The Complexity of Separation for Levels in Concatenation Hierarchies

Thomas Place
LaBRI Bordeaux University and IUF, France

Marc Zeitoun
LaBRI Bordeaux University, France

Abstract

We investigate the complexity of the separation problem associated to classes of regular languages. For a class $\mathcal{C}$, $\mathcal{C}$-separation takes two regular languages as input and asks whether there exists a third language in $\mathcal{C}$ which includes the first and is disjoint from the second. First, in contrast with the situation for the classical membership problem, we prove that for most classes $\mathcal{C}$, the complexity of $\mathcal{C}$-separation does not depend on how the input languages are represented: it is the same for nondeterministic finite automata and monoid morphisms. Then, we investigate specific classes belonging to finitely based concatenation hierarchies. It was recently proved that the problem is always decidable for levels 1/2 and 1 of any such hierarchy (with inefficient algorithms). Here, we build on these results to show that when the alphabet is fixed, there are polynomial time algorithms for both levels. Finally, we investigate levels 3/2 and 2 of the famous Straubing-Thérien hierarchy. We show that separation is PSpace-complete for level 3/2 and between PSpace-hard and EXPTime for level 2.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Regular languages, separation, concatenation hierarchies, complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.47

Related Version A full version of the paper is available at https://arxiv.org/abs/1810.09287.

Funding Both authors acknowledge support from the DeLTA project (ANR-16-CE40-0007).

1 Introduction

For more than 50 years, a significant research effort in theoretical computer science was made to solve the membership problem for regular languages. This problem consists in determining whether a class of regular languages is decidable, that is, whether there is an algorithm inputing a regular language and outputing 'yes' if the language belongs to the investigated class, and 'no' otherwise.

Many results were obtained in a long and fruitful line of research. The most prominent one is certainly Schützenberger's theorem [19], which gives such an algorithm for the class of star-free languages. For most interesting classes also, we know precisely the computational cost of the membership problem. As can be expected, this cost depends on the way the input language is given. Indeed, there are several ways to input a regular language. For instance, it can be given by a nondeterministic finite automaton (NFA), or, alternately, by a morphism into a finite monoid. While obtaining an NFA representation from a morphism into a monoid has only a linear cost, the converse direction is much more expensive: from an NFA with n states, the smallest monoid recognizing the same language may have an exponential number of elements (the standard construction yields 2^{n^2} elements). This explains why the complexity of the membership problem depends on the representation of the input. For instance, for the class of star-free languages, it is PSpace-complete if one starts from NFAs (and actually, even from DFAs [2]) while it is NL when starting from monoid morphisms.

Recently, another problem, called separation, has replaced membership as the cornerstone in the investigation of regular languages. It takes as input *two* regular langages instead of one, and asks whether there exists a third language from the class under investigation including the first input language and having empty intersection with the second one. This problem has served recently as a major ingredient in the resolution of difficult membership problems, such as the so-called dot-depth two problem [16] which remained open for 40 years (see [13, 18, 6] for recent surveys on the topic). Dot-depth two is a class belonging to a famous *concatenation hierarchy* which stratifies the star-free languages: the dot-depth [1]. A specific concatenation hierarchy is built in a generic way. One starts from a base class (level 0 of the hierarchy) and builds increasingly growing classes (called levels and denoted by 1/2, 1, 3/2, 2, . . .) by alternating two standard closure operations: polynomial and Boolean closure. Concatenation hierarchies account for a significant part of the open questions in this research area. The state of the art regarding separation is captured by only three results [17, 9]: in finitely based concatenation hierarchies (i.e. those whose basis is a finite class) levels 1/2, 1 and 3/2 have decidable separation. Moreover, using specific transfer results [15], this can be pushed to the levels 3/2 and 2 for the two most famous finitely based hierarchies: the dot-depth [1] and the Straubing-Thérien hierarchy [21, 22].

Unlike the situation for membership and despite these recent decidability results for separability in concatenation hierarchies, the complexity of the problems and of the corresponding algorithms has not been investigated so far (except for the class of piecewise testable languages [3, 11, 5], which is level 1 in the Straubing-Thérien hierarchy). The aim of this paper is to establish such complexity results. Our contributions are the following:

- We present a **generic** reduction, which shows that for many natural classes, the way the input is given (by NFAs or finite monoids) has **no impact** on the complexity of the separation problem. This is proved using two LogSpace reductions from one problem to the other. This situation is surprising and opposite to that of the membership problem, where an exponential blow-up is unavoidable when going from NFAs to monoids.
- Building on the results of [17], we show that when the alphabet is fixed, there are polynomial time algorithms for levels 1/2 and 1 in any finitely based hierarchy.
- We investigate levels 3/2 and 2 of the famous Straubing-Thérien hierarchy, and we show that separation is PSpace-complete for level 3/2 and between PSpace-hard and EXPTime for level 2. The upper bounds are based on the results of [17] while the lower bounds are based on independent reductions.

Organization. In Section 2, we give preliminary terminology on the objects investigated in the paper. Sections 3, 4 and 5 are then devoted to the three above points. Due to space limitations, many proofs are postponed to the full version of the paper.

2 Preliminaries

In this section, we present the key objects of this paper. We define words and regular languages, classes of languages, the separation problem and finally, concatenation hierarchies.

2.1 Words and regular languages

An alphabet is a *finite* set A of symbols, called *letters*. Given some alphabet A, we denote by A^+ the set of all nonempty finite words and by A^* the set of all finite words over A (*i.e.*, $A^* = A^+ \cup \{\varepsilon\}$). If $u \in A^*$ and $v \in A^*$ we write $u \cdot v \in A^*$ or $uv \in A^*$ for the concatenation of u and v. A *language* over an alphabet A is a subset of A^*. Abusing terminology, if $u \in A^*$ is some word, we denote by u the singleton language $\{u\}$. It is standard to extend concatenation to languages: given $K, L \subseteq A^*$, we write $KL = \{uv \mid u \in K \text{ and } v \in L\}$. Moreover, we also consider marked concatenation, which is less standard. Given $K, L \subseteq A^*$, *a marked concatenation* of K with L is a language of the form KaL, for some $a \in A$.

We consider *regular languages*, which can be equivalently defined by *regular expressions*, *nondeterministic finite automata* (NFAs), *finite monoids* or *monadic second-order logic* (MSO). In the paper, we investigate the separation problem which takes regular languages as input. Since we are focused on complexity, how we represent these languages in our inputs matters. We shall consider two kinds of representations: NFAs and monoids. Let us briefly recall these objects and fix the terminology (we refer the reader to [7] for details).

NFAs. An NFA is a tuple $\mathcal{A} = (A, Q, \delta, I, F)$ where A is an alphabet, Q a finite set of states, $\delta \subseteq Q \times A \times Q$ a set of transitions, $I \subseteq Q$ a set of initial states and $F \subseteq Q$ a set of final states. The language $L(\mathcal{A}) \subseteq A^*$ consists of all words labeling a run from an initial state to a final state. The regular languages are exactly those which are recognized by an NFA. Finally, we write "DFA" for *deterministic* finite automata, which are defined in the standard way.

Monoids. We turn to the algebraic definition of regular languages. A *monoid* is a set M endowed with an associative multiplication $(s, t) \mapsto s \cdot t$ (also denoted by st) having a neutral element 1_M, *i.e.*, such that $1_M \cdot s = s \cdot 1_M = s$ for every $s \in M$. An *idempotent* of a monoid M is an element $e \in M$ such that $ee = e$.

Observe that A^* is a monoid whose multiplication is concatenation (the neutral element is ε). Thus, we may consider monoid morphisms $\alpha : A^* \to M$ where M is an arbitrary monoid. Given such a morphism, we say that a language $L \subseteq A^*$ is *recognized* by α when there exists a set $F \subseteq M$ such that $L = \alpha^{-1}(F)$. It is well-known that the regular languages are also those which are recognized by a morphism into a *finite* monoid. When representing a regular language L by a morphism into a finite monoid, one needs to give both the morphism $\alpha : A^* \to M$ (*i.e.*, the image of each letter) and the set $F \subseteq M$ such that $L = \alpha^{-1}(F)$.

2.2 Classes of languages and separation

A class of languages $\mathcal{C}$ is a correspondence $A \mapsto \mathcal{C}(A)$ which, to an alphabet A, associates a set of languages $\mathcal{C}(A)$ over A.

▶ Remark. When two alphabets A, B satisfy $A \subseteq B$, the definition of classes does not require $\mathcal{C}(A)$ and $\mathcal{C}(B)$ to be comparable. In fact, it may happen that a particular language $L \subseteq A^* \subseteq B^*$ belongs to $\mathcal{C}(A)$ but not to $\mathcal{C}(B)$ (or the opposite). For example, we may consider the class $\mathcal{C}$ defined by $\mathcal{C}(A) = \{\emptyset, A^*\}$ for every alphabet A. When $A \subsetneq B$, we have $A^* \in \mathcal{C}(A)$ while $A^* \notin \mathcal{C}(B)$.

We say that $\mathcal{C}$ is a *lattice* when for every alphabet A, we have $\emptyset, A^* \in \mathcal{C}(A)$ and $\mathcal{C}(A)$ is closed under finite union and finite intersection: for any $K, L \in \mathcal{C}(A)$, we have $K \cup L \in \mathcal{C}(A)$ and $K \cap L \in \mathcal{C}(A)$. Moreover, a *Boolean algebra* is a lattice $\mathcal{C}$ which is additionally closed under complement: for any $L \in \mathcal{C}(A)$, we have $A^* \setminus L \in \mathcal{C}(A)$. Finally, a class $\mathcal{C}$ is *quotienting* if it is closed under quotients. That is, for every alphabet A, $L \in \mathcal{C}(A)$ and word $u \in A^*$, the following properties hold:

$$u^{-1}L \stackrel{\text{def}}{=} \{w \in A^* \mid uw \in L\} \quad \text{and} \quad Lu^{-1} \stackrel{\text{def}}{=} \{w \in A^* \mid wu \in L\} \quad \text{both belong to } \mathcal{C}(A).$$

All classes that we consider in the paper are (at least) quotienting lattices consisting of *regular languages*. Moreover, some of them satisfy an additional property called *closure under inverse image*.

Recall that A^* is a monoid for any alphabet A. We say that a class $\mathcal{C}$ is *closed under inverse image* if for every two alphabets A, B, every monoid morphism $\alpha : A^* \to B^*$ and every language $L \in \mathcal{C}(B)$, we have $\alpha^{-1}(L) \in \mathcal{C}(A)$. A quotienting lattice (resp. quotienting Boolean algebra) closed under inverse image is called a *positive variety* (resp. *variety*).

Separation. Consider a class of languages $\mathcal{C}$. Given an alphabet A and two languages $L_1, L_2 \subseteq A^*$, we say that L_1 is $\mathcal{C}$-separable from L_2 when there exists a third language $K \in \mathcal{C}(A)$ such that $L_1 \subseteq K$ and $L_2 \cap K = \emptyset$. In particular, K is called a *separator* in $\mathcal{C}$. The $\mathcal{C}$-separation problem is now defined as follows:

Input:	An alphabet A and two regular languages $L_1, L_2 \subseteq A^*$.
Output:	Is L_1 $\mathcal{C}$-separable from L_2 ?

▶ Remark. Separation generalizes the simpler *membership problem*, which asks whether a single regular language belongs to $\mathcal{C}$. Indeed $L \in \mathcal{C}$ if and only if L is $\mathcal{C}$-separable from $A^* \setminus L$ (which is also regular and computable from L).

Most papers on separation are mainly concerned about decidability. Hence, they do not go beyond the above presentation of the problem (see [3, 16, 12, 17] for example). However, this paper specifically investigates complexity. Consequently, we shall need to be more precise and take additional parameters into account. First, it will be important to specify whether the alphabet over which the input languages is part of the input (as above) or a constant. When considering separation for some fixed alphabet A, we shall speak of "$\mathcal{C}(A)$-separation". When the alphabet is part of the input, we simply speak of "$\mathcal{C}$-separation".

Another important parameter is how the two input languages are represented. We shall consider NFAs and monoids. We speak of *separation for* NFAs *and separation for monoids*. Note that one may efficiently reduce the latter to the former. Indeed, given a language $L \subseteq A^*$ recognized by some morphism $\alpha : A^* \to M$, it is simple to efficiently compute a NFA with $|M|$ states recognizing L (see [7] for example). Hence, we have the following lemma.

▶ **Lemma 1.** *For any class $\mathcal{C}$, there is a* LogSpace *reduction from $\mathcal{C}$-separation for monoids to $\mathcal{C}$-separation for* NFAs.

Getting an efficient reduction for the converse direction is much more difficult since going from NFAs (or even DFAs) to monoids usually involves an exponential blow-up. However, we shall see in Section 3 that for many natural classes $\mathcal{C}$, this is actually possible.

2.3 Concatenation hierarchies

We now briefly recall the definition of concatenation hierarchies. We refer the reader to [18] for a more detailed presentation. A particular concatenation hierarchy is built from a starting class of languages $\mathcal{C}$, which is called its *basis*. In order to get robust properties, we restrict $\mathcal{C}$ to be a quotienting Boolean algebra of regular languages. The basis is the only parameter in the construction. Once fixed, the construction is generic: each new level is built from the previous one by applying generic operators: either Boolean closure, or polynomial closure. Let us first define these two operators.

Definition. Consider a class $\mathcal{C}$. We denote by $\mathit{Bool}(\mathcal{C})$ the *Boolean closure* of $\mathcal{C}$: for every alphabet A, $\mathit{Bool}(\mathcal{C})(A)$ is the least set containing $\mathcal{C}(A)$ and closed under Boolean operations. Moreover, we denote by $\mathit{Pol}(\mathcal{C})$ the *polynomial closure* of $\mathcal{C}$: for every alphabet A, $\mathit{Pol}(\mathcal{C})(A)$ is the least set containing $\mathcal{C}(A)$ and closed under union and marked concatenation (if $K, L \in \mathit{Pol}(\mathcal{C})(A)$ and $a \in A$, then $K \cup L, KaL \in \mathit{Pol}(\mathcal{C})(A)$).

Consider a quotienting Boolean algebra of regular languages $\mathcal{C}$. The concatenation hierarchy of basis $\mathcal{C}$ is defined as follows. Languages are classified into levels of two kinds: full levels (denoted by 0, 1, 2,...) and half levels (denoted by 1/2, 3/2, 5/2,...). Level 0 is the basis (*i.e.*, $\mathcal{C}$) and for every $n \in \mathbb{N}$,

- The *half level* $n + 1/2$ is the *polynomial closure* of the previous full level, *i.e.*, of level n.
- The *full level* $n + 1$ is the *Boolean closure* of the previous half level, *i.e.*, of level $n + 1/2$.

$$0 \xrightarrow{\mathit{Pol}} 1/2 \xrightarrow[\mathit{Bool}]{} 1 \xrightarrow{\mathit{Pol}} 3/2 \xrightarrow[\mathit{Bool}]{} 2 \xrightarrow{\mathit{Pol}} 5/2 \cdots\cdots$$

We write $\frac{1}{2}\mathbb{N} = \{0, 1/2, 1, 2, 3/2, 3, \dots\}$ for the set of all possible levels in a concatenation hierarchy. Moreover, for any basis $\mathcal{C}$ and $n \in \frac{1}{2}\mathbb{N}$, we write $\mathcal{C}[n]$ for level n in the concatenation hierarchy of basis $\mathcal{C}$. It is known that every half-level is a quotienting lattice and every full level is a quotienting Boolean algebra (see [18] for a recent proof).

We are interested in finitely based concatenation hierarchies: if $\mathcal{C}$ is the basis, then $\mathcal{C}(A)$ is finite for every alphabet A. Indeed, it was shown in [17] that for such hierarchies separation is always decidable for the levels 1/2 and 1 (in fact, while we do not discuss this in the paper, this is also true for level 3/2, see [9] for a preliminary version). In Section 4, we build on the results of [17] and show that when the alphabet is fixed, this can be achieved in polynomial time for both levels 1/2 and 1. Moreover, we shall also investigate the famous *Straubing-Thérien* hierarchy in Section 5. Our motivation for investigating this hierarchy in particular is that the results of [17] can be pushed to levels 3/2 and 2 in this special case.

3 Handling NFAs

In this section, we investigate how the representation of input languages impact the complexity of separation. We prove that for many natural classes $\mathcal{C}$ (including most of those considered in the paper), $\mathcal{C}$-separation has the same complexity for NFAs as for monoids. Because of these results, we shall be able to restrict ourselves to monoids in later sections.

▶ Remark. This result highlights a striking difference between separation and the simpler membership problem. For most classes $\mathcal{C}$, $\mathcal{C}$-membership is strictly harder for NFAs than for monoids. This is because when starting from a NFA, typical membership algorithms require to either determinize $\mathcal{A}$ or compute a monoid morphism recognizing $L(\mathcal{A})$ which involves an exponential blow-up in both cases. Our results show that the situation differs for separation.

We already have a generic efficient reduction from $\mathcal{C}$-separation for monoids to $\mathcal{C}$-separation for NFAs (see Lemma 1). Here, we investigate the opposite direction: given some class $\mathcal{C}$, is it possible to *efficiently* reduce $\mathcal{C}$-separation for NFAs to $\mathcal{C}$-separation for monoids ? As far as we know, there exists no such reduction which is generic to all classes $\mathcal{C}$.

▶ Remark. There exists an *inefficient* generic reduction from separation for NFAs to the separation for monoids. Given as input two NFAs $\mathcal{A}_1, \mathcal{A}_2$, one may compute monoid morphisms recognizing $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$. This approach is not satisfying as it involves an exponential blow-up: we end-up with monoids M_i of size $2^{|Q_i|^2}$ where Q_i is the set of states of $\mathcal{A}_i$.

Here, we present a set of conditions applying to a pair of classes $(\mathcal{C}, \mathcal{D})$. When they are satisfied, there exists an efficient reduction from $\mathcal{C}$-separation for NFAs to $\mathcal{D}$-separation for monoids. By themselves, these conditions are abstract. However, we highlight two concrete applications. First, for every positive variety $\mathcal{C}$, the pair $(\mathcal{C}, \mathcal{C})$ satisfies the conditions. Second, for every finitely based concatenation hierarchies of basis $\mathcal{C}$, there exists another finite basis $\mathcal{D}$ such that for every $n \in \frac{1}{2}\mathbb{N}$, the pair $(\mathcal{C}[n], \mathcal{D}[n])$ satisfies the conditions

We first introduce the notions we need to present the reduction and the conditions required to apply it. Then, we state the reduction itself and its applications.

3.1 Generic theorem

We fix a special two letter alphabet $\mathbb{E} = \{0, 1\}$. For the sake of improved readability, we abuse terminology and assume that when considering an arbitrary alphabet A, it always has empty intersection with $\mathbb{E}$. This is harmless as we may work up to bijective renaming.

We exhibit conditions applying to a pair of classes $(\mathcal{C}, \mathcal{D})$. Then, we prove that they imply the existence of an efficient reduction from $\mathcal{C}$-separation for NFAs to $\mathcal{D}$-separation for monoids. This reduction is based on a construction which takes as input a NFA $\mathcal{A}$ (over some arbitrary alphabet A) and builds a modified version of the language $L(\mathcal{A})$ (over $A \cup \mathbb{E}$) which is recognized by a "small" monoid. Our conditions involve two kinds of hypotheses:

1. First, we need properties related to inverse image: "$\mathcal{D}$ must be an an extension of $\mathcal{C}$".
2. The construction is parametrized by an object called "tagging". We need an algorithm which builds special taggings (with respect to $\mathcal{D}$) efficiently.

We now make these two notions more precise. Let us start with extension.

Extensions. Consider two classes $\mathcal{C}$ and $\mathcal{D}$. We say that $\mathcal{D}$ is an extension of $\mathcal{C}$ when for every alphabet A, the two following conditions hold:

- If $\gamma : (A \cup \mathbb{E})^* \to A^*$ is the morphism defined by $\gamma(a) = a$ for $a \in A$ and $\gamma(b) = \varepsilon$ for $b \in \mathbb{E}$, then for every $K \in \mathcal{C}(A)$, we have $\gamma^{-1}(K) \in \mathcal{D}(A \cup \mathbb{E})$.
- For every $u \in \mathbb{E}^*$, if $\lambda_u : A^* \to (A \cup \mathbb{E})^*$ is the morphism defined by $\lambda_u(a) = au$ for $a \in A$, then for every $K \in \mathcal{D}(A \cup \mathbb{E})$, we have $\lambda_u^{-1}(K) \in \mathcal{C}(A)$.

Positive varieties give an important example of extension. Since they are closed under inverse image, it is immediate that for every positive variety $\mathcal{C}$, $\mathcal{C}$ is an extension of itself.

Taggings. A *tagging* is a pair $P = (\tau : \mathbb{E}^* \to T, G)$ where τ is a morphism into a finite monoid and $G \subseteq T$. We call $|G|$ the *rank* of P and $|T|$ its size. Moreover, given some NFA $\mathcal{A} = (A, Q, \delta, I, F)$, P is *compatible with* $\mathcal{A}$ when the rank $|G|$ is larger than $|\delta|$.

For our reduction, we shall require special taggings. Consider a class $\mathcal{D}$ and a tagging $P = (\tau : \mathbb{E}^* \to T, G)$. We say that P *fools* $\mathcal{D}$ when, for every alphabet A and every morphism $\alpha : (A \cup \mathbb{E})^* \to M$ into a finite monoid M, if all languages recognized by α belong to $\mathit{Bool}(\mathcal{D})(A \cup \mathbb{E})$, then, there exists $s \in M$, such that for every $t \in G$, we have $w_t \in \mathbb{E}^*$ which satisfies $\alpha(w_t) = s$ and $\tau(w_t) = t$.

Our reduction requires an efficient algorithm for computing taggings which fool the output class $\mathcal{D}$. Specifically, we say that a class $\mathcal{D}$ is *smooth* when, given as input $k \in \mathbb{N}$, one may compute in LogSpace (with respect to k) a tagging of rank at least k which fools $\mathcal{D}$.

Main theorem. We may now state our generic reduction theorem. The statement has two variants depending on whether the alphabet is fixed or not.

▶ **Theorem 2.** *Let $\mathcal{C}, \mathcal{D}$ be quotienting lattices such that $\mathcal{D}$ is smooth and extends $\mathcal{C}$. Then the two following properties hold:*

- *There is a* LogSpace *reduction from $\mathcal{C}$-separation for* NFAs *to $\mathcal{D}$-separation for monoids.*
- *For every fixed alphabet A, there is a* LogSpace *reduction from $\mathcal{C}(A)$-separation for* NFAs *to $\mathcal{D}(A \cup \mathbb{E})$-separation for monoids.*

We have two main applications of Theorem 2 which we present at the end of the section. Let us first describe the reduction. As we explained, we use a construction building a language recognized by a "small" monoid out of an input NFA and a compatible tagging.

Consider a NFA $\mathcal{A} = (A, Q, \delta, I, F)$ and let $P = (\tau : \mathbb{E}^* \to T, G)$ be a compatible tagging (i.e. $|\delta| \leq |G|$). We associate a new language $L[\mathcal{A}, P]$ over the alphabet $A \cup \mathbb{E}$ and show that one may efficiently compute a recognizing monoid whose size is polynomial with respect to $|Q|$ and the rank of P (i.e $|G|$). The construction involves two steps. We first define an intermediary language $K[\mathcal{A}, P]$ over the alphabet $A \times T$ and then define $L[\mathcal{A}, P]$ from it.

We define $K[\mathcal{A}, P] \subseteq (A \times T)^*$ as the language recognized by a new NFA $\mathcal{A}[P]$ which is built by relabeling the transitions of $\mathcal{A}$. Note that the definition of $\mathcal{A}[P]$ depends on arbitrary linear orders on G and δ. We let $\mathcal{A}[P] = (A \times T, Q, \delta[P], I, F)$ where $\delta[P]$ is obtained by relabeling the transitions of $\mathcal{A}$ as follows. Given $i \leq |\delta|$, if $(q_i, a_i, r_i) \in \delta$ is the i-th transition of $\mathcal{A}$, we replace it with the transition $(q_i, (a_i, t_i), r_i) \in \delta[P]$ where $t_i \in G$ is the i-th element of G (recall that $|\delta| \leq |G|$ by hypothesis).

▶ Remark. A key property of $\mathcal{A}[P]$ is that, by definition, all transitions are labeled by distinct letters in $A \times T$. This implies that $K[\mathcal{A}, P] = L(\mathcal{A}[P])$ is recognized by a monoid of size at most $|Q|^2 + 2$.

We may now define the language $L[\mathcal{A}, P] \subseteq (A \cup \mathbb{E})^*$. Observe that we have a natural map $\mu : (A\mathbb{E}^*)^* \to (A \times T)^*$. Indeed, consider $w \in (A\mathbb{E}^*)^*$. Since $A \cap \mathbb{E} = \emptyset$ (recall that this is a global assumption), it is immediate that w admits a *unique* decomposition $w = a_1 w_1 \cdots a_n w_n$ with $a_1, \dots, a_n \in A$ and $w_1, \dots, w_n \in \mathbb{E}^*$. Hence, we may define $\mu(w) = (a_1, P(w_1)) \cdots (a_n, P(w_n)) \in (A \times T)^*$. Finally, we define,

$$L[\mathcal{A}, P] = \mathbb{E}^* \cdot \mu^{-1}(K[\mathcal{A}, P]) \subseteq (A \cup \mathbb{E})^*$$

We may now state the two key properties of $L[\mathcal{A}, P]$ upon which Theorem 2 is based. It is recognized by a small monoid and the construction is connected to the separation.

▶ **Proposition 3.** *Given a* NFA *$\mathcal{A} = (A, Q, \delta, I, F)$ and a compatible tagging P of rank n, one may compute in* LogSpace *a monoid morphism $\alpha : (A \cup \mathbb{E})^* \to M$ recognizing $L[\mathcal{A}, P]$ and such that $|M| \leq n + |A| \times n^2 \times (|Q|^2 + 2)$.*

▶ **Proposition 4.** *Let $\mathcal{C}, \mathcal{D}$ be quotienting lattices such that $\mathcal{D}$ extends $\mathcal{C}$. Consider two* NFAs *$\mathcal{A}_1$ and $\mathcal{A}_2$ over some alphabet A and let P be a compatible tagging that fools $\mathcal{D}$. Then, $L(\mathcal{A}_1)$ is $\mathcal{C}(A)$-separable from $L(\mathcal{A}_2)$ if and only if $L[\mathcal{A}_1, P]$ is $\mathcal{D}(A \cup \mathbb{E})$-separable from $L[\mathcal{A}_2, P]$.*

Let us explain why these two propositions imply Theorem 2. Let $\mathcal{C}, \mathcal{D}$ be quotienting lattices such that $\mathcal{D}$ is smooth and extends $\mathcal{C}$. We show that the second assertion in the theorem holds (the first one is proved similarly).

Consider two NFAs $\mathcal{A}_i = (A, Q_j, \delta_j, I_j, F_j)$ for $j = 1, 2$. We let $k = max(|\delta_1|, |\delta_2|)$. Since $\mathcal{D}$ is smooth, we may compute (in LogSpace) a tagging $P = (\tau : \mathbb{E}^* \to T, G)$ of rank $|G| \geq k$. Then, we may use Proposition 3 to compute (in LogSpace) monoid morphisms recognizing $L[\mathcal{A}_1, P]$ and $L[\mathcal{A}_2, P]$. Finally, by Proposition 4, $L(\mathcal{A}_1)$ is $\mathcal{C}(A)$-separable from $L(\mathcal{A}_2)$ if and only if $L[\mathcal{A}_1, P]$ is $\mathcal{D}(A \cup \mathbb{E})$-separable from $L[\mathcal{A}_2, P]$. Altogether, this construction is a LogSpace reduction to $\mathcal{D}$-separation for monoids which concludes the proof.

3.2 Applications

We now present the two main applications of Theorem 2. We start with the most simple one positive varieties. Indeed, we have the following lemma.

▶ **Lemma 5.** *Let $\mathcal{C}$ be a positive variety. Then, $\mathcal{C}$ is an extension of itself. Moreover, if $Bool(\mathcal{C}) \neq \text{REG}$, then $\mathcal{C}$ is smooth.*

That a positive variety is an extension of itself is immediate (one uses closure under inverse image). The difficulty is to prove smoothness. We may now combine Theorem 2 with Lemma 5 to get the following corollary.

▶ **Corollary 6.** *Let $\mathcal{C}$ be a positive variety such that $Bool(\mathcal{C}) \neq \text{REG}$. There exists a* LogSpace *reduction from $\mathcal{C}$-separation for* NFAs *to $\mathcal{C}$-separation for monoids.*

Corollary 6 implies that for any positive variety $\mathcal{C}$, the complexity of $\mathcal{C}$-separation is the same for monoids and NFAs. We illustrate this with an example: the *star-free languages.*

▶ **Example 7.** Consider the star-free languages (SF): for every alphabet A, $\text{SF}(A)$ is the least set of languages containing all singletons $\{a\}$ for $a \in A$ and closed under Boolean operations and concatenation. It is folklore and simple to verify that SF is a variety. It is known that SF-membership is in NL for monoids (this is immediate from Schützenberger's theorem [19]). On the other hand, SF-membership is PSpace-complete for NFAs. In fact, it is shown in [2] that PSpace-completeness still holds for *deterministic* finite automata (DFAs).

For SF-separation, we may combine Corollary 6 with existing results to obtain that the problem is in EXPTime and PSpace-hard for both NFAs and monoids. Indeed, the EXPTime upper bounds is proved in [14] for monoids and we may lift it to NFAs with Corollary 6. Finally, the PSpace lower bound follows from [2]: SF-membership is PSpace-hard for DFAs. This yields that SF-separation is PSpace-hard for both DFAs and NFAs (by reduction from membership to separation which is easily achieved in LogSpace when starting from a DFA). Using Corollary 6 again, we get that SF-separation is PSpace-hard for monoids as well. ◀

We turn to our second application: finitely based concatenation hierarchies. Consider a finite quotienting Boolean algebra $\mathcal{C}$. We associate another finite quotienting Boolean algebra $\mathcal{C}_{\mathbb{E}}$ which we only define for alphabets of the form $A \cup \mathbb{E}$ (this is harmless: $\mathcal{C}_{\mathbb{E}}$ is used as the output class of our reduction). Let A be an alphabet and consider the morphism $\gamma : (A \cup \mathbb{E})^* \to A^*$ defined by $\gamma(a) = a$ for $a \in A$ and $\gamma(0) = \gamma(1) = \varepsilon$. We define,

$$\mathcal{C}_{\mathbb{E}}(A \cup \mathbb{E}) = \{\gamma^{-1}(L) \mid L \in \mathcal{C}(A)\}$$

It is straightforward to verify that $\mathcal{C}_{\mathbb{E}}$ remains a finite quotienting Boolean algebra. Moreover, we have the following lemma.

▶ **Lemma 8.** *Let $\mathcal{C}$ be a finite quotienting Boolean algebra. For every $n \in \frac{1}{2}\mathbb{N}$, $\mathcal{C}_{\mathbb{E}}[n]$ is smooth and an extension of $\mathcal{C}[n]$.*

In view of Theorem 2, we get the following corollary which provides a generic reduction for levels within finitely based hierarchies.

▶ **Corollary 9.** *Let $\mathcal{C}$ be a finite basis and $n \in \frac{1}{2}\mathbb{N}$. There exists a* LogSpace *reduction from $\mathcal{C}[n]$-separation for* NFAs *to $\mathcal{C}_{\mathbb{E}}[n]$-separation for monoids.*

4 Generic upper bounds for low levels in finitely based hierarchies

In this section, we present generic complexity results for the fixed alphabet separation problem associated to the lower levels in finitely based concatenation hierarchies. More precisely, we show that for every finite basis $\mathcal{C}$ and every alphabet A, $\mathcal{C}[1/2](A)$- and $\mathcal{C}[1](A)$-separation are respectively in NL and in P. These upper bounds hold for both monoids and NFAs: we prove them for monoids and lift the results to NFAs using the reduction of Corollary 9.

▶ Remark. We do **not** present new proofs for the decidability of $\mathcal{C}[1/2]$- and $\mathcal{C}[1]$-separation when $\mathcal{C}$ is a finite quotienting Boolean algebra. These are difficult results which are proved in [17]. Instead, we recall the (inefficient) procedures which were originally presented in [17] and carefully analyze and optimize them in order to get the above upper bounds.

For the sake of avoiding clutter, we fix an arbitrary finite quotienting Boolean algebra $\mathcal{C}$ and an alphabet A for the section.

4.1 Key sub-procedure

The algorithms $\mathcal{C}[1/2](A)$- and $\mathcal{C}[1](A)$-separation presented in [17] are based on a common sub-procedure. This remains true for the improved algorithms which we present in the paper. In fact, this sub-procedure is exactly what we improve to get the announced upper complexity bounds. We detail this point here. Note that the algorithms require considering special monoid morphisms (called "$\mathcal{C}$-compatible") as input. We first define this notion.

$\mathcal{C}$-compatible morphisms. Since $\mathcal{C}$ is finite, one associates a classical equivalence $\sim_{\mathcal{C}}$ defined on A^*. Given $u, v \in A^*$, we write $u \sim_{\mathcal{C}} v$ if and only if $u \in L \Leftrightarrow v \in L$ for all $L \in \mathcal{C}(A)$. Given $w \in A^*$, we write $[w]_{\mathcal{C}} \subseteq A^*$ for its $\sim_{\mathcal{C}}$-class. Since $\mathcal{C}$ is a finite quotienting Boolean algebra, $\sim_{\mathcal{C}}$ is a congruence of finite index for concatenation (see [18] for a proof). Hence, the quotient $A^*/{\sim_{\mathcal{C}}}$ is a monoid and the map $w \mapsto [w]_{\mathcal{C}}$ a morphism.

Consider a morphism $\alpha : A^* \to M$ into a finite monoid M. We say that α is $\mathcal{C}$-compatible when there exists a *monoid morphism* $s \mapsto [s]_{\mathcal{C}}$ from M to $A^*/{\sim_{\mathcal{C}}}$ such that for every $w \in A^*$, we have $[w]_{\mathcal{C}} = [\alpha(w)]_{\mathcal{C}}$. Intuitively, the definition means that α "computes" the $\sim_{\mathcal{C}}$-classes of words in A^*. The following lemma is used to compute $\mathcal{C}$-compatible morphisms (note that the LogSpace bound holds because $\mathcal{C}$ and A is fixed).

▶ **Lemma 10.** *Given two morphisms recognizing regular languages $L_1, L_2 \subseteq A^*$ as input, one may compute in* LogSpace *a $\mathcal{C}$-compatible morphism which recognizes both L_1 and L_2.*

In view of Lemma 10, we shall assume in this section without loss of generality that our input in separation for monoids is a single $\mathcal{C}$-compatible morphism recognizing the two languages that need to be separated.

Sub-procedure. Consider two $\mathcal{C}$-compatible morphisms $\alpha : A^* \to M$ and $\beta : A^* \to N$. We say that a subset of N is *good* (for β) when it contains $\beta(A^*)$ and is closed under multiplication. For every good subset S of N, we associate a subset of $M \times 2^N$. We then consider the problem of deciding whether specific elements belong to it (this is the sub-procedure used in the separation algorithms).

▶ Remark. The set $M \times 2^N$ is clearly a monoid for the componentwise multiplication. Hence we may multiply its elements and speak of idempotents in $M \times 2^N$.

An (α, β, S)-*tree* is an unranked ordered tree. Each node x must carry a label $lab(x) \in M \times 2^N$ and there are three possible kinds of nodes:

- **Leaves**: x has no children and $lab(x) = (\alpha(w), \{\beta(w)\})$ for some $w \in A^*$.
- **Binary**: x has exactly two children x_1 and x_2. Moreover, if $(s_1, T_1) = lab(x_1)$ and $(s_2, T_2) = lab(x_2)$, then $lab(x) = (s_1 s_2, T)$ with $T \subseteq T_1 T_2$.
- **S-Operation**: x has a unique child y. Moreover, the following must be satisfied:
  1. The label $lab(y)$ is an idempotent $(e, E) \in M \times 2^N$.
  2. $lab(x) = (e, T)$ with $T \subseteq E \cdot \{t \in S \mid [e]_\mathcal{C} = [t]_\mathcal{C} \in S\} \cdot E$.

We are interested in deciding whether elements in $M \times 2^N$ are the root label of some computation tree. Observe that computing all such elements is easily achieved with a least fixpoint procedure: one starts from the set of leaf labels and saturates this set with three operations corresponding to the two kinds of inner nodes. This is the approach used in [17] (actually, the set of all root labels is directly defined as a least fixpoint and (α, β, S)-trees are not considered). However, this is costly since the computed set may have exponential size with respect to $|N|$. Hence, this approach is not suitable for getting efficient algorithms. Fortunately, solving $\mathcal{C}[1/2](A)$- and $\mathcal{C}[1](A)$-separation does not require to have the whole set of possible root labels in hand. Instead, we shall only need to consider the elements $(s, T) \in M \times 2^N$ which are the root label of some tree **and** such that T is a **singleton set**. It turns out that these specific elements can be computed efficiently. We state this in the next theorem which is the key technical result and main contribution of this section.

▶ **Theorem 11.** *Consider two $\mathcal{C}$-compatible morphisms $\alpha : A^* \to M$ and $\beta : A^* \to N$ and a good subset $S \subseteq N$. Given $s \in M$ and $t \in N$, one may test in* NL *with respect to $|M|$ and $|N|$ whether there exists an (α, β, S)-tree with root label $(s, \{t\})$.*

Theorem 11 is proved in the full version of the paper. We only present a brief outline which highlights two propositions about (α, β, S)-trees upon which the theorem is based.

We first define a complexity measure for (α, β, S)-trees. Consider two $\mathcal{C}$-compatible morphisms $\alpha : A^* \to M$ and $\beta : A^* \to N$ as well as a good subset $S \subseteq N$. Given an (α, β, S)-tree $\mathbb{T}$, we define the *operational height of* $\mathbb{T}$ as the greatest number $h \in \mathbb{N}$ such that $\mathbb{T}$ contains a branch with h S-operation nodes.

Our first result is a weaker version of Theorem 11. It considers the special case when we restrict ourselves to (α, β, S)-trees whose operational heights are bounded by a constant.

▶ **Proposition 12.** *Let $h \in \mathbb{N}$ be a constant and consider two $\mathcal{C}$-compatible morphisms $\alpha : A^* \to M$ and $\beta : A^* \to N$ and a good subset $S \subseteq N$. Given $s \in M$ and $t \in N$, one may test in* NL *with respect to $|M|$ and $|N|$ whether there exists an (α, β, S)-tree of operational height at most h and with root label $(s, \{t\})$.*

Our second result complements the first one: in Theorem 11, it suffices to consider (α, β, S)-trees whose operational heights are bounded by a constant (depending only on the class $\mathcal{C}$ and the alphabet A which are fixed here). Let us first define this constant. Given a finite monoid M, we define the $\mathcal{J}$-depth of M as the greatest number $h \in \mathbb{N}$ such that one may find h pairwise distinct elements $s_1, \dots, s_h \in M$ such that for every $i < h$, $s_{i+1} = x s_i y$ for some $x, y \in M$

▶ Remark. The term "$\mathcal{J}$-depth" comes from the Green's relations which are defined on any monoid [4]. We do not discuss this point here.

Recall that the quotient set $A^*/{\sim_\mathcal{C}}$ is a monoid. Consequently, it has a $\mathcal{J}$-depth. Our second result is as follows.

▶ **Proposition 13.** *Let $h \in \mathbb{N}$ be the $\mathcal{J}$-depth of $A^*/{\sim_\mathcal{C}}$. Consider two $\mathcal{C}$-compatible morphisms $\alpha : A^* \to M$ and $\beta : A^* \to N$, and a good subset $S \subseteq N$. Then, for every $(s, T) \in M \times 2^N$, the following properties are equivalent:*

1. *(s, T) is the root label of some (α, β, S)-tree.*
2. *(s, T) is the root label of some (α, β, S)-tree whose operational height is at most h.*

In view of Proposition 13, Theorem 11 is an immediate consequence of Proposition 12 applied in the special case when h is the $\mathcal{J}$-depth of $A^*/{\sim_\mathcal{C}}$ and $m = 1$.

4.2 Applications

We now combine Theorem 11 with the results of [17] to get the upper complexity bounds for $\mathcal{C}[1/2](A)$- and $\mathcal{C}[1](A)$-separation that we announced at the begging of the section.

Application to $\mathcal{C}[1/2]$. Let us first recall the connection between $\mathcal{C}[1/2]$-separation and (α, β, S)-trees. The result is taken from [17].

▶ **Theorem 14** ([17]). *Let $\alpha : A^* \to M$ be a $\mathcal{C}$-compatible morphism and $F_0, F_1 \subseteq M$. Moreover, let $S = \alpha(A^*) \subseteq M$. The two following properties are equivalent:*
- *$\alpha^{-1}(F_0)$ is $\mathcal{C}[1/2]$-separable from $\alpha^{-1}(F_1)$.*
- *for every $s_0 \in F_0$ and $s_1 \in F_1$, there exists no (α, α, S)-tree with root label $(s_0, \{s_1\})$.*

By Theorem 11 and the Immerman–Szelepcsényi theorem (which states that NL = *co*-NL), it is straightforward to verify that checking whether the second assertion in Theorem 14 holds can be done in NL with respect to $|M|$. Therefore, the theorem implies that $\mathcal{C}[1/2](A)$-separation for monoids is in NL. This is lifted to NFAs using Corollary 9.

▶ **Corollary 15.** *For every finite basis $\mathcal{C}$ and alphabet A, $\mathcal{C}[1/2](A)$-separation is in* NL *for both* NFAs *and monoids.*

Application to $\mathcal{C}[1]$. We start by recalling the $\mathcal{C}[1]$-separation algorithm which is again taken from [17]. In this case, we consider an auxiliary sub-procedure which relies on (α, β, S)-trees.

Consider a $\mathcal{C}$-compatible morphism $\alpha : A^* \to M$. Observe that M^2 is a monoid for the componentwise multiplication. We let $\beta : A^* \to M^2$ as the morphism defined by $\beta(w) = (\alpha(w), \alpha(w))$ for every $w \in A^*$. Clearly, β is $\mathcal{C}$-compatible: given $(s, t) \in M^2$, it suffices to define $[(s, t)]_\mathcal{C} = [s]_\mathcal{C}$. Using (α, β, S)-trees, we define a procedure $S \mapsto \mathit{Red}(\alpha, S)$ which takes as input a good subset $S \subseteq M^2$ (for β) and outputs a subset $\mathit{Red}(\alpha, S) \subseteq S$.

$$\mathit{Red}(\alpha, S) = \{(s, t) \in S \mid (s, \{(t, s)\}) \in M \times 2^{M^2} \text{ is the root label of an } (\alpha, \beta, S)\text{-tree}\} \subseteq S$$

It is straightforward to verify that $\mathit{Red}(\alpha, S)$ remains a good subset of M^2. We now have the following theorem which is taken from [17].

▶ **Theorem 16** ([17]). *Let $\alpha : A^* \to M$ be a morphism into a finite monoid and $F_0, F_1 \subseteq M$. Moreover, let $S \subseteq M^2$ be the greatest subset of $\alpha(A^*) \times \alpha(A^*)$ such that $\mathit{Red}(\alpha, S) = S$. Then, the two following properties are equivalent:*
- *$\alpha^{-1}(F_0)$ is $\mathit{Bool}(\mathit{Pol}(\mathcal{C}))$-separable from $\alpha^{-1}(F_1)$.*
- *for every $s_0 \in F_0$ and $s_1 \in F_1$, $(s_0, s_1) \notin S$.*

Observe that Theorem 11 implies that given an arbitrary good subset S of $\alpha(A^*) \times \alpha(A^*)$, one may compute $\mathit{Red}(\alpha, S) \subseteq S$ in P with respect to $|M|$. Therefore, the greatest subset S of $\alpha(A^*) \times \alpha(A^*)$ such that $\mathit{Red}(\alpha, S) = S$ can be computed in P using a greatest fixpoint algorithm. Consequently, Theorem 16 yields that $\mathcal{C}[1](A)$-separation for monoids is in P. Again, this is lifted to NFAs using Corollary 9.

▶ **Corollary 17.** *For every finite basis $\mathcal{C}$ and alphabet A, $\mathcal{C}[1](A)$-separation is in* P *for both* NFAs *and monoids.*

5 The Straubing-Thérien hierarchy

In this final section, we consider one of the most famous concatenation hierarchies: the Straubing-Thérien hierarchy [21, 22]. We investigate the complexity of separation for the levels 3/2 and 2.

▶ Remark. Here, the alphabet is part of the input. For fixed alphabets, these levels can be handled with the generic results presented in the previous section (see Theorem 18 below).

The basis of the Straubing-Thérien hierarchy is the trivial variety ST[0] defined by $\mathrm{ST}[0](A) = \{\emptyset, A^*\}$ for every alphabet A. It is known and simple to verify (using induction) that all half levels are positive varieties and all full levels are varieties.

The complexity of separation for the level one (ST[1]) has already been given a lot of attention. Indeed, this level corresponds to a famous class which was introduced independently from concatenation hierarchies: the piecewise testable languages [20]. It was shown independently in [3] and [11] that ST[1]-separation is in P for NFAs (and therefore for DFAs and monoids as well). Moreover, it was also shown in [5] that the problem is actually P-complete for NFAs and DFAs[1]. Additionally, it is shown in [3] that ST[1/2]-separation is in NL.

In the paper, we are mainly interested in the levels ST[3/2] and ST[2]. Indeed, the Straubing-Thérien hierarchy has a unique property: the generic separation results of [17] apply to these two levels as well. Indeed, these are also the levels 1/2 and 1 in another finitely based hierarchy. Consider the class AT of *alphabet testable languages*. For every alphabet A, AT(A) is the set of all Boolean combinations of languages A^*aA^* for $a \in A$. One may verify that AT is a variety and that AT(A) is finite for every alphabet A. Moreover, we have the following theorem which is due to Pin and Straubing [8] (see [18] for a modern proof).

▶ **Theorem 18** ([8]). *For every* $n \in \frac{1}{2}\mathbb{N}$, *we have* $\mathrm{AT}[n] = \mathrm{ST}[n+1]$.

The theorem implies that ST[3/2] = AT[1/2] and ST[2] = AT[1]. Therefore, the results of [17] yield the decidability of separation for both ST[3/2] and ST[2] (the latter is the main result of [17]). As expected, this section investigates complexity for these two problems.

5.1 The level 3/2

We have the following tight complexity bound for ST[3/2]-separation.

▶ **Theorem 19.** ST[3/2]*-separation is* PSpace*-complete for both* NFAs *and monoids.*

The PSpace upper bound is proved by building on the techniques introduced in the previous section for handling the level 1/2 of an arbitrary finitely based hierarchies. Indeed, we have ST[3/2] = AT[1/2] by Theorem 18. However, let us point out that obtaining this upper bound requires some additional work: the results of Section 4 apply to the setting in which the alphabet is fixed, this is not the case here. In particular, this is why we end up with a PSpace upper bound instead of the generic NL upper presented in Corollary 15. The detailed proof is postponed to the full version of the paper.

In this abstract, we focus on proving that ST[3/2]-separation is PSpace-hard. The proof is presented for NFAs: the result can then be lifted to monoids with Corollary 6 since ST[3/2]

[1] Since ST[1] is a variety, P-completeness for ST[1]-separation can also be lifted to monoids using Corollary 6.

is a positive variety. We use a LogSpace reduction from the quantified Boolean formula problem (QBF) which is among the most famous PSpace-complete problems.

We first describe the reduction. For every quantified Boolean formula Ψ, we explain how to construct two languages L_Ψ and L'_Ψ. It will be immediate from the presentation that given Ψ as input, one may compute NFAs for L_Ψ and L'_Ψ in LogSpace. Then, we show that this construction is the desired reduction: Ψ is true if and only if L_Ψ is not ST[3/2]-separable from L'_Ψ.

Consider a quantified Boolean formula Ψ and let n be the number of variables it involves. We assume without loss of generality that Ψ is in prenex normal form and that the quantifier-free part of Ψ is in conjunctive normal form (QBF remains PSpace-complete when restricted to such formulas). That is,

$$\Psi = Q_n\ x_n \cdots Q_1\ x_1\ \varphi$$

where $x_1 \dots x_n$ are the variables of Ψ, $Q_1, \dots, Q_n \in \{\exists, \forall\}$ are quantifiers and φ is a quantifier-free Boolean formula involving the variables $x_1 \dots x_n$ which is in conjunctive normal form.

We describe the two regular languages L_Ψ, L'_Ψ by providing regular expressions recognizing them. Let us first specify the alphabet over which these languages are defined. For each variable x_i occurring in Ψ, we create two letters that we write x_i and $\overline{x_i}$. Moreover, we let,

$$X = \{x_1, \dots, x_n\} \quad \text{and} \quad \overline{X} = \{\overline{x_1}, \dots, \overline{x_n}\}$$

Additionally, our alphabet also contains the following letters: $\#_1, \dots, \#_i, \$$. For $0 \leq i \leq n$, we define an alphabet B_i. We have:

$$B_0 = X \cup \overline{X} \quad \text{and} \quad B_i = X \cup \overline{X} \cup \{\#_1, \dots, \#_i, \$\}$$

Our languages are defined over the alphabet B_n: $L_\Psi, L'_\Psi \subseteq B_n^*$. They are built by induction: for $0 \leq i \leq n$ we describe two languages $L_i, L'_i \subseteq B_i^*$ (starting with the case $i = 0$). The languages L_Ψ, L'_Ψ are then defined as L_n, L'_n.

Construction of L_0, L'_0. The language L_0 is defined as $L_0 = (B_0)^*$. The language L'_0 is defined from the quantifier-free Boolean formula φ. Recall that by hypothesis φ is in conjunctive normal form: $\varphi = \bigwedge_{j \leq k} \varphi_j$ were φ_i is a disjunction of literals. For all $j \leq k$, we let $C_j \subseteq B_0 = X \cup \overline{X}$ as the following alphabet:

- Given $x \in X$, we have $x \in C_j$, if and only x is a literal in the disjunction φ_j.
- Given $\overline{x} \in \overline{X}$, we have $\overline{x} \in C_j$, if and only $\neg x$ is a literal in the disjunction φ_j.

Finally, we define $L'_0 = C_1 C_2 \cdots C_k$.

Construction of L_i, L'_i for $i \geq 1$. We assume that L_{i-1}, L'_{i-1} are defined and describe L_i and L'_i. We shall use the two following languages in the construction:

$$T_i = (\#_i x_i (B_{i-1} \setminus \{\overline{x_i}\})^* \$ x_i)^* \quad \text{and} \quad \overline{T_i} = (\#_i \overline{x_i} (B_{i-1} \setminus \{x_i\})^* \$ \overline{x_i})^*$$

The definition of L_i, L'_i from L_{i-1}, L'_{i-1} now depends on whether the quantifier Q_i is existential or universal.

- If Q_i is an existential quantifier (i.e. $Q_i = \exists$):

$$\begin{array}{rcl} L_i & = & (\#_i(x_i + \overline{x_i})L_{i-1}\$(x_i + \overline{x_i}))^* \#_i \\ L'_i & = & (\#_i(x_i + \overline{x_i})L'_{i-1}\$(x_i + \overline{x_i}))^* \#_i \$ \left(T_i \#_i + \overline{T_i} \#_i\right) \end{array}$$

- If the Q_i is an universal quantifier (i.e. $Q_i = \forall$):

$$\begin{array}{lcl} L_i & = & (\#_i(x_i + \overline{x_i})L_{i-1}\$(x_i + \overline{x_i}))^*\#_i \\ L'_i & = & \overline{T_i}\#_i\$(\#_i(x_i + \overline{x_i})L'_{i-1}\$(x_i + \overline{x_i}))^*\#_i\$T_i\#_i \end{array}$$

Finally, L_Ψ, L'_Ψ are defined as the languages $L_n, L'_n \subseteq (B_n)^*$. It is straightforward to verify from the definition, than given Ψ as input, one may compute NFAs for L_Ψ and L'_Ψ in LogSpace. Consequently, it remains to prove that this construction is the desired reduction. We do so in the following proposition.

▶ **Proposition 20.** *For every quantified Boolean formula Ψ, Ψ is true if and only if L_Ψ is not* ST[3/2]*-separable from L'_Ψ.*

Proposition 20 is proved by considering a stronger result which states properties of all the languages L_i, L'_i used in the construction of L_Ψ, L'_Ψ (the argument is an induction on i). While we postpone the detailed proof to the full version of the paper, let us provide a sketch which presents this stronger result.

Proof of Proposition 20 (sketch). Consider a quantified Boolean formula Ψ. Moreover, let $B_0, \dots, B_n$ and $L_i, L'_i \subseteq (B_i)^*$ as the alphabets and languages defined above. The key idea is to prove a property which makes sense for all languages L_i, L'_i. In the special case when $i = n$, this property implies Proposition 20.

Consider $0 \leq i \leq n$. We write Ψ_i for the sub-formula $\Psi_i := Q_i\ x_i \cdots Q_1\ x_1\ \varphi$ (with the free variables $x_{i+1}, \dots, x_n$). In particular, $\Psi_0 := \varphi$ and $\Psi_n := \Psi$. Moreover, we call "*i-valuation*" a sub-alphabet $V \subseteq B_i$ such that,

1. $\#_1, \dots, \#_i, \$ \in V$ and $x_1, \overline{x_1}, \dots, x_i, \overline{x_i} \in V$, and,
2. for every j such that $i < j \leq n$, one of the two following property holds:
   - $x_j \in V$ and $\overline{x_j} \notin V$, or,
   - $x_j \notin V$ and $\overline{x_j} \in V$.

Clearly, an i-valuation corresponds to a truth assignment for all variables x_j such that $j > i$ (i.e. those that are free in Ψ_i): when the first (resp. second) assertion in Item 2 holds, x_j is assigned to $\top$ (resp. $\bot$). Hence, abusing terminology, we shall say that an i-valuation V *satisfies* Ψ_i if Ψ_i is true when replacing its free variables by the truth values provided by V.

Finally, for $0 \leq i \leq n$, if $V \subseteq B_i$ is an i-valuation, we let $[V] \subseteq V^*$ as the following language. Given $w \in V^*$, we have $w \in [V]$ if and only if for every $j > i$ either $x_j \in \mathsf{alph}(w)$ or $\overline{x_j} \in \mathsf{alph}(w)$ (by definition of i-valuations, exactly one of these two properties must hold). Proposition 20 is now a consequence of the following lemma.

▶ **Lemma 21.** *Consider $0 \leq i \leq n$. Then given an i-valuation V, the two following properties are equivalent:*

1. *Ψ_i is satisfied by V.*
2. *$L_i \cap [V]$ is not* ST[3/2]*-separable from $L'_i \cap [V]$.*

Lemma 21 is proved by induction on i using standard properties of the polynomial closure operation (see [18] for example). The proof is postponed to the full version of the paper. Let us explain why the lemma implies Proposition 20.

Consider the special case of Lemma 21 when $i = n$. Observe that $V = B_n$ is an n-valuation (the second assertion in the definition of n-valuations is trivially true since there are no j such that $n < j \leq n$). Hence, since $\Psi = \Psi_n$ and $L_\Psi, L'_\Psi = L_n, L'_n$, the lemma yields that,

1. Ψ is satisfied by V (i.e. Ψ is true).
2. $L_\Psi \cap [V]$ is not ST[3/2]-separable from $L'_\Psi \cap [V]$.

Moreover, we have $[V] = (B_n)^*$ by definition. Hence, we obtain that Ψ is true if and only if L is not ST[3/2]-separable from L' which concludes the proof of Proposition 20. ◀

5.2 The level two

For the level two, there is a gap between the lower and upper bound that we are able to prove. Specifically, we have the following theorem.

▶ **Theorem 22.** ST[2]*-separation is in* EXPTime *and* PSpace*-hard for both* NFAs *and monoids.*

Similarly to what happened with ST[3/2], the EXPTime upper bound is obtained by building on the techniques used in the previous section. Proving PSpace-hardness is achieved using a reduction from ST[3/2]-separation (which is PSpace-hard by Theorem 19). The reduction is much simpler than what we presented for ST[3/2] above. It is summarized by the following proposition.

▶ **Proposition 23.** *Consider an alphabet A and $H, H' \subseteq A^*$. Let $B = A \cup \{\#, \$\}$ with $\#, \$ \notin A$, $L = \#(H'\#(A^*\$\#)^*)^* H\#(A^*\$\#)^* \subseteq B^*$ and $L' = \#(H'\#(A^*\$\#)^*)^* \subseteq B^*$. The two following properties are equivalent:*
1. *H is* ST[3/2]*-separable from H'.*
2. *L is* ST[2]*-separable from L'.*

Proposition 23 is proved using standard properties of the polynomial and Boolean closure operations. The argument is postponed ot the full version of the paper. It is clear than given as input NFAs for two languages H, H', one may compute NFAs for the languages L, L' defined Proposition 23 in LogSpace. Consequently, the proposition yields the desired LogSpace reduction from ST[3/2]-separation for NFAs to ST[2]-separation for NFAs. This proves that ST[2]-separation is PSpace-hard for NFAs (the result can then be lifted to monoids using Corollary 6) since ST[2] is a variety).

6 Conclusion

We showed several results, all of them raising new questions. First we proved that for many important classes of languages (including all positive varieties), the complexity of separation does not depend on how the input languages are represented. A natural question is whether the technique can be adapted to encompass more classes. In particular, one may define more permissive notions of positive varieties by replacing closure under inverse image by weaker notions. For example, many natural classes are *length increasing positive varieties*: closure under inverse image only has to hold for length increasing morphisms (*i.e.*, morphisms $\alpha : A^* \to B^*$ such that $|\alpha(w)| \geq |w|$ for every $w \in A^*$). For example, the levels of another famous concatenation hiearchy, the dot-depth [1] (whose basis is $\{\emptyset, \{\varepsilon\}, A^+, A^*\}$) are length increasing positive varieties. Can our techniques be adapted for such classes? Let us point out that there exists no example of natural class $\mathcal{C}$ for which separation is decidable and strictly harder for NFAs than for monoids. However, there are classes $\mathcal{C}$ for which the question is open (see for example the class of locally testable languages in [10]).

We also investigated the complexity of separation for levels 1/2 and 1 in finitely based concatenation hierarchies. We showed that when the alphabet is fixed, the problems are respectively in NL and P for any such hierarchy. An interesting follow-up question would be to push these results to level 3/2, for which separation is also known to be decidable in any finitely based concatenation hierarchy [9]. A rough analysis of the techniques used in [9] suggests that this requires moving above P.

Finally, we showed that in the famous Straubing-Thérien hierarchy, ST[3/2]-separation is PSpace-complete and ST[2]-separation is in EXPTime and PSpace-hard. Again, a natural question is to analyze ST[5/2]-separation whose decidability is established in [9].

References

1 Janusz A. Brzozowski and Rina S. Cohen. Dot-Depth of Star-Free Events. *Journal of Computer and System Sciences*, 5(1):1–16, 1971.

2 Sang Cho and Dung T. Huynh. Finite automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88(1):99–116, 1991.

3 Wojciech Czerwiński, Wim Martens, and Tomáš Masopust. Efficient Separability of Regular Languages by Subsequences and Suffixes. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP'13)*, pages 150–161. Springer-Verlag, 2013.

4 James Alexander Green. On the Structure of Semigroups. *Annals of Mathematics*, 54(1):163–172, 1951.

5 Tomás Masopust. Separability by piecewise testable languages is PTIME-complete. *Theoretical Computer Science*, 711:109–114, 2018.

6 Jean-Éric Pin. The Dot-Depth Hierarchy, 45 Years Later. In *The Role of Theory in Computer Science - Essays Dedicated to Janusz Brzozowski*, pages 177–202, 2017.

7 Jean-Éric Pin. Mathematical Foundations of Automata Theory. In preparation, 2018. URL: `https://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf`.

8 Jean-Eric Pin and Howard Straubing. Monoids of upper triangular Boolean matrices. In *Semigroups. Structure and Universal Algebraic Problems*, volume 39 of *Colloquia Mathematica Societatis Janos Bolyal*, pages 259–272. North-Holland, 1985.

9 Thomas Place. Separating Regular Languages with Two Quantifier Alternations. Unpublished, a preliminary version can be found at `https://arxiv.org/abs/1707.03295`, 2018.

10 Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating Regular Languages by Locally Testable and Locally Threshold Testable Languages. In *Proceedings of the 33rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, FSTTCS'13, pages 363–375, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

11 Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating Regular Languages by Piecewise Testable and Unambiguous Languages. In *Proceedings of the 38th International Symposium on Mathematical Foundations of Computer Science*, MFCS'13, pages 729–740. Springer-Verlag, 2013.

12 Thomas Place and Marc Zeitoun. Separating Regular Languages with First-order Logic. In *Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL'14) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'14)*, pages 75:1–75:10. ACM, 2014.

13 Thomas Place and Marc Zeitoun. The tale of the quantifier alternation hierarchy of first-order logic over words. *SIGLOG News*, 2(3):4–17, 2015.

14 Thomas Place and Marc Zeitoun. Separating Regular Languages with First-Order Logic. *Logical Methods in Computer Science*, 12(1), 2016.

15 Thomas Place and Marc Zeitoun. Adding successor: A transfer theorem for separation and covering. Unpublished, a preliminary version can be found at `http://arxiv.org/abs/1709.10052`, 2017.

16 Thomas Place and Marc Zeitoun. Going higher in the First-order Quantifier Alternation Hierarchy on Words. Unpublished, a preliminary version can be found at `https://arxiv.org/abs/1404.6832`, 2017.

17 Thomas Place and Marc Zeitoun. Separation for Dot-depth Two. In *Proceedings of the 32th Annual ACM/IEEE Symposium on Logic in Computer Science, (LICS'17)*, pages 202–213. IEEE Computer Society, 2017.

18 Thomas Place and Marc Zeitoun. Generic results for concatenation hierarchies. *Theory of Computing Systems (ToCS)*, 2018. Selected papers from CSR'17.

19 Marcel Paul Schützenberger. On Finite Monoids Having Only Trivial Subgroups. *Information and Control*, 8:190–194, 1965.

20 Imre Simon. Piecewise testable events. In *2nd GI Conference on Automata Theory and Formal Languages*, pages 214–222, 1975.

21 Howard Straubing. A Generalization of the Schützenberger Product of Finite Monoids. *Theoretical Computer Science*, 13(2):137–150, 1981.

22 Denis Thérien. Classification of Finite Monoids: The Language Approach. *Theoretical Computer Science*, 14(2):195–208, 1981.