Separability by Short Subsequences and Subwords

Piotr Hofman[1] and Wim Martens[1]

1 Department of Computer Science, University of Bayreuth, Germany

Abstract

The separability problem for regular languages asks, given two regular languages I and E, whether there exists a language S that separates the two, that is, includes I but contains nothing from E. Typically, S comes from a simple, less expressive class of languages than I and E. In general, a simple separator S can be seen as an approximation of I or as an explanation of how I and E are different. In a database context, separators can be used for explaining the result of regular path queries or for finding explanations for the difference between paths in a graph database, that is, how paths from given nodes u_1 to v_1 are different from those from u_2 to v_2. We study the complexity of separability of regular languages by combinations of subsequences or subwords of a given length k. The rationale is that the parameter k can be used to influence the size and simplicity of the separator. The emphasis of our study is on tracing the tractability of the problem.

1998 ACM Subject Classification H.1.0 Models and Principles — General; H.2 Database Management; F.4.3 Formal Languages — Decision problems

Keywords and phrases Separability, complexity, graph data, debugging

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.1

1 Introduction

More and more people today are being confronted with systems that are becoming increasingly complex. Computers are becoming more and more powerful and, in the current boom of *big data*, are making decisions based on rapidly growing data sets. When we use such systems, or when we develop them, it is crucial that we have a sufficient understanding of why they do what they do. For example, when a robot performs an unexpected action, a developer should understand why the robot did the action before she can fix the error. Similarly, when a query on a database returns an answer that should not be there, we need some understanding of the data and the query before we can make a correction.

This motivates a need to search for explanations of the behavior of complex systems. We want to make a first step towards investigating to which extent *separation problems* can be useful for explaining the result of queries. Separation problems come from language theory and study differences between languages. Assume that we have two regular word languages I and E, given by their non-deterministic finite automata. A language S *separates* I from E if it *includes* I and *excludes* E, that is $I \subseteq S$ and $S \cap E = \emptyset$. If S comes from a class of languages $\mathcal{S}$, it could be seen as an approximation of I within $\mathcal{S}$. The language E can be used to tune how closely S should approximate I. For example, if E is the complement of I, then no approximation is possible and finding a separator reduces to finding an $S \in \mathcal{S}$ that is equal to I.[1] In this paper, we are mostly interested in separators that come from classes of very simple languages that only express properties about subsequences and subwords.

[1] In this case, separation corresponds to a *rewritability* or *definability* problem.

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martin Ugarte; pp. 1–17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We claim that separation problems are rather general and seem to be helpful in a wide range of scenarios. For example, separators can give users a description of why tuples are not selected by a regular path query. Say that a regular path query r on a graph database G does not return an answer (u, v) that we expected. In the usual semantics of regular path queries, this means that there is no path from u to v that matches r in G. If we consider the graph G as a finite automaton A_G with initial state u and accepting state v, then a separator for r and A_G that is simple enough to be understandable by a human could provide a description of why the expected tuple (u, v) was not in the answer. For example, if $S = \Sigma^* a \Sigma^* b \Sigma^*$ would be a separator, it could mean that r requires paths to have an a-edge before a b-edge (since $r \subseteq S$) but no such path exists from u to v. Notice that, in this scenario, A_G is expected to be much larger than r.

Similarly, separators may be useful to understand differences between subgraphs. More precisely, consider a system that is abstracted as a (large) finite, edge-labeled graph G in which each edge-label represents an action of the system. Assume that the system, after having started in some initial state i, arrives, after a long up-time, in a certain undesirable state s^- whereas, according to common sense, it should have arrived in s^+. As an aid to understand why the system arrived in s^- instead of s^+ it may be helpful to compute a separator between between the systems (G, i, s^-) and (G, i, s^+), consisting of all paths in G that lead from i to s^- and from i to s^+, respectively. For example, if $S = \Sigma^* a \Sigma^* b \Sigma^*$ would be a separator, it would mean that all labels of paths from i to s^- have a subsequence ab, whereas this is not the case for any of the paths from i to s^+. In this sense, S has the potential to pinpoint a difference between (G, i, s^-) and (G, i, s^+) in simple terms.

In this paper we want to make a first step in this direction. More precisely, we study the complexity of separability. That is, given two languages I and E and a class of separators $\mathcal{S}$, we study the complexity of deciding if there exists an $S \in \mathcal{S}$ that separates I from E. Here, I and E are given as non-deterministic finite automata (or, equivalently, edge-labeled graphs) and for $\mathcal{S}$ we consider languages that reason about the presence and/or absence of certain subsequences or subwords.

Previous work [5, 15] considered separability with respect to *piecewise testable languages*. A language is piecewise testable if it can be defined as a boolean combination of languages of the form $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$, where the a_i are symbols from Σ. So, piecewise testable languages reason about subsequences of words. It can be decided in PTIME if the language of two given NFAs I and E can be separated by a piecewise testable language [5, 15]. Tractability of this problem may come as a slight surprise in the light that many basic static analysis questions concerning NFAs (such as containment and universality, for example) are already PSPACE-complete. However, in the case that a separator exists, it is not yet clear from [5, 15] how to construct a separator that would be useful for explaining the behavior of a system to a user. Indeed, the work shows that *non-existence* of a separator for I and E can be witnessed by a polynomial-size common pattern but, if a separator exists, it can be a complex boolean combination that reasons about long subsequences.

We want to come closer to simple separators by limiting the boolean combinations and the length of the subsequences involved. That is, we look at unions, intersections, and positive combinations of languages of the form $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$ where n is bounded from above by a given parameter k. We also investigate similar combinations of subword languages, that is, languages of the form $\Sigma^* a_1 a_2 \cdots a_n \Sigma^*$. Our motivation to look at subsequence languages and subword languages comes from our belief that these may be helpful in generating understandable explanations: one does not have to be an expert in the internal design of a system to understand that one can avoid the error state by "not performing action b after

action a".

Our results focus on finding which combinations of separators and languages I, E may lead to favorable complexities for separability. Apart from the most general cases we consider, the complexity results range from PTIME to Π_2^P for separability by combinations of subsequence languages and from PTIME to PSPACE for combinations of subword languages. For the most general cases, our best current upper bound is NEXPTIME. Some of our results are reductions to simpler problems, such as separability of a language from a word. These simpler problems are interesting in their own respect. For example, it is also interesting to generate a simple reason why, e.g., a regular expression cannot be matched in a very long text.

Finally, we stress that we think that a system for generating simple separators or explanations should explore many different classes of separators. In this paper we only focus on subwords or subsequences, but a simple explanation for, say, the behavior of a query may also consist of completely different concepts. Just to mention one example, we intend to investigate separability by Parikh images in the future as well. Kopczynsky and Widjaja Lin [7] show that this approach could be feasible from a complexity point of view. A system could then search in parallel for separators in a wide array of classes and offer the user the simplest ones it can find.

The motivation in this paper mainly comes from explaining the results of queries, which is also a significant motivation for *provenance in databases*, a very successful line of research (see, e.g., [3, 23] for an overview). Storing and handling provenance in databases for explaining the results of queries is an approach that is orthogonal to ours, since we do not rely on the availability of provenance data. Another approach on explaining the results of queries, on relational data, was recently taken by Roy and Suciu [18]. Their approach also does not depend on the presence of provenance data and is based on *intervention*, which means that they investigate which tuples significantly affect the result of the answer.

Related Work

This paper is directly motivated by [5, 15], where it was shown that it can be decided in PTIME if two regular languages given by their NFA can be separated by a piecewise testable language. Recently, this problem has also been shown to be PTIME-complete [24]. Separability by locally testable and locally threshold testable languages, which are closely related to piecewise testable languages, was investigated by Place et al. [14]. They provide algorithms to solve both problems in co-NEXPTIME and 2EXPSPACE respectively. In addition they proved that both problems are coNP-hard. Place and Zeitoun recently proved that deciding separability of regular languages by first-order-logic is in EXPTIME [16] if the languages are given by their semigroups. For given NFAs, their techniques imply a 2EXPTIME upper bound. We also refer to [17] for an overview of these results.

Our approaches have roots in separability by piecewise testable and locally testable languages. Piecewise testable languages were defined and studied by Simon [19, 20], who showed that a regular language is piecewise testable iff its syntactic monoid is J-trivial and iff both the minimal DFA for the language and the minimal DFA for the reversal are partially ordered. Stern [21] proved that it is PTIME-decidable if a language, given by its DFA, is piecewise testable. A language is *locally testable* if membership of a word can be tested by inspecting its prefixes, suffixes and infixes, up to some length that depends on the language. The problem if a given regular language is locally testable was posed by McNaughton and Papert [12] and independently solved by McNaughton and Zalcstein [11, 26] and by Brzozowski and Simon [2].

Separation is closely related to Craig interpolation [4]. Craig interpolants are defined with respect to a given implication $\varphi \Rightarrow \psi$ and are formulas ρ such that $\varphi \Rightarrow \rho$ and $\rho \Rightarrow \psi$. Moreover, ρ only contains atoms that occur in *both* φ and ψ. Hence, ρ can be seen as a separator between φ and $\neg\psi$. Craig interpolants have been used for verifying safety in a system in the context of model checking [6, 10]. The classical results on Craig interpolation say that, in first-order logic, every valid implication has an interpolant. (So, for valid implications, it is trivial to decide if an interpolant exists.) Lutz and Wolter investigated complexity questions in the context of interpolants for the description logic $\mathcal{ALC}$ [8]. In particular, they showed that deciding whether there is a uniform interpolant (over a given signature) of a given TBox is 2EXPTIME-complete.

The concept of *query inseparability* was recently investigated by Botoeva et al. [1]. This problem asks, for two knowledge bases K_1 and K_2 and a class C of queries, whether every query in C has the same answer over K_1 and K_2.

Very recently, Masopust and Thomazo investigated the complexity of the *characterization problem* for k-piecewise testable languages, that is, boolean combinations of $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$ with $n \leq k$ [9]. The characterisation problem asks whether a given language *is* a k-piecewise testable language.

2 Preliminaries

For a finite set S, we denote its cardinality by $|S|$. By Σ we always denote an alphabet, that is, a finite set of symbols. A (Σ-)*word* w is a finite concatenation of symbols $a_1 \cdots a_n$, where $n \geq 0$ and $a_i \in \Sigma$ for all $i = 1, \ldots, n$. The *length of* w, denoted by $|w|$, is n. The empty word is denoted by ε and has length zero. The set of all Σ-words is denoted by Σ^*. A *language* is a set of words.

We assume familiarity with finite automata and regular expressions. In regular expressions, we also use sets to denote disjunctions of symbols. For example, $\Sigma^* a \Sigma^*$ denotes all words that have an a.

We denote a *(nondeterministic) finite automaton* or *NFA* $\mathcal{A}$ as a tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is its finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of accepting states. We sometimes use $q_1 \xrightarrow{a} q_2 \in \delta$ to denote that $q_2 \in \delta(q_1, a)$. The *size* of $\mathcal{A}$, denoted by $|\mathcal{A}|$, is defined as $|Q| + \sum_{q,a} |\delta(q, a)|$, which is the total number of transitions and states. An NFA is *deterministic* (or a *DFA*) when every $\delta(q, a)$ consists of at most one element.

For an automaton A or regular expression r we write $L(A)$, resp., $L(r)$ for their language. We sometimes identify a regular expression r or automaton A with its language and write, for example, $w \in r$ instead of $w \in L(r)$.

2.1 Subsequences and Subwords

Let $v = a_1 \cdots a_n$. If $w \in \Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$, we say that v is a *subsequence* of w and we denote it by $v \preccurlyeq w$. Moreover v is a k-subsequence if it is a subsequence and has length at most k. If $w \in \Sigma^* v \Sigma^*$, then v is a *subword* of w, denoted $v \trianglelefteq w$. It is a k-subword if, additionally, it has length at most k. For $k \in \mathbb{N}$, a *k-subsequence language* is a language of the form $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_\ell \Sigma^*$ with $\ell \leq k$ and $a_1, \ldots, a_\ell \in \Sigma$. Similarly, a *$k$-subword language* is a language of the form $\Sigma^* a_1 a_2 \cdots a_\ell \Sigma^*$ with $\ell \leq k$ and $a_1, \ldots, a_\ell \in \Sigma$. A language is a *subsequence language*, resp., *subword language* if there exists a k such that it is a k-subsequence, resp., k-subword language.

We use the following notation on automata, expressions, and languages L:

- k-Subseqs(L) (resp., k-Subwords(L)) is the set of k-subsequences (resp., k-subwords) of words from the language L, (sometimes we identify a word w with a language $\{w\}$ and we use notation k-Subwords(w) instead of k-Subwords($\{w\}$)),
- closure$^{\preccurlyeq}(L)$ (resp., closure$^{\sqsubseteq}(L)$) is the set of words that contain a word from L as a subsequence (resp., subword).

▶ **Observation 1.** *For a given word w, one can construct in polynomial time a DFA of size $O(|w|^2)$ which recognizes all words that are a subsequence of w.*

As a corollary, notice that we can also construct a DFA of size $O(k \cdot |w|^2)$ for all k-subsequences of a given word in polynomial time. This can be obtained from the DFA of Observation 1 by taking a product with the state DFA that accepts all words of length at most k. (This DFA has $k+1$ states.)

▶ **Observation 2.** *For a given word w and alphabet Σ one can construct in polynomial time a DFA which accepts all Σ-words that are supersequences of w.*

2.2 Separability

For two languages I and E, we say that language S *separates I from E* if S contains I and is disjoint from E. Notation-wise, we will consistently use I for the language to be *included* in S and E for the language to be *excluded* from S. We say that I *is separable from E* by a class of languages $\mathcal{S}$ if there exists an $S \in \mathcal{S}$ that separates I from E.

We are interested in cases where the separating language S come from classes of *simple* languages $\mathcal{S}$, such as:

- subsequence languages;
- subword languages;
- finite unions, intersections, positive combinations, or boolean combinations of subsequence- or subword languages.

By *boolean combinations* we mean finite combinations of unions, intersections, and complements. *Positive* combinations are boolean combinations that do not use complements. We note that boolean combinations of subsequence languages are also known as *piecewise testable languages* [19, 20].

We parametrize the families of separators by the length of the subsequences or subwords that we allow. For example, if $\mathcal{S}$ is the class of unions of subsequence languages, then $\mathcal{S}_k$ denotes the class of unions of k-subsequence languages. We study the following problem.

Problem:	Separability of $\mathcal{I}$ from $\mathcal{E}$ by $\mathcal{S}$	($\mathcal{I}, \mathcal{E}, \mathcal{S}$: classes of languages)
Input:	Languages $I \in \mathcal{I}$, $E \in \mathcal{E}$ given by NFAs, and a parameter $k \in \mathbb{N}$ in unary.	
Question:	Is there an $S \in \mathcal{S}_k$ that separates I from E?	

We consider variants of the separability problem where $\mathcal{S}$ comes from the aforementioned simple classes of separator languages. The classes $\mathcal{I}$ and $\mathcal{E}$ are usually either regular languages or singleton words.

The input parameter k serves as a measure for how complex we allow the separating language to be, i.e., combinations of k-subsequence languages or k-subword languages. Of course, since a separator can still be relatively complex even if k is small, we also want to explore other parameters in future work.

When we speak about the complexity of separability, we always assume that I and E are given as NFAs. We denote the sizes of these NFAs by $|I|$ and $|E|$ respectively. We assume that k is provided in unary (or, alternatively, k should not be larger than the given NFAs) to simplify some of the proofs.

2.3 Inclusion and Exclusion Equivalence

In this section we provide tools that allow us to simplify some cases of separability. We say that language I is *inclusion-equivalent* to I' with respect to a class of separators $\mathcal{S}$ if, for every language E, we have that

$$I \text{ is separable from } E \text{ by } \mathcal{S} \iff I' \text{ is separable from } E \text{ by } \mathcal{S}.$$

Similarly, E is *exclusion-equivalent* to E' with respect to $\mathcal{S}$ if, for every language I,

$$I \text{ is separable from } E \text{ by } \mathcal{S} \iff I \text{ is separable from } E' \text{ by } \mathcal{S}.$$

We extend this terminology to automata, so that we can also say, for example, that A and A' are inclusion-equivalent or exclusion-equivalent if their languages are.

In the remainder of this section, we prove basic properties about inclusion- and exclusion equivalent languages. The properties hold for general languages, so they do not require regularity of I or E.

▶ **Lemma 3.** *Let $\mathcal{S}$ be a class of separators. Let language I be inclusion-equivalent to I' w.r.t. $\mathcal{S}$ and E be exclusion-equivalent to E' w.r.t. $\mathcal{S}$. Then, for each $S \in \mathcal{S}$,*

$$S \text{ separates } I \text{ from } E \text{ iff } S \text{ separates } I' \text{ from } E'.$$

We are not aware of any work that defines equivalences between languages up to separability, but we note that a similar notion appears in Place et al. [15]. They defined an equivalence for separability of single words by a k-piecewise testable language, i.e., a boolean combination of k-subsequences. Additionally they mentioned that, for a given k, there exists a smallest k-piecewise testable language that contains a given regular language L. However, without a restriction on k, a smallest piecewise testable language containing L, i.e., a canonical piecewise testable approximation for L, does not exist.

We believe that it is useful to think about equivalences between languages. Due to non-existence of a smallest piecewise testable language equivalent to a given language L, we do not have any nice characterisation for equivalences with respect to boolean combinations. However, we provide characterizations for the weaker classes of separators we consider in this paper. The characterizations apply to more general notions than subsequences or subwords. More precisely, they hold for quasi-orders (preorder) on words.

▶ **Definition 4.** A *quasi-order* is a binary relation which is transitive and reflexive. A *well-quasi-order* is a quasi order $\lesssim$ such that any infinite sequence of elements $x_0, x_1, x_2, \ldots$ contains an increasing pair $x_i \lesssim x_j$ with $i < j$.

We present the lemmas for quasi-orders here, but readers only interested in subsequences or subwords can simply think about the subsequence (resp., subword) ordering $\preccurlyeq$ ($\trianglelefteq$) when reading them. There is one exception, however. For Lemma 6 we need a *well*-quasi-orders. It is well-known that $\preccurlyeq$ is a well-quasi-order (due to Higman's lemma) but $\trianglelefteq$ is not. Indeed, for the infinite sequence $x_n = 10^n1$ (for increasing values of n), there is no $i < j$ such that x_i is a subword of x_j. However, as we will see later, the lemmas do apply for k-subsequences and k-subwords.

Let $\lesssim$ be a quasi-order on words over alphabet Σ. For a word $w \in \Sigma^*$, the *$\lesssim$-language induced by w* is the language $\{u \in \Sigma^* \mid w \lesssim u\}$, that is, the set of all words u that are at least as large as w with respect to $\lesssim$. We usually leave the word w implicit and say that a language is a $\lesssim$-language if there exists such a word w. Subsequence and subword languages are examples of $\lesssim$-languages.

▶ **Lemma 5.** *For every quasi-order $\lesssim$ on words, the following are equivalent for languages E and E':*

(a) E is exclusion-equivalent to E' w.r.t. $\lesssim$-languages;
(b) E is exclusion-equivalent to E' w.r.t. unions of $\lesssim$-languages;
(c) E is exclusion-equivalent to E' w.r.t. intersections of $\lesssim$-languages;
(d) E is exclusion-equivalent to E' w.r.t. positive combinations of $\lesssim$-languages; and
(e) $\forall_{w\in E}\exists_{w'\in E'}$ such that $w \lesssim w'$ and $\forall_{w'\in E'}\exists_{w\in E}$ such that $w' \lesssim w$.

We now turn to a similar characterisation as Lemma 5 for inclusion-equivalence. For inclusion-equivalence, however, the characterisation is no longer the same for all positive combinations. For example, $\{ab, aa\}$ is inclusion-equivalent to $\{a\}$ with respect to subsequence languages and intersections thereof, but not with respect to unions or positive combinations of subsequence languages.

We characterize these two cases in the following two lemmas.

▶ **Lemma 6.** *Let $\lesssim$ be a well-quasi-order on words. The following are equivalent for languages I and I':*

(a) I is inclusion-equivalent to I' w.r.t. unions of $\lesssim$-languages;
(b) I is inclusion-equivalent to I' w.r.t. positive combinations of $\lesssim$-languages;
(c) $\text{closure}^{\lesssim}(I) = \text{closure}^{\lesssim}(I')$; and
(d) For every $\lesssim$-minimal element $i \in I$ there is a $\lesssim$-minimal element in $i' \in I'$ such that $i \lesssim i'$ and $i' \lesssim i$.

Where $\text{closure}^{\lesssim}(I)$ is the set of all words that are larger or equal with respect to $\lesssim$ than some word from I.

▶ **Lemma 7.** *For every quasi-order $\lesssim$ on words, the following are equivalent for languages I and I':*

(a) I is inclusion-equivalent to I' w.r.t. $\lesssim$-languages;
(b) I is inclusion-equivalent to I' w.r.t. intersections of $\lesssim$-languages;
(c) $\cap_{w\in I}\{u \in \Sigma^* : u \lesssim w\} = \cap_{w'\in I'}\{u \in \Sigma^* : u \lesssim w'\}$

We now argue that Lemmas 5–7 can be used in the context of k-subword and k-subsequence languages. To this end, for $k \in \mathbb{N}$ and words w_1, w_2, we define

$$w_1 \preccurlyeq_k w_2 \text{ if and only if } k\text{-Subseqs}(w_1) \subseteq k\text{-Subseqs}(w_2).$$

Similarly, we say that $w_1 \trianglelefteq_k w_2$ if k-Subwords$(w_1) \subseteq k$-Subwords(w_2). Since $\preccurlyeq_k$ and $\trianglelefteq_k$ are well-quasi orders for every $k \in \mathbb{N}$, we have that Lemmas 5–7 apply to k-subsequences and k-subwords as well.

2.4 Witnesses for Non-Separability

We provide simple characterizations of non-separability that state, in each of the cases, what kind of witness one should search for proving non-separability. We found these characterizations to be quite useful in proofs.

▶ **Lemma 8.** *Let I and E be two regular languages. Then I is not separable from E*

(a) by a union of k-subsequence languages iff there exists a word $w_I \in I$ that can not be separated from E, i.e., such that every k-subsequence s of w_I appears in some word $w_s \in E$.
(b) by an intersection of k-subsequence languages iff there exists a word $w_E \in E$ that cannot be separated from I, i.e., such that every k-subsequence that appears in every word from I also appears in w_E.
(c) by a positive combination of k-subsequence languages iff there exists words $w_I \in I$ and $w_E \in E$ such that k-Subseqs$(w_I) \subseteq k$-Subseqs(w_E).
(d) by a boolean combination of k-subsequences iff there exist words w_I in I and w_E in E such that k-Subseqs$(w_I) = k$-Subseqs(w_E).

Item (d) from the lemma is rather standard and follows almost immediately from the definition of k-piecewise testable languages by Simon [19, 20]; see also Lemma 4.1 in [17].

The corresponding lemma for k-subword languages is analogous. The different cases in the lemma give a good idea of the different flavors of the separation problem when one searches for a witness of non-separability. For example, in case (d), we are looking for words in I and E that have precisely the same sets of k-subsequences. It is usually much harder to argue that such witnesses are small than in, say, case (c) where only inclusion in one direction is required.

3 A Tractable Case

The main result of this section is a tractability result of separability of I from E by k-subsequences, by finite unions, intersections, and positive combinations thereof, if a certain condition holds on E. The main idea is that we reduce E to a small language E' which is exclusion-equivalent and solve the separation problem of I and E'. Afterwards, we generalize this to two more expressive form of separators, namely *k-subsequences of constant-length words*, and finite unions thereof. Here, for a constant c, a k-subsequence of c-length words is a language of the form

$$\Sigma^* w_1 \Sigma^* \cdots \Sigma^* w_\ell \Sigma^*$$

where $\ell \leq k$ and each w_i has length at most c. We think that such languages could be helpful to separate languages in practice, because they seem to be rather expressive, potentially simple to understand, yet have a tractable separation problem.

We need a little bit of terminology to get started. We call a set $X \subseteq Q$ of states of an automaton $A = (Q, \Sigma, \delta, q_0, F)$ a *strongly connected component*, or *SCC*, if it is a maximal strongly connected component in the usual graph representation of A. Let $X \subseteq Q$ be a set of states of an automaton $A = (Q, \Sigma, \delta, q_0, F)$. We say that A' is obtained from A by *collapsing* X if A' is the image of A under a homomorphism $g : Q \to (Q \setminus X) \uplus \{q_X\}$ which is the identity on $Q \setminus X$ and maps each $u \in X$ to q_X. Furthermore, q_X is accepting in A' if and only if X contains an accepting state.

▶ **Lemma 9.** *For a given automaton A let X be one of its SCCs. If A' is obtained from A by collapsing X, then A and A' are exclusion-equivalent with respect to subsequence languages,*

unions of subsequence languages, intersections of subsequence languages, and positive combinations.

Notice that Lemma 9 is an easy corollary of Lemma 5. Furthermore, if A and A' are exclusion-equivalent with respect to subsequence languages then, for every fixed $k \in \mathbb{N}$, they are also exclusion-equivalent with respect to the (less expressive) k-subsequence languages. (Similarly for unions, intersections, and positive combinations thereof.) Therefore, Lemma 9 relativises to k-subsequence languages.

The following notion, a *core* of an NFA, will be central in our search for tractable separation. Basically, we will obtain tractability for separation of languages for which we can find a small approximation of its core. The overall idea is similar to the idea of *kernelization* in the context of finding fixed-parameter tractability, but, as far as we know, our approach does not necessarily lead to a proof of fixed-parameter tractability of the general problem.

▶ **Definition 10.** An NFA C is a *core* of a language E, if the following hold:

1. E and C are exclusion-equivalent w.r.t. positive combinations of subsequences,
2. C is minimal among all NFAs of exclusion-equivalent languages to E w.r.t. positive combinations of subsequences.

So, the rationale of a core C is that it captures the whole complexity of E when it comes to separation. When we want to decide whether I is separable from E, we can obtain the correct result by deciding separability of I from C instead. The challenge is to be able to compute a core (or sufficiently small approximation thereof) from the NFA of E. The following observation gives us a first step.

▶ **Observation 11.** *Each core of a regular language E contains no loops other than self-loops.*

Proof. If a core C is not a DAG with self-loops then it contains a non-trivial SCC. Thus, according to Lemma 9 we can collapse the non-trivial SCC and obtain a smaller core, which contradicts the minimality of C. ◀

Hence, an initial approximation of a core of E can be obtained by collapsing all its SCCs. We will see later that cores can be even smaller in general.

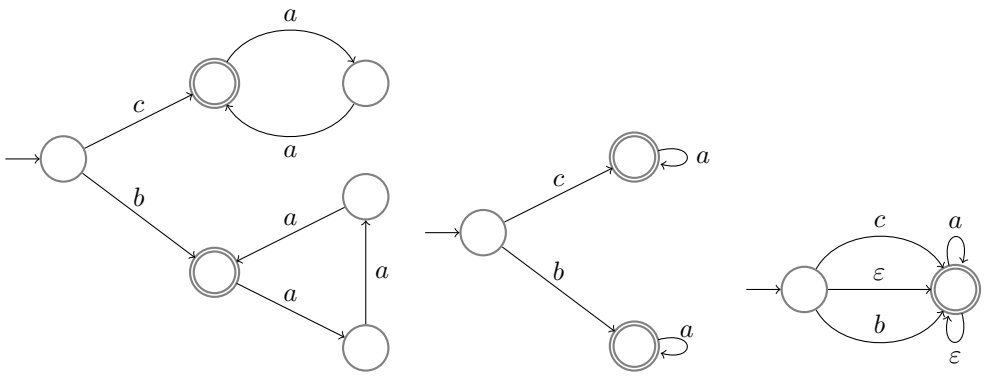
Observe that, in general, computing a core is at least NP-hard, because minimizing an automaton which is a DAG is NP-hard [22]. Therefore we will search for approximations of cores that we know how to compute efficiently.

3.1 Core-Approximations

Given an NFA $A = (Q, \Sigma, \delta, q_0, F)$, the following procedure computes an exclusion-equivalent NFA C' that may be much smaller than A and therefore can be seen as an approximation of a core.

For each SCC $X \subseteq Q$ we collapse X. As collapsing SCCs does not change the exclusion-equivalence class of an NFA, the obtained DAG D with self-loops is exclusion-equivalent to A (Lemma 9). In a next step we collapse further to obtain a possibly even smaller exclusion-equivalent NFA:

- First, for every transition $q_1 \xrightarrow{a} q_2$ in D, we add a transition $q_1 \xrightarrow{\varepsilon} q_2$, thereby obtaining an ε-NFA D_ε. Notice that, according to Lemma 5, languages D_ε and D are exclusion-equivalent w.r.t. (k-)subsequence languages, unions thereof, intersections thereof, and positive combinations thereof.



(a) Automaton $A(E)$. **(b)** After collapsing SCCs. **(c)** After the second reduction.

Figure 1 Illustration of a core-approximation.

- Second, we take a *weak bisimulation quotient* of D_ε. This step does not change the language, so the exclusion-equivalence class trivially remains the same.

Bisimulation quotients are probably the simplest and best known heuristic to minimize an NFA [13]. *Weak* bisimulation is simply the version of the bisimulation quotient that takes ε-transitions into account. Weak bisimulation gives us a better refinement of the automaton D_ε than the ordinary bisimulation quotient. We illustrate a core-approximation in Figure 1.

We refer to the resulting automaton as a *core-approximation* of A. The subsequent results in this section imply tractability of the separation problems whenever we can compute a constant-size core-approximation.

3.2 Using Core-Approximations to Separate

We explain how an arbitrary language E' that is exclusion-equivalent to E can be used to separate I from E. In particular, the exposition applies to cores and core-approximations.

In the cases of separability by k-subsequences and combinations thereof that have unions, the number of states of E, and therefore also the number of states of E's cores, gives an upper bound for the length of the subsequences that we need to consider for separation.

▶ **Lemma 12.** *If automata I and E are separable by a k-subsequence language (resp., union of k-subsequence languages, or positive combination of k-subsequence languages), then they are separable by an $|E|$-subsequence language (resp., union of $|E|$-subsequence languages, or positive combination of $|E|$-subsequence languages).*

These bounds can be used to obtain the following upper bounds on separability by k-subsequence languages, intersections, unions, and positive combinations thereof. In the Lemma, f denotes a function and *poly* denotes a polynomial function.

▶ **Lemma 13.** *For a given automata I, E and a number k we can decide if I is separable from E*

(a) by a k-subsequence language in time $O(poly(|I|) \cdot \Sigma \cdot f(|E|))$;
(b) by an intersection of k-subsequence languages in time $O(poly(|I|) \cdot |\Sigma|^{|E|+1} \cdot f(|E|))$;
(c) by a union of k-subsequence languages in time $O(poly(|I|) \cdot f(|E|))$; and

(d) by a positive combination of k-subsequence languages in time $O(poly(|I|) \cdot f(|E|))$.

Proof sketch. In the proof, f is an exponential function in cases (a) and (b) and double exponential in (c) and (d), but we stress that we have not yet attempted any optimization. All our algorithms are based on exhaustive checking of witnesses that fulfil certain constraints; we provide a sketch for case (a).

(a) By Lemma 12 we know that we can bound k by $|E|$. Let X be the set of words u that have length at most k and such that there is no $v \in E$ such that $u \preccurlyeq v$. We can separate I from E if and only if there is a word $s \in X$ that is a subsequence of every word in I. Without loss of generality we can restrict ourselves to minimal words in X with respect to the $\preccurlyeq$ order. Indeed, if $w \preccurlyeq w'$ and closure$^{\preccurlyeq}(w')$ separates I from E then closure$^{\preccurlyeq}(w)$ is also a separator. The number of such minimal words is bounded by $|\Sigma_E|^k + |\Sigma|$, where Σ_E is the alphabet used in E. So, a naive algorithm could simply enumerate all $|\Sigma_E|^k + |\Sigma|$ such minimal words and test the conditions. Testing if such a word w does not have a supersequence in E can be done in polynomial time by computing from w the DFA A_w of Observation 2 and testing if its intersection with E is empty. Testing if w is a subsequence of every word in I can be answered by testing if I is included in $L(A_w)$. Since A_w is deterministic, this can be done in time polynomial in $|I|$ and the length of w too. Thus, we get that the overall complexity is bounded by $(p_1(|I|) + p_2(|E|)) \cdot (\Sigma + |\Sigma_E|^{|E|})$, where p_1 and p_2 are polynomials. ◀

From Lemma 13, we immediately get tractability of separability if we can find a core-approximation of E that has constant size. This is trivial, for example, if E has a constant number of SCCs.

▶ **Theorem 14.** *For a given automata I, E and a number k, if the core approximation of E has constant size, we can decide in PTIME if I is separable from E by*

(a) a k-subsequence language,
(b) an intersection of k-subsequence languages,
(c) a union of k-subsequence languages, and
(d) a positive combination of k-subsequence languages.

3.3 Sequences of Words

We now generalize the algorithm for Theorem 14 to deal with more expressive separators that combine subsequences and subwords. For k and c in $\mathbb{N}$, a language of *k-subsequences of c-words* is a language of the form $\Sigma^* w_1 \Sigma^* \cdots \Sigma^* w_k \Sigma^*$ where each w_i has length at most c. In the remainder of this section, k should be thought of as an input to the separability problem as before, and c should be thought of as a constant. Our aim is to show that, if c is constant, then we can extend Theorem 14 to languages of k-sequences of c-words.

The idea consists of doing a preprocessing step on the NFA for E and then performing an analogous construction as in the previous section. Essentially, the preprocessing step consists of extending E's alphabet such that it also reads c-tuples of Σ-symbols.

More formally, let $A = (Q, \Sigma, \delta, q_0, F)$ be an NFA. By $A^{\leq c} = (Q, \Sigma^{\leq c}, \delta^{\leq c}, q_0, F)$ we denote the NFA obtained from A as follows:

- $\Sigma^{\leq c} := \uplus_{1 \leq i \leq c} \Sigma^i$
- for every $a \in \Sigma$ and $q \in Q$, $\delta^{\leq c}(q, a) := \delta(q, a)$
- for every $(a_1, \ldots, a_i) \in \Sigma^{\leq c}$ and $q \in Q$, $\delta^{\leq c}(q, (a_1, \ldots, a_i))$ is the set of states that can be reached in A by reading the word $a_1 \cdots a_i$, that is, $\cup_{p \in \delta^{\leq c}(q, (a_1, \ldots, a_{i-1}))} \delta(p, a_i)$.

That is, $A^{\leq c}$ behaves exactly the same as A but, whenever it reads a tuple $(a_1, \ldots, a_i)$ it behaves as if A would read the word $a_1 \cdots a_i$.

When we have given automata I and E, we can use the core-approximation of $E^{\leq c}$ to separate I from E by languages of k-subsequences of c-words. Here, we construct the core-approximation of $E^{\leq c}$ (w.r.t. k-subsequences over $\Sigma^{\leq c}$) as explained in Section 3.1.

▶ **Theorem 15.** *For a given automata I, E, a number k, and a constant c, if the core approximation of $E^{\leq c}$ has constant size, it is decidable in PTIME if I is separable from E by*

- *a language of k-subsequence of c-words, or*
- *a union of languages of k-subsequences of c-words.*

The proof of the Theorem is obtained by minor adaptations in the proof of Theorem 14 (a) and (c) to deal with the different alphabet of $E^{\leq c}$. (To this end, we also need a slightly different version of Lemma 12, adapted to the new alphabet. Its proof is analogous.)

We conclude this section by remarking that the approach we used to show Theorem 15 does not naively generalize to separators that have intersection. Basically, when intersection is allowed, we cannot treat constant-length words as single symbols anymore. For example, when ab and ba are treated as single symbols in Σ, then $\Sigma^*ab\Sigma^* \cap \Sigma^*ba\Sigma^*$ does not contain aba. When ab and ba are words of length two, then $aba \in \Sigma^*ab\Sigma^* \cap \Sigma^*ba\Sigma^*$. So, the naive application of the former algorithm may return the wrong result.

4 Separability by k-Subsequences

In Sections 4 and 5 we present the result of a systematical investigation of the complexity of separability in different constellations. More precisely, we consider all combinations of separating I from E where I and E can be a regular language or a word. As possible separators, in Section 4 we consider all combinations of k-subsequence languages that we also considered before and, in Section 5 we consider the same combinations of k-subword languages. We note that many complexity bounds are not yet tight.

The complexity landscape in this section shows separability by k-subsequences is often hard, even if one of the languages is a singleton word. More precisely, if we restrict either I or E (but not both) to be a single word, separation seems to remain NP- or coNP-hard in almost all cases.[2] Only when both I and E are words, we can prove that separability by k-subsequence languages and combinations thereof is in PTIME. On the positive side, almost all upper bounds lie within Π_2^P which is lower than the typical PSPACE bound which one expects for many static analysis problems for NFAs such as universality and containment.

▶ **Theorem 16.** *Given NFAs for I and E, and a number k, separability of I from E*

(a) by k-subsequence languages is NP-complete;
(b) by unions of k-subsequence languages is NP-hard and in Π_2^P;
(c) by intersections of k-subsequence languages is coNP-hard and in Π_2^P; and
(d) by positive combinations of k-subsequence languages is coNP-complete.
(e) by a boolean combinations of k-subsequence languages is coNP-hard and in NEXPTIME.

All hardness results already follow from Theorem 17, where I is a singleton. In cases (a) and (b), we have reductions from SAT that use an acyclic NFA for E. However, the proof

[2] There is one notable exception in which we do not yet know the precise complexity: Theorem 18(b).

requires non-determinism in the NFA. For (c), (d), and (e), we have several reductions. One is from the Hamilton path problem and shows the problem is hard even if the automaton for E acyclic, but it makes linearly many copies of the input graph for the Hamilton problem. The other is from a problem investigated by Wood [25], doesn't produce an acyclic automaton, but has a shorter proof, which we present here. Hardness already holds for $k = 1$, which is a sharp contrast with the PTIME upper bounds in Theorems 14 and 15.

4.1 Restricted Cases

We now consider severely restricted cases in which at least one of the two languages is restricted to be a single word. If we restrict I to be a single word, then we see that all separability problems become coNP-complete when the separator languages have intersection, and NP-complete otherwise.

▶ **Theorem 17.** *Given word w_I, an NFA for E, and a number k, separability of w_I from E*

(a) by k-subsequence languages or by unions of k-subsequence languages is NP-complete;
(b) by intersections, positive combinations, or boolean combinations of k subsequence languages is coNP-complete.

Proof sketch. We sketch the lower bound proofs. The hardness proof for (a) is by reduction from SAT. For a given formula φ in conjunctive normal form, over the variables $\{x_1, \ldots, x_k\}$, we construct w_I and E such that φ is satisfiable if and only if w_I can be separated from E by a k-subsequence language.

More precisely, E is defined over the alphabet $\{T, F\}$ and contains those words of length k that correspond to valuations that do not satisfy the formula φ. Valuations are encoded as words in a straightforward manner: a word $X_1 \cdots X_k$ with each $X_i \in \{T, F\}$ encodes the assignment that assigns x_i true if and only if $X_i = T$. The construction of the automaton to recognize E is in paragraph below. The word w_I is defined as $(TF)^k$. Since w_I contains every k-subsequence over $\{T, F\}$, the word w_I can be separated from the language E by a k-subsequence if and only if $E \neq (T + F)^k$ by Lemma 8. This is equivalent to the fact that there is a valuation which satisfies the formula φ.

Clearly, k and w_I can be constructed in polynomial time from φ. It remains to show how to construct an NFA for E. Let $\varphi = C_1 \wedge \cdots \wedge C_m$. This NFA is a union of sub-automata that accept those words of length k that encode valuations that do *not* satisfy C_i. Figure 2 contains and ε-NFA E for the formula $\varphi = (x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee \neg x_4) \wedge (x_1 \vee x_3 \vee x_4)$. Notice that this ε-NFA for E and thus also an NFA for E can be constructed in polynomial time.

The coNP hardness for (b) is almost immediate from the NP-hardness of the following problem [25]: Given a DFA A over alphabet Σ, is there a word in $L(A)$ that contains every letter in Σ?

Indeed, given the DFA A over $\Sigma = \{a_1, \ldots, a_n\}$, we can define $w_I = a_1 \cdots a_n$ and $E = L(A)$ and ask if there is an intersection of languages of the form $\Sigma^* a_i \Sigma^*$ that separates w_I from E. Notice in this case, if there is a separator then the intersection $\cap_{i=1}^{n}(\Sigma^* a_i \Sigma^*)$ of *all* such languages is a separator as well. So, w_I is separable from E by an intersection of 1-subsequence languages if and only if it is separable by $\cap_{i=1}^{n}(\Sigma^* a_i \Sigma^*)$. This means that w_I is separable from E if and only if no word in $L(A)$ contains every letter from Σ.

Given the proof for intersections, observe that if there exists a separator that uses positive boolean combinations, namely of a form $X_1 \cup X_2 \cup \cdots \cup X_n$ then one of of X_i separates w_I from E as well. Thus the problem for positive boolean combination is equivalent to

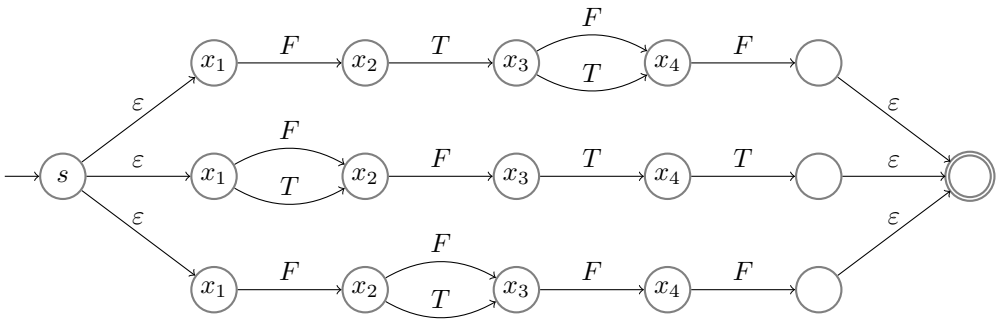


Figure 2 Construction of ε-NFA for E in the proof of Theorem 17(a).

the problem for intersection and is coNP-hard as well. Finally, in the solution of the above problem, negation does not help, as w_E contains all letters from Σ. Thus solving the boolean combination problem is equivalent to solving the intersection problem as well. ◀

The converse case in which we restrict E to be a single word shows a similar picture, but note that we do not have a coNP lower bound or PTIME upper bound for separability by unions or positive combinations. For unions, for example, we need to decide if there exists a word $w_I \in I$ such that all its k-subsequences appear in w_E (Lemma 8). One can easily construct a polynomial-size DFA for all k-subsequences that do not appear in w_E, but in general there is no small DFA for all words that *contain* a k-subsequence that does not appear in w_E. (The intuition is that such an automaton needs to guess which symbols to use for the subsequence.) Therefore, checking if all words in I contain some k-subsequence that does not appear in w_E seems to be difficult.[3] Conversely, for a hardness proof, the fact that w_E is a single word gives little freedom for encoding gadgets.

▶ **Theorem 18.** *Given word w_E, an NFA for I, and a number k, separability of I from w_E*

(a) by k-subsequence languages or by intersections of k-subsequence languages is NP-complete;
(b) by unions or positive combinations of k-subsequence languages is in coNP; and
(c) by boolean combinations of k-subsequence languages is coNP-complete.

Hardness in case (a) is by reduction from the longest common subsequence problem. For (c), the question is equivalent to Theorem 17(b) because the separators are closed under complement.

Finally, when both languages are restricted to be a single word, separability can be decided in PTIME by using standard automata techniques (Observation 1).

▶ **Theorem 19.** *Given words w_I, w_E, and number k, separability of w_I from w_E by k-subsequence languages or by unions, intersections, positive combinations, or boolean combinations thereof is in PTIME.*

5 Separability by k-Subwords

Analysis of subwords provides us a similar overview like in subsequence case, but the complexities are more diverse. We see more cases that are tractable, but the arguments why

[3] If I contains a polynomial number of shortest words, it can be done in polynomial time due to Theorem 19.

this is so are rather easy. The main reason why, in some cases, separability by k-subwords seems to be easier than by subsequences is because a given word w can only have $O(k|w|)$ many subwords, whereas it can have exponentially many subsequences. That said, subwords can also be used to reason about the distance between positions in a word. This can be exploited to encode corridor tiling and which makes separability by boolean combinations of k-sequences PSPACE-hard.

▶ **Theorem 20.** *Given NFAs for I and E, and a number k, separability I from E*

(a) by k-subword languages is in PTIME;
(b) by unions or positive combinations of k-subword languages is PSPACE-complete;
(c) by intersections of k-subword languages is coNP-complete; and
(d) by boolean combinations of k-subword languages is in NEXPTIME and PSPACE-hard.

5.1 Restricted Cases

When we restrict one of the languages to be a word, we see that separability becomes coNP-complete at worst.

▶ **Theorem 21.** *Given word w_I, an NFA for E, and a number k, separability of w_I from E*

(a) by k-subword languages or by unions of k-subword languages is in PTIME; and
(b) by intersections, positive combinations, or boolean combinations of k subword languages is coNP-complete.

▶ **Theorem 22.** *Given word w_E, an NFA for I, and a number k, separability of I from w_E*

(a) by k-subword languages or by unions, intersections, or positive combinations thereof is in PTIME; and
(b) by boolean combinations of k-subword languages is coNP-complete.

Finally, for the sake of completeness, we mention that, when both languages are a word, separation is trivially in PTIME.

▶ **Theorem 23.** *Given words w_I, w_E, and number k, separability of w_I from w_E by k-subword languages or by unions, intersections, positive combinations, or boolean combinations thereof is in PTIME.*

6 Separability by k-Prefixes and k-Suffixes

For the sake of completeness, we mention that deciding separability by combinations of prefixes of length up to k is in polynomial time in all cases we consider. A *k-prefix language* is a language of the form $w\Sigma^*$ where $|w| \leq k$. All boolean combinations of k-prefix languages are defined similarly as before.

▶ **Observation 24.** *Given NFAs for I and E, and a number k, separation of I from E by k-prefix languages, or by unions, intersections, positive combinations, and boolean combinations thereof is in PTIME.*

The proofs are rather straightforward adaptations of the corresponding proofs in [5]. Naturally, the observation also holds for k-suffix languages by a reduction that simply reverses the NFAs for I and E.

7 Conclusions

Separation of regular languages seems to be a worthwhile approach to investigate for tackling several problems in graph databases, for example, approximating regular path queries (by specifying a query to be approximated and a second query of paths that should not be matched), computing explanations of the results of regular path queries, and for computing explanations of the differences between edge-labeled s-t-graphs in general.

When one searches for separators to provide explanations, it does not really matter to a user which class of separators is considered, as long as the separator is simple enough to understand and interpret. In fact, a system is likely to be perceived to be much more useful and intelligent when it is able to return simple specimens from many classes of separators, as opposed to complex specimens that come from a limited number of classes. Intuitively, the former case corresponds to a situation where explanations can be very diverse and, in the latter case explanations always have a similar flavor. In this paper we investigated to which extent subsequences, subwords, and combinations thereof can be used to describe the difference between regular languages (or graphs) I and E. Our main motivation for considering subsequences and subwords are that we feel that they are easy to understand and, at the same time, may be reasonably expressive. It was already shown in [5, 15] that, for regular languages, deciding the existence of an arbitrarily complex separator involving subsequences, prefixes or suffixes is in PTIME. Here we made a step towards finding simpler separators in the sense that we considered a given bound k on the length of the subwords and subsequences involved. We showed that, if E can be reduced to a sufficiently small substructure, its core-approximation, there is good hope that the difference between I and E can be described in simple terms, if they can be separated.

Efficient construction of separators is clearly the goal of this line of research and is a challenging problem. We would like to understand when separators exist, when they are small, and when they are efficiently computable. This paper gives us better understanding of this case, which can then serve as a basis towards producing separators that are, ultimately, human readable.

Acknowledgments

We would like to thank the anonymous reviewers of ICDT 2015 for many helpful remarks. This work was supported by DFG grant MA4938/2-1.

References

1 E. Botoeva, R. Kontchakov, V. Ryzhikov, F. Wolter, and M. Zakharyaschev. Query inseparability for description logic knowledge bases. In *Principles of Knowledge Representation and Reasoning (KR)*, 2014.

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

3 P. Buneman and W. C. Tan. Provenance in databases. In *International Conference on Management of Data (SIGMOD)*, pages 1171–1173, 2007.

4 W. Craig. Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. *The Journal of Symbolic Logic*, 22(3), 1957.

5 W. Czerwinski, W. Martens, and T. Masopust. Efficient separability of regular languages by subsequences and suffixes. In *International Conference on Automata, Languages and Programming (ICALP)*, pages 150–161, 2013.

6 T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *Principles of Programming Languages (POPL)*, pages 232–244, 2004.

7 E. Kopczynski and A. Widjaja To. Parikh images of grammars: Complexity and applications. In *Logic in Computer Science (LICS)*, pages 80–89, 2010.

8 C. Lutz and F. Wolter. Foundations for uniform interpolation and forgetting in expressive description logics. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 989–995, 2011.

9 T. Masopust and M. Thomazo. On k-piecewise testability (preliminary report). *CoRR*, abs/1412.1641, 2014.

10 K. L. McMillan. Applications of craig interpolants in model checking. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 1–12, 2005.

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

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

13 R. Paige and R. Tarjan. Three parition refinement algorithms. *SIAM Journal on Computing*, 16:973–989, 1987.

14 T. Place, L. van Rooijen, and M. Zeitoun. Separating regular languages by locally testable and locally threshold testable languages. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 363–375, 2013.

15 T. Place, L. van Rooijen, and M. Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *Mathematical Foundations of Computer Science (MFCS)*, pages 729–740, 2013.

16 T. Place and M. Zeitoun. Separating regular languages with first-order logic. In *Computer Science Logic – Logic in Computer Science (CSL-LICS)*, 2014.

17 L. Van Rooijen. *Une approche combinatoire du problème de séparation pour les langages réguliers.* PhD thesis, Université de Bordeaux, 2014.

18 S. Roy and D. Suciu. A formal approach to finding explanations for database queries. In *International Conference on Management of Data (SIGMOD)*, pages 1579–1590, 2014.

19 I. Simon. *Hierarchies of Events with Dot-Depth One*. PhD thesis, Dept. of Applied Analysis and Computer Science, University of Waterloo, Canada, 1972.

20 I. Simon. Piecewise testable events. In *Proceedings of GI Conference on Automata Theory and Formal Languages*, pages 214–222. Springer, 1975.

21 J. Stern. Complexity of some problems from the theory of automata. *Information and Control*, 66(3):163–176, 1985.

22 L. Stockmeyer and A. Meyer. Word problems requiring exponential time: Preliminary report. In *Symposium on Theory of Computing (STOC)*, pages 1–9, 1973.

23 W. C. Tan. Provenance in databases: Past, current, and future. *IEEE Data Engineering Bulletin*, 30(4):3–12, 2007.

24 Š. Holub, G. Jirśková, and T. Masopust. On upper and lower bounds on the length of alternating towers. In *Mathematical Foundations of Computer Science (MFCS), Part I*, pages 315–326, 2014.

25 P. T. Wood. Containment for XPath fragments under DTD constraints. In *International Conference on Database Theory (ICDT)*, 2003. Full version, obtained through personal communication.

26 Y. Zalcstein. Locally testable languages. *Journal of Computer and System Sciences*, 6(2):151–167, 1972.