorem 10, 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 16.** *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 [23] that separability by a regular language is undecidable for context-free languages. The proof of Theorem 16 is essentially the same since the reduction provided in [23] 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 16 follows.

7 Conclusion

We proved separation theorems for both LT and LTT. In both cases, algorithms to test separability, in co-Nexptime and 2-Expspace respectively, are derived from these theorems. 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. 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 D. Beauquier and J. E. Pin. Languages and scanners. *Theor. Comp. Sci.*, 84(1):3–21, 1991.

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

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

5 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.

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. On pointlike sets and joins of pseudovarieties. *IJAC*, 8(2), 1998.

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

22 H. Straubing. Finite semigroup varieties of the form $V * D$. *J. Pure Appl. Algebra*, 36, 1985.

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

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

25 W. Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, 25(3):360–376, 1982.

26 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.

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