Subsequence Matching and Analysis Problems for Formal Languages

Szilárd Zsolt Fazekas
Akita University, Japan

Tore Koß
University of Göttingen, Germany

Florin Manea
University of Göttingen, Germany

Robert Mercaş
Loughborough University, UK

Timo Specht
University of Göttingen, Germany

Abstract

In this paper, we study a series of algorithmic problems related to the subsequences occurring in the strings of a given language, under the assumption that this language is succinctly represented by a grammar generating it, or an automaton accepting it. In particular, we focus on the following problems: Given a string w and a language L, does there exist a word of L which has w as subsequence? Do all words of L have w as a subsequence? Given an integer k alongside L, does there exist a word of L which has all strings of length k, over the alphabet of L, as subsequences? Do all words of L have all strings of length k as subsequences? For the last two problems, efficient algorithms were already presented in [Adamson et al., ISAAC 2023] for the case when L is a regular language, and efficient solutions can be easily obtained for the first two problems. We extend that work as follows: we give sufficient conditions on the class of input-languages, under which these problems are decidable; we provide efficient algorithms for all these problems in the case when the input language is context-free; we show that all problems are undecidable for context-sensitive languages. Finally, we provide a series of initial results related to a class of languages that strictly includes the regular languages and is strictly included in the class of context-sensitive languages, but is incomparable to the of class context-free languages; these results deviate significantly from those reported for language-classes from the Chomsky hierarchy.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Design and analysis of algorithms

Keywords and phrases Stringology, String Combinatorics, Subsequence, Formal Languages, Context-Free Languages, Context-Sensitive Languages

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.28

Funding *Szilárd Zsolt Fazekas*: Supported by the JSPS Kakenhi Grant Number 23K10976.
Florin Manea: Supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) in the framework of the Heisenberg Programme project number 466789228.
Robert Mercaş: Supported by a Return Fellowship from the Alexander von Humboldt Foundation for a research stay at University of Göttingen, Germany.

1 Introduction

A string v is a subsequence of a string w, denoted $v \leq w$ in the following, if there exist (possibly empty) strings $x_1, \ldots, x_{\ell+1}$ and $v_1, \ldots, v_\ell$ such that $v = v_1 \cdots v_\ell$ and $w = x_1 v_1 \cdots x_\ell v_\ell x_{\ell+1}$. In other words, v can be obtained from w by removing some of its letters.

The concept of subsequence appears and plays important roles in many different areas of theoretical computer science. Prime examples are the areas of combinatorics on words, formal languages, automata theory, and logics, where subsequences are studied in connection to piecewise testable languages [71, 72, 37, 38], in connection to subword-order and downward-closures [31, 46, 45, 76, 77, 6]), in connection to binomial equivalence, binomial complexity, or to subword histories [66, 24, 49, 48, 69, 57, 67]. Subsequences are important objects of study also in the area of algorithm-design and complexity; to this end, we mention some classical algorithmic problems such as the computation of *longest common subsequences* or of the *shortest common supersequences* [16, 34, 36, 54, 56, 62, 8, 10], the testing of the Simon congruence of strings and the computation of the arch factorisation and universality of strings [32, 26, 73, 74, 18, 9, 19, 21, 28, 42, 22]; see also [44] for a survey on combinatorial pattern matching problems related to subsequences. Moreover, these algorithmic problems and other closely related ones have recently regained interest in the context of fine-grained complexity [12, 13, 3, 1, 2]. Nevertheless, subsequences appear also in more applied settings: for modelling concurrency [65, 70, 14], in database theory (especially *event stream processing* [7, 29, 78]), in data mining [50, 51], or in problems related to bioinformatics [11]. Interestingly, a new setting, motivated by database theory [39, 40, 25], considers subsequences of strings, where the substrings occurring between the positions where the letters of the subsequence are embedded are constrained by regular or length constraints; a series of algorithmic results (both for upper and lower bounds) on matching and analysis problems for the sets of such subsequences occurring in a string were obtained [20, 41, 5, 55].

The focus of this paper is the study of the subsequences of strings of a formal language, the main idea behind it being to extend the fundamental problems related to matching subsequences in a string and to the analysis of the sets of subsequences of a single string to the case of sets of strings. To this end, grammars (or automata) are succinct representations of (finite or infinite) sets of strings they generate (respectively, accept), so we are interested in matching and analysis problems related to the set of subsequences of the strings of a language, given by the grammar generating it (respectively, the automaton accepting it). This research direction is, clearly, not new. To begin with, we recall the famous result of Higman [33] which states that the downward closure of every language (i.e., the set of all subsequences of the strings of the respective language) is regular. Clearly, it is not always possible to compute an automaton accepting the downward closure of a given language, but gaining a better understanding when it is computable is an important area of research, as the set of subsequences of a language plays meaningful roles in practical applications (e.g., abstractions of complex systems, see [76, 77, 6] and the references therein). Computing the downward closure of a language is a general (although, often inefficient) way to solve subsequence-matching problems for languages; for instance, we can immediately check, using a finite automaton for the downward closure, if a string occurs as subsequence of a string of the respective language. However, it is often the case that more complex analysis problems regarding the subsequences occurring in the strings of a language cannot be solved efficiently (or, sometimes, at all) using the downward closure; such a problem is to check if a given string occurs as subsequence in all the strings of a language (chosen from a complex enough class, such as the class of context-free languages).

As a direct predecessor of this paper, motivated by similar questions, [4] approached algorithmic matching and analysis problems related to the universality of regular languages (for short, REG). More precisely, a string over Σ is called k-universal if its set of subsequences includes all strings of length k over Σ; the study of these universal strings was the focus of many recent works [9, 19, 68] and the motivation for studying universality properties in the context of subsequences is discussed in detail in, e.g., [19, 4]. The main problems addressed in [4] are the following: for $L \in \text{REG}$, over the alphabet Σ, and a number k, decide if there exists a k-universal string in L (respectively, if all strings of L are k-universal). The authors of [4] discussed efficient algorithms solving these problems and complexity lower bounds. In this paper, we extend the work of [4] firstly by proposing a more structured approach for the algorithmic study of the subsequences occurring in strings of formal languages and secondly by considering more general classes of languages, both from the Chomsky hierarchy (such as the class of context-free languages or that of context-sensitive languages) and non-classical (the class of languages accepted by deterministic finite automata with translucent letters).

Our work on subsequence-matching and analysis problems in languages defined by context-free grammars (for short, CFG) also extends a series of results related to matching subsequences in strings given as a straight line program (for short, SLP; a CFG generating a single string), or checking whether a string given as an SLP is k-universal, for some given k, e.g., see [52, 68]. In our paper, we consider the case when the input context-free languages and the CFGs generating them are unrestricted.

The approached problems and an overview of our results. As mentioned above, we propose a more structured approach for matching- and analysis-problems related to subsequences of the strings of a formal language. More precisely, we define and investigate the following five problems.

▶ **Problem 1** (∃-Subsequence)**.** *Given a language L by a machine (grammar) M accepting (respectively, generating) it and a string w, is there a string $v \in L$ such that $w \leq v$?*

▶ **Problem 2** (∀-Subsequence)**.** *Given a language L by a machine (grammar) M accepting (respectively, generating) it and a string w, do we have for all strings $v \in L$ that $w \leq v$?*

▶ **Problem 3** (∃-k-universal)**.** *Given a language L by a machine (grammar) M accepting (respectively, generating) it and integer k, check if there is a k-universal string in L.*

▶ **Problem 4** (∀-k-universal)**.** *Given a language L by a machine (grammar) M accepting (respectively, generating) it and integer k, check if all strings of $L(M)$ are k-universal.*

Alternatively, strictly from the point of view of designing an algorithmic solution, the problem above can be approached via its complement: that is, deciding if there exists at least one string in $L(M)$ which is not k-universal.

▶ **Problem 5** (∞-universal)**.** *Given a language L by a machine (grammar) M accepting (respectively, generating it) decide if there exist m-universal strings in L, for all positive integers m.*

To give some intuition on our terminology, Problems 1 and 3 can be seen as *matching problems* (find a string which contains a certain subsequence or set of subsequences), while the other three problems are *analysis problems* (decide properties concerning multiple strings of the language).

Going a bit more into details, in the main part of this paper, we investigate these problems for the case when the language L is chosen from the class of context-free languages (for short, CFL; given by CFGs in Chomsky normal form), or from the class of context-sensitive languages (for short, CSL; given by context-sensitive grammars), or from the class of languages accepted by deterministic finite automata with translucent letters (given by an automaton of the respective kind). The choice of presentation of the languages from given classes, unsurprisingly, makes a big difference w.r.t. hardness. For instance, certain singleton languages can be encoded by SLPs (essentially CFGs) exponentially more succinctly than by classical DFA, which of course introduces significant extra computation into solving subsequence-related queries [52, 68]. But, before approaching these classes of languages, we provide a series of general decidability results on these five problems, for which the choice of grammar or automaton as the way of specifying the input language L is not consequential.

For short, our results are the following. We first give (in Section 3) a series of simple sufficient conditions on a class $\mathcal{C}$ of languages (related to the computation of downward closures as well as to decidability properties for the respective class) which immediately lead to decision procedures for the considered problems; however, these procedures are inherently inefficient, even for classes such as CFL. In this context, generalizing the work of [4], we approach (in subsequent sections of this paper) each of the above problems for $\mathcal{C}$ being the class CFL and, respectively, the class CSL. While all the problems are undecidable for CSLs, we present efficient algorithms for the case of CFLs. In particular, the results obtained for CFL are similar to the corresponding results obtained for REG (i.e., if a problem was solvable in polynomial or FPT-time for REG, we obtain an algorithm from the same class for CFL). In that regard, it seemed natural to search for a class of languages which does not exhibit this behaviour, while retaining the decidability of (at least some of) these problems. To this end, we identify the class of languages accepted by deterministic finite automata with translucent letters (a class of automata which does not process the input in a sequential fashion) and show (in the final section of this paper) a series of initial promising results related to them.

2 Preliminaries

Let $\mathbb{N} = \{1, 2, \ldots\}$ denote the natural numbers and set $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ as well as $[n] = \{1, \ldots, n\}$ and $[i, n] = \{i, i + 1, \ldots, n\}$, for all $i, n \in \mathbb{N}_0$ with $i \leq n$.

An *alphabet* $\Sigma = \{1, 2, \ldots, \sigma\}$ is a finite set of symbols, called *letters*. A *string* w is a finite concatenation of letters from a given alphabet with the number of these letters giving its *length* $|w|$. The string with no letters is the *empty string* ε of length 0. The set of all finite strings over the alphabet Σ, denoted by Σ^*, is the free monoid generated by Σ with concatenation as operation. A subset $L \in \Sigma^*$ is called a *(formal) language*. Let Σ^n denote all strings in Σ^* exactly of length $n \in \mathbb{N}_0$.

For $1 \leq i \leq j \leq |w|$ denote the i^{th} letter of w by $w[i]$ and the *factor* of w starting at position i and ending at position j as $w[i, j] = w[i] \cdots w[j]$. If $i = 1$ the factor is also called a prefix, while if $j = |w|$ it is called a suffix of w. For each $a \in \Sigma$ set $|w|_a = |\{i \in [|w|] \mid w[i] = a\}|$.

Let $\mathrm{alph}(w)$ denote the set of all letters of Σ occurring in w. A length n string $u \in \Sigma^*$ is called *subsequence* of w, denoted $u \leq w$, if $w = w_1 u[1] w_2 u[2] \cdots w_n u[n] w_{n+1}$, for some $w_1, \ldots, w_{n+1} \in \Sigma^*$. For $k \in \mathbb{N}_0$, a string $w \in \Sigma^*$ is called *k-universal* (w.r.t. Σ) if every $u \in \Sigma^k$ is a subsequence of w. The *universality-index* $\iota(w)$ is the largest k such that w is k-universal.

▶ **Definition 6.** *The arch factorization of a string $w \in \Sigma^*$ is given by $w = ar_1(w) \cdots ar_{\iota(w)}(w) r(w)$ with $\iota(ar_i(w)) = 1$ and $ar_i(w)[|ar_i(w)|] \notin \mathrm{alph}(ar_i(w)[1, |ar_i(w)| - 1])$, for all $i \in [1, \iota(w)]$. Furthermore, $\mathrm{alph}(r(w)) \subsetneq \Sigma$ applies. The strings $ar_i(w)$ are called* arches *and $r(w)$ is called* the rest *of w.*
The modus $m(w)$ of w is defined as the concatenation of the last letters of each arch: $m(w) = ar_1(w)[|ar_1(w)|] \cdots ar_{\iota(w)}(w)[|ar_{\iota(w)}(w)|]$.

As an example, in the arch factorisation $w = (bca)\cdot(accab)\cdot(cab)\cdot b$ of $w \in \{a, b, c\}^*$, the parentheses denote the three arches and the rest $r(w) = b$. Further, we have $\iota(w) = 3$ and $m(bcaaccabcabb) = abb$. For more details about the arch factorization and the universality index see [32, 9].

A string v is an absent subsequence of another string w if v is not a subsequence of w [42, 43]. A shortest absent subsequence of a string w (for short, $\mathrm{SAS}(w)$) is an absent subsequence of w of minimal length, i.e., all subsequences of shorter length are found in w. We note that, for a given word w and some letter $a \notin \mathrm{alph}(r(w))$, an SAS of w is $m(w)a$ [32, 42]. An immediate observation is that any string v which is an $\mathrm{SAS}(w)$ satisfies $|v| = \iota(w) + 1$.

In this paper, we work with absent subsequences of a word w which are the shortest among all absent subsequences of w and, additionally, start with a and end with b, for some $a \in \Sigma \cup \{\varepsilon\}$ and $b \in \Sigma$. Such a shortest string which starts with a and ends with b and is an absent subsequence of w is denoted $\mathrm{SAS}_{a,b}(w)$. For instance, an $\mathrm{SAS}_{\varepsilon,b}(w)$, for some $b \notin \mathrm{alph}(r(w))$, is an $\mathrm{SAS}(w)$, such that its starting letter is not fixed, but the ending one must be b.

▶ **Definition 7.** *A grammar over an alphabet Σ is a 4-tuple $G = (V, \Sigma, P, S)$ consisting of: a set $V = \{A, B, C, \dots\}$ of non-terminal symbols, a set $\Sigma = \{a, b, c, \dots\}$ of terminal symbols with $V \cap \Sigma = \emptyset$, a non-empty set $P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ of productions and a start symbol $S \in V$.*

We represent productions $(p, q) \in P$ by $p \to q$. In G, $u = xpz$ with $x, z \in (V \cup \Sigma)^*$ is directly derivable to $v = xqz$ if a production $(p, q) \in P$ exists; in this case, we write $u \Rightarrow_G v$; the subscript G is omitted when no confusion arises. More generally, for $m \in \mathbb{N}$, we say that u is derivable to v in m steps (denoted $w \Rightarrow_G^m v$) if there exist strings $w_0, w_1, \dots, w_m \in (V \cup \Sigma)^*$ (called sentential forms) with $u = w_0 \Rightarrow_G w_1 \wedge w_1 \Rightarrow_G w_2 \wedge \cdots \wedge w_{m-1} \Rightarrow_G w_m = v$. If u is derivable to v in m steps, for some $m \in \mathbb{N}_0$, we write $u \Rightarrow_G^* v$, i.e., $\Rightarrow_G^*$ is the reflexive and transitive closure of $\Rightarrow_G$. With $L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$ we denote the language generated by G. We call a derivation a sequence $S \Rightarrow \cdots \Rightarrow w \in L(G)$. The number of steps used in the derivation is the derivation's length.

▶ **Definition 8.** *A grammar $G = (V, \Sigma, P, S)$ with $P \subseteq V \times (V \cup \Sigma)^+$ is a* context-free *grammar (for short, CFG). A language L is a context-free language (for short, CFL) if and only if there is a CFG G with $L(G) = L$.*
A grammar $G = (V, \Sigma, P, S)$, where for all $(p, q) \in P$ we have $|p| \leq |q|$, is a context-sensitive *grammar (for short, CSG). A language L is a context-sensitive language (for short, CSL) if and only if there is a CSG G with $L(G) = L$.*

The definitions above tacitly assume that CFLs and CSLs do not contain the empty string ε. Indeed, for the problems considered here, we can make this assumption. Whether $\varepsilon \in L$ or not plays no role in deciding Problems 1, 3, and 5, while $\varepsilon \in L$ immediately leads to a negative answer for Problem 2 (unless $w = \varepsilon$) and Problem 4 (unless $k = 0$). So, for simplicity, we only address languages that, by definition, *do not* contain the empty string (see also the discussions in [47, 35] about how the presence of ε in formal languages can be handled).

Also, note that every unary CFL is regular [64], so when discussing our problems for the class of CFLs we assume that the input languages are over an alphabet with at least two letters.

▶ **Definition 9.** *A CFG $G = (V, \Sigma, P, S)$ is in Chomsky normal form (CNF) if and only if $P \subseteq V \times (V^2 \cup \Sigma)$ and, for all $A \in V$, there exist some $w_A, w'_A, w''_A \in \Sigma^*$ such that $A \Rightarrow^* w_A$ and $S \Rightarrow^* w'_A A w''_A$ (these last two properties essentially say that every non-terminal of G is* useful*).*

When we discuss our problems in the case of CFLs, we assume our input is a CFG G in CNF. This does not change our results since, according to [47] and the references therein, we can transform any grammar G in polynomial time into a CFG G' in CNF such that $|G'| \in \mathcal{O}(|G|^2)$ and $L(G) = L(G')$, where $|G|$ refers to the size of a grammar determined in terms of total size of its productions.

In some cases it may be easier to view derivations in a CFG G as a derivation (parse) tree. These are rooted, ordered trees. The inner nodes of such trees are labeled with non-terminals and the leaf-nodes are labeled with symbols $X \in (V \cup \Sigma)$. An inner node A has, from left to right, the children $X_1, \ldots, X_k$, for some integer $k \geq 1$, if the grammar contains the production $A \to X_1 \cdots X_k$. As such, if we concatenate, from left to right, the leaves of a derivation tree T with root A we get a string α (called in the following the border of T) such that $A \to^* \alpha$. The depth of a derivation tree is the length of the longest simple-path starting with the root and ending with a leaf (i.e., the number of edges on this path). If G is in CNF, then all its derivation trees are binary.

▶ **Definition 10.** *For any language $L \subset \Sigma^*$ the* downward closure *$L\!\downarrow$ of L is defined as the language containing all subsequences of strings of L, i.e., $L\!\downarrow = \{v \in \Sigma^* \mid \exists w \in L : v \leq w\}$. The complementary notion of the* upward closure *$L\!\uparrow$ of a language L is the language containing all supersequences of strings in L, i.e., $L\!\uparrow = \{w \in \Sigma^* \mid \exists v \in L : v \leq w\}$.*

Our problems focus on properties of formal languages, and Problems 3, 4 and 5 are strongly connected to universality seen as a property of a language, therefore we extend the concept of universality to formal languages. We distinguish between two different ways of analyzing the universality of a language.

▶ **Definition 11.** *Let $L \subseteq \Sigma^*$ be a language. We call L k-universal universal if for every $w \in L$ it holds that $\iota(w) \geq k$. The universal universality index $\iota_\forall(L)$ is the largest k, such that L is k-universal universal. We call L k-existential universal if a string $w \in L$ exists with $\iota(w) \geq k$. The existential universality index $\iota_\exists(L)$ is the largest k, such that L is k-existential universal. In all the definitions above, the universality index of words and, respectively, languages is computed w.r.t. Σ.*

In case of a singleton language $L = \{w\}$ it holds that $\iota_\exists(L) = \iota_\forall(L) = \iota(w)$. In general the universal universality index $\iota_\forall(L)$ is the infimum of the set of all universality indices of strings in L and therefore is lower bounded by 0 and upper bounded by $\iota(w)$, for any $w \in L$ (so it is finite, for $L \neq \emptyset$). The existential universality index $\iota_\exists(L)$ is the supremum of the set of all universality indices of strings in L and, as such, can be infinite. In this setting the answer to Problem 3 and, respectively, Problem 4, can be solved by computing $\iota_\exists(L)$ and, respectively, $\iota_\forall(L)$, and then checking whether $k \leq \iota_\exists(L)$ and, respectively, $k \leq \iota_\forall(L)$. Furthermore, Problem 5 asks whether $\iota_\exists(L)$ is infinite or not. The following two lemmas are not hard to show.

▶ **Lemma 12.** *Given a string $w \in \Sigma^*$, with $|w| = n$ and $|\Sigma| = \sigma$, we can construct in time $\mathcal{O}(n\sigma)$ a minimal DFA, with $n + 1$ states, accepting the set of strings which have w as a subsequence.*

▶ **Lemma 13.** *For $k > 0$ and an alphabet Σ with $|\Sigma| = \sigma$ we can construct in time $\mathcal{O}(2^\sigma k \operatorname{poly}(\sigma))$ a minimal DFA, with $(2^\sigma - 1)k + 1$ states, accepting the set of k-universal strings over Σ.*

The computational model we use to state our algorithms is the standard unit-cost word RAM with logarithmic word-size ω (meaning that each memory word can hold ω bits). It is assumed that this model allows processing inputs of size n, where $\omega \geq \log n$; in other words, the size n of the data never exceeds (but, in the worst case, is equal to) 2^ω. Intuitively, the size of the memory word is determined by the processor, and larger inputs require a stronger processor (which can, of course, deal with much smaller inputs as well). Indirect addressing and basic arithmetical operations on such memory words are assumed to work in constant time. Note that numbers with ℓ bits are represented in $O(\ell/\omega)$ memory words, and working with them takes time proportional to the number of memory words on which they are represented. This is a standard computational model for the analysis of algorithms, defined in [23].

Our algorithms have languages as input, that is sets of strings over some finite alphabet. Therefore, we follow a standard stringology-assumption, namely that we work with an *integer alphabet*: we assume that this alphabet is $\Sigma = \{1, 2, \ldots, \sigma\}$, with $|\Sigma| = \sigma$, such that σ fits in one memory word. For a more detailed general discussion on the integer alphabet model see, e. g., [17]. In all problems discussed here, the input language is given as a grammar generating it or as an automaton accepting it. We assume that all the sets defining these generating/accepting devices (e.g., set of non-terminals, set of states, set of final states, relation defining the transition function or derivation, etc.) have at most 2^ω elements and their elements are integers smaller or equal to 2^ω (i.e., their cardinality and elements can be represented as integers fitting in one memory word). In some of the problems discussed in this paper, we assume that we are given a number k. Again, we assume that this integer fits in one memory word.

One of our algorithms (for Problem 3 in the case of CFL, stated in Theorem 23) runs in exponential time and uses exponential space w.r.t. the size of the input. In particular, both the space and time complexities of the respective algorithm are exponential, with constant base, in σ (the size of the input alphabet) but polynomial w.r.t. all the other components of the input. To avoid clutter, we assume that our exponential-time and -space algorithm runs on a RAM model where we can allocate as much memory as our algorithms needs (i.e., the size of the memory-word ω is big enough to allow addressing all the memory we need in this algorithm in constant time); for the case of $\sigma \in O(1)$, this additional assumption becomes superfluous, and for non-constant σ we stress out that the big size of memory words is only used for building large data structures, but not for speeding up our algorithms by, e.g., allowing constant-time operations on big numbers (that is, numbers represented on more than $c \log N$ bits, for some constant c and N being the size of the input).

3 General Results

We consider the problems introduced in Section 1, for the case when the language L is chosen from a class $\mathcal{C}$, and give a series of sufficient conditions for them to be decidable.

Consider a class $\mathcal{G}$ of grammars (respectively, a class $\mathcal{A}$ of automata) generating (respectively, accepting) the languages of the class $\mathcal{C}$. For simplicity, for the rest of this section, we assume that in all the problems we take as input a grammar G_L such that $L(G_L) = L$, but note that all the results hold for the case when we consider that the languages are given by an automaton from the class $\mathcal{A}$ accepting them.

Let $\mathcal{C}'$ be the class of languages $L \cap R$, where $L \in \mathcal{C}$ and $R \in$ REG. We use two hypotheses:

H1. Given a grammar G of the class $\mathcal{G}$ we can algorithmically construct a non-deterministic finite automaton A accepting the downward closure of $L(G)$.

H2. Given a grammar G of the class $\mathcal{G}$ and a non-deterministic finite automaton A, we can algorithmically decide whether the language $L(G) \cap L(A)$ is empty.

First we show that, under H1, Problems 1, 3, and 5 are decidable.

▶ **Theorem 14.** *If H1 holds, then Problems 1, 3, and 5 are decidable.*

Proof. We start by observing that the following straightforward properties hold:

- for a string w, there exists $v \in L$ such that $w \leq v$ if and only if $w \in L\downarrow$.
- for an integer $k > 0$, there exists $v \in L$ such that v is k-universal if and only if there exists $v' \in L\downarrow$ such that v' is k-universal.

In each of Problems 1, 3, and 5, we are given a grammar G generating the language L. According to H1, we construct a non-deterministic automaton A accepting $L\downarrow$, the downward closure of L.

For Problem 1, it is sufficient to check if $L(A) = L\downarrow$ contains the string w, which is clearly decidable. For Problem 3 we need to decide if L contains a k-universal string. By our observations, it is enough to check if $L\downarrow$ contains a k-universal string. This can be decided, for the automaton A, according to [4]. For Problem 5 we need to decide if L contains a k-universal string, for all $k \leq 0$. This is also decidable, for A, according to the results of [4]. ◀

Secondly, we show that, under H2, Problems 1, 2, 3, and 4 are decidable.

▶ **Theorem 15.** *If H2 holds, then Problems 1, 2, 3, and 4 are decidable.*

Proof. In all the inputs of Problems 1, 2, 3, and 4 when considering CFL and CSL, we are given a grammar G, which generates the language L.

For Problems 1 and 2, by Lemma 12 we construct a DFA B accepting the regular language $w\uparrow$ of strings which have w as a subsequence. If the intersection of L (given as the grammar G which generates it) and $L(B)$ is empty, which is decidable, under H2, then the considered instance of Problem 1 is answered negatively; otherwise, it is answered positively. By making the final state of B non-final, and all the other states final, we obtain a DFA B' which accepts $\Sigma^* \setminus w\uparrow$. If the intersection of L and $L(B')$ is empty, then the answer to the considered instance of Problem 2 is positive; otherwise, it is negative.

For Problems 3 and 4, by Lemma 13 we construct a DFA B accepting the regular language of k-universal strings. If the intersection of L and $L(B)$ is empty, then the answer to the considered instance of Problem 3 is negative; otherwise, it is positive. By making the final state of B non-final, and all the other states final, we obtain a DFA B' which accepts exactly all strings which are not k-universal. If the intersection of L and $L(B')$ is empty, then the answer to the considered instance of Problem 4 is positive; otherwise, it is negative. ◀

It is worth noting that, even for classes which fulfill both hypotheses above (such as the CFLs [75, 35]), there are several reasons why the algorithms resulting from the above theorems are not efficient. On the one hand, constructing an automaton which accepts the downward closure of a language is not always possible, and even when this construction is possible (when the language is from a class for which H1 holds) it cannot always be done efficiently. For instance, in the case of CFLs, this may take inherently exponential time w.r.t. the size of the input grammar [30]; in this paper, we present more efficient algorithms for

Problems 1, 2, 3, and 4 in the case of CFLs, which do not rely on Theorem 15. On the other hand, the results of Theorem 15 rely, at least partly, on the construction of a DFA accepting all k-universal strings, which takes exponential time in the worst case, as it may have an exponential number of states (both w.r.t. the size of the input alphabet and w.r.t. the binary representation of the number k, which is given as input for some of these problems).

Interestingly, the class CSL does not fulfil any of the above hypotheses. In fact, as our last general result, we show that all five problems we consider here are undecidable for CSL.

▶ **Theorem 16.** *Problems 1, 2, 3, 4, 5 are undecidable for the class of CSL, given as CSGs.*

Proof. To obtain the undecidability of all the problems, we show reductions from the emptiness problem for Context Sensitive Languages. Assume that we have a CSL L, specified by a grammar G, as the input for the emptiness problem for CSL. Assume L is over the alphabet $\Sigma = \{1, \ldots, \sigma\}$, and that the CSG G, has the starting symbol S. Let 0 be a fresh letter (i.e., $0 \notin \Sigma$).

To show the undecidability of Problems 1 and 2, we construct a new grammar G' which has all the non-terminals, terminals, and productions of G and, additionally, G' has a new starting symbol S' and the productions $S' \to \sigma S$ and $S' \to 0$.

It is immediate that there exists a string $w \in L(G')$ which contains σ as a subsequence, if and only if $L(G)$ is not empty. Furthermore, all strings of $L(G')$ contain 0 as a subsequence (that is, the production $S' \to \sigma S$ is not the first production in the derivation of any terminal string) if and only if $L(G)$ is empty. As the emptyness problem is undecidable for CSL (given as grammars), it follows that Problems 1 and 2 are also undecidable for this class of languages.

To show the undecidability of Problem 3, we construct a new grammar G' which has all the non-terminals, terminals, and productions of G and, additionally, G' has a new starting symbol S' and the production $S' \to (12 \cdots \sigma)S$. Clearly, $L(G')$ contains a 1-universal string (over Σ) if and only if $L(G) \neq \emptyset$. Thus, it follows that Problem 3 is also undecidable for this class of languages.

To show the undecidability of Problem 4, we construct a new grammar G' which has all the non-terminals, terminals, and productions of G and, additionally, G' has a new starting symbol S' and the productions $S' \to 012 \cdots \sigma$ and $S' \to S$. Clearly, all the strings of $L(G')$ are 1-universal (over $\Sigma \cup \{0\}$) if and only if $L(G) = \emptyset$ (as any string which would be derived in G' starting with the production $S' \to S$ would not contain 0). Hence, Problem 4 is also undecidable for CSL.

To show the undecidability of Problem 5, we construct a new grammar G' which has all the non-terminals, terminals, and productions of G and, additionally, G' has a new starting symbol S' and a fresh non-terminal R and the productions $S' \to 012 \cdots \sigma$, $S' \to RS$, $R \to 01 \cdots \sigma R$, and $R \to 01 \cdots \sigma$. Clearly, $L(G')$ contains m-universal strings (over $\Sigma \cup \{0\}$), for all $m \geq 1$, if and only if $L(G) \neq \emptyset$ (as we can use R to pump arches in the strings of $L(G')$ if and only if there exists at least one derivation where S can be derived to a terminal string). Accordingly, Problem 5 is also undecidable for CSL. ◀

Given that all the problems become undecidable for $\mathcal{C} = \text{CSL}$, we now focus our investigation on classes of languages strictly contained in the class of CSLs.

4 Problems 1 and 2

For the rest of this section, assume that $|w| = m$ and $|\Sigma| = \sigma$. Let us begin by noting that Problems 1 and 2 can be solved in polynomial time for the class REG following the approach of Theorem 15. Indeed, in this case, we assume that L is specified by the NFA A, with s

states, with $L(A) = L$, and then we either have to check the emptiness of the intersection of $L = L(A)$ with the language accepted by the deterministic automaton constructed in Lemma 12, or, respectively, with the complement of this language; both these tasks clearly take polynomial time.

We now consider the two problems for the class of CFLs. We first recall the following folklore lemma (see, e.g., [35]).

▶ **Lemma 17.** *Let $G = (V, \Sigma, P, S)$ be a CFG in CNF, and let $A = (Q, \Sigma, q_0, F, \delta)$ be a DFA. Then we can construct in polynomial time a CFG G_A in CNF such that $L(G_A) = L(G) \cap L(A)$.*

We can now state the main result of this section. We can apply Lemma 17, for Problem 1, to the input CFG and the DFA constructed in Lemma 12, or, for Problem 2, to the input CFG and the complement of the respective DFA. In both cases, we compute a CFG in CNF generating the intersection of a CFL and a REG language, and we have to check whether the language generated by that grammar is empty or not; all these can be implemented in polynomial time.

▶ **Theorem 18.** *Problems 1 and 2, for an input grammar with n non-terminals and an input string of length m, are decidable in polynomial time for CFL.*

5 Problems 3 and 5

Let us begin by noting that in [4] it was shown that for a given NFA A with s states (with input alphabet Σ, where $|\Sigma| = \sigma$) and an integer $k \geq 0$, we can decide whether $L(A)$ contains a k-universal string (i.e., Problem 3 for the class REG) in time $\mathcal{O}(\text{poly}(s, \sigma)2^{\sigma})$; in other words, Problem 3 is fixed parameter tractable (FPT) w.r.t. the parameter σ. A polynomial time algorithm (running in $\mathcal{O}(\text{poly}(s, \sigma)$ time) was given for Problem 5, relying on the observation that, given an NFA A, the language $L(A)$ contains strings with arbitrarily large universality if and only if A contains a state q, which is reachable from the initial state and from which one can reach a final state, and a cycle which contains this state, whose label is 1-universal. Coming back to Problem 3 for REG, the same paper shows that it is actually NP-complete. This is proved by a reduction from the Hamiltonian Path problem (HPP, for short), in which a graph with n vertices, the input of HPP, is mapped to an input of Problem 3 consisting in an automaton with $\mathcal{O}(n^2)$ states over an alphabet of size n. This reduction also implies that, assuming that the Exponential Time Hypothesis (ETH, for short) holds, there is no $2^{o(\sigma)}\text{poly}(s, \sigma)$ time algorithm solving Problem 3 (as this would imply the existence of an $2^{o(n)}$ time algorithm solving HPP); see [53] for more details related to the ETH and HPP.

Further, we consider Problems 3 and 5 for the class CFL, and we assume that, in both cases, we are given a CFL L by a CFG $G = (V, \Sigma, P, S)$ in CNF, with n non-terminals, over an alphabet Σ, with σ letters, and an integer $k \geq 1$ (in binary representation).

To transfer the lower bound derived for Problem 3 in the case of REG (specified as NFAs) to the larger class of CFLs, we recall the folklore result that a CFG in CNF can be constructed in polynomial time from an NFA (by constructing a regular grammar from the NFA, and then putting the grammar in CNF, see [35]). So, the same reduction from [4] can be used to show that, assuming ETH holds, there is no $2^{o(\sigma)}\text{poly}(n, \sigma)$ time algorithm solving Problem 3. This reduction shows also that Problem 3 is NP-hard; whether this problem is in NP remains open.

We now focus on the design of a $2^{\mathcal{O}(\sigma)}\text{poly}(n,\sigma)$ time algorithm solving Problem 3 (which would also show that this problem is FPT) and show that Problem 5 can be solved in polynomial time. Let us recall that Problem 5 requires deciding whether $\iota_\exists(L)$ is finite, and, if yes, Problem 3 requires checking whether $\iota_\exists(L) \geq k$.

We start by introducing a new concept which leads to a series of combinatorial observations.

▶ **Definition 19.** *Let $G = (V, \Sigma, P, S)$ be a CFG. A non-terminal $A \in V$ generates a* 1-universal cycle *if and only if there exists a derivation $A \Rightarrow^* w_1 A w_2$ of the grammar G with $w_1, w_2 \in \Sigma^*$ and $\max(\iota(w_1), \iota(w_2)) \geq 1$.*

We can show the following result.

▶ **Lemma 20.** *Let $G = (V, \Sigma, P, S)$ be a CFG in CNF and $L = L(G)$. Then $\iota_\exists(L)$ is infinite if and only if there exists a non-terminal $X \in V$ such that X generates a 1-universal cycle.*

Proof. Assume we have a non-terminal $A \in V$ which generates a *1-universal cycle*. This means that there exists a derivation $A \Rightarrow^* w_1 A w_2$ with $w_1, w_2 \in \Sigma^*$ and $\iota(w_1) \geq 1$ or $\iota(w_2) \geq 1$. As G is in CNF, we have that there exist $w'_A, w''_A \in \Sigma^*$ and the derivation $S \Rightarrow^* w'_A A w''_A$, and, also, that there exists $w_A \in \Sigma^*$ such that $A \Rightarrow^* w_A$. We immediately get that, for all $n \geq 1$, the following derivation is valid: $S \Rightarrow^* w'_A A w''_A \Rightarrow^* w'_A w_1 A w_2 w''_A \Rightarrow^* w'_A (w_1)^2 A (w_2)^2 w''_A \Rightarrow^* w'_A (w_1)^n A (w_2)^n w''_A \Rightarrow^* w'_A (w_1)^n w_A (w_2)^n w''_A = w$. As $\iota(w_1) \geq 1$ or $\iota(w_2) \geq 1$, it follows that $\iota(w) \geq n$. So, $\iota_\exists(L)$ is infinite.

We now show the converse implication. More precisely, we show by induction on the number of non-terminals of G that if $\iota_\exists(L(G))$ is infinite then G has at least one useful non-terminal $X \in V$ such that X has a 1-universal cycle. For this induction proof, we can relax the restrictions on G: more precisely, we still assume that the set P of productions of G fulfils $P \subseteq V \times (V^2 \cup \Sigma)$ but do not require that every non-terminal of G is useful; it suffices to require the starting symbol to be useful.

The result is immediate if G has a single non-terminal, i.e., the start symbol S. We now assume that our statement holds for CFLs generated by grammars with at most m non-terminals, and assume that L is a CFL generated by a CFG G with $m + 1$ non-terminals. We want to show that G has at least one useful non-terminal $X \in V$ such that X has a 1-universal cycle. We can assume, w.l.o.g., that S does not have a 1-universal cycle (otherwise, the result already holds).

Now, consider for each useful $A \in V \setminus \{S\}$ the CFG (which fulfills the requirements of our statement) $G_A = (V \setminus \{S\}, \Sigma, A, P')$, where P' is obtained from P by removing all productions involving S. Clearly, if there exists some $A \in V$ such that $\iota_\exists(L(G_A))$ is infinite, then, by induction, G_A contains a useful non-terminal $X \in V$ such that X has a 1-universal cycle. As G_A is obtained from G by removing some productions and one non-terminal, it is clear that X also has a 1-universal cycle in G and is also useful in G, so our statement holds. Let us now assume, for the sake of a contradiction, that, for each useful $A \in V$, there exists an integer $N_A \geq 1$ such that $\iota_\exists(L(G_A)) \leq N_A$. Take $N = 1 + \max\{N_A \mid A \in V\}$. As $\iota_\exists(L)$ is infinite, there exists a string $w \in L(G)$ with $\iota(w) \geq 2N + 3$. Since $w \in L(G)$, $S \Rightarrow^* w$ holds.

Let T_S be the derivation tree of w with root S and note that all non-terminals occurring in T_S are useful. Let p the longest simple path of T_S starting in S and having the end-node S (in the case when there are more such paths, we simply choose one of them). We denote by T_S^p the sub-tree of T_S rooted in the end-node of p. If w' is the string obtained by reading the leaves of T_S^p left-to-right, then we have the following derivation corresponding to T_S: $S \Rightarrow v_S S v'_S \Rightarrow^* v_S w' v'_S = w$, where $v_S, v'_S \in \Sigma^*$. Since, by our assumption, S does not have a 1-universal cycle, we get that $\iota(v_S) = 0$, $\iota(v'_S) = 0$, and that $\iota(w') \geq 2N + 1$.

Further, we consider T_S^p, and note that no other node of this tree, except the root, is labelled with S. Assume that the first step in the derivation $S \Rightarrow^* w'$ is $S \Rightarrow AB$, for some non-terminals $A, B \in V$ and production $S \to AB$, and that the children of the root S in the tree T_S^p are the sub-trees T_A and T_B. Let w_A be the border of T_A and w_B be the border of T_B. Clearly, it follows that at least one of the strings w_A and w_B is N-universal. We can assume, w.l.o.g., that $\iota(w_A) \geq N$. But $w_A \in L(G_A)$ and $\iota_\exists(L(G_A)) < N$ (by the definition of N). This is a contradiction with our assumption that $\iota(L(G_X))$ is finite, for all $X \in V$. So, there exists $X \in V$ for which $\iota_\exists(L(G_X))$ is infinite and, as we have seen, this means that our statement holds. ◀

So, according to Lemma 20, if the CFG G, which is the input of our problem, contains at least one non-terminal $X \in V$ which has a 1-universal cycle, we answer positively the instances of Problems 3 and 5 defined by G and, in the case of Problem 3, additionally by an integer $k \geq 1$. Next, we show that one can decide in polynomial time whether such a non-terminal exists in a grammar. However, if G does not contain any non-terminal with a 1-universal cycle, while the instance of Problem 5 is already answered negatively, it is unclear how to answer Problem 3. To address this, we try to find a way to efficiently construct a string of maximal universality index, and, for that, we need another combinatorial result.

▶ **Lemma 21.** *Let $G = (V, \Sigma, P, S)$ be a CFG in CNF, with $|V| = n$, $|\Sigma| = \sigma$, and $L = L(G)$. Furthermore, assume $\iota_\exists(L)$ is finite. There exists a string w of L with $\iota(w) = \iota_\exists(L)$ such that the derivation tree of w has depth at most $4n\sigma$.*

Proof. Let $w_0 \in L$ be a string such that $\iota(w_0) = \iota_\exists(L)$, and let T_0 be its derivation tree. Assume that T_0 has depth greater than $4n\sigma$. Then there exists a simple-path p in T_0 from the root to a leaf of length at least $4n\sigma + 1$ (i.e., contains $4n\sigma + 2$ nodes on it). By the pigeonhole-principle, there is one non-terminal $A \in V$ which occurs at least 4σ times on this path. Therefore, there exists the derivation $S \Rightarrow^* v_0 A v_0' \Rightarrow^* v_0 v_1 A v_1' v_0' \Rightarrow^* \ldots \Rightarrow^* v_0 v_1 \cdots v_{4\sigma-1} A v_{4\sigma-1}' \cdots v_1' v_0' \Rightarrow^* v_0 v_1 \cdots v_{4\sigma-1} w_0' v_{4\sigma-1}' \cdots v_1' v_0' = w_0$, with $v_0, v_0', \ldots, v_{4\sigma-1}, v_{4\sigma-1}', w_0' \in \Sigma^*$.

As $\iota_\exists(L)$ is finite, by Lemma 20, A has no 1-universal cycle, so $\iota(v_1 \cdots v_{4\sigma-1}) = \iota(v_{4\sigma-1}' \cdots v_1') = 0$.

We now go with i from 1 to $4\sigma - 1$ and construct a set M_ℓ as follows. For this we use the rest of the arch factorization of a word $r(\cdot)$, which is the suffix not associated with any of the arches of the respective word. We maintain a set U, which is initialized with $\mathrm{alph}(r(v_0))$; we also initialize $M_\ell = \emptyset$. Then, when considering i, if $\mathrm{alph}(v_i) \not\subseteq U$, we let $U \leftarrow U \cup \mathrm{alph}(v_i)$ and $M_\ell \leftarrow M_\ell \cup \{i\}$; before moving on and repeating this procedure for $i + 1$, if $U = \Sigma$, we set $U \leftarrow \emptyset$. Let us note that, during this process, because $\iota(v_1 \cdots v_{4\sigma-1}) = 0$, we set $U \leftarrow \emptyset$ at most once. Also, since M_ℓ is updated only when $\mathrm{alph}(v_i) \not\subseteq U$, it means that M_ℓ is updated at most $2\sigma - 2$ times. So $|M_\ell| \leq 2\sigma - 2$.

Similarly, to construct a set M_r, for i from $4\sigma - 1$ downto 1, we maintain a set U, initialized with $\mathrm{alph}(r(v_0 v_1 \cdots v_{4\sigma-1} w_0'))$; we also initialize $M_r = \emptyset$. Then, when considering i, if $\mathrm{alph}(v_i') \not\subseteq U$, we let $U \leftarrow U \cup \mathrm{alph}(v_i')$ and $M_r \leftarrow M_r \cup \{i\}$; before moving on and repeating this procedure for $i - 1$, if $U = \Sigma$, we set $U \leftarrow \emptyset$. As before, we get that M_r is updated at most $2\sigma - 2$ times, and $|M_r| \leq 2\sigma - 2$.

It is worth noting that the indices stored in M_ℓ and M_r indicate the strings v_i and v_i', respectively, which contain letters that are relevant when computing the arch factorization of w_0. The indices not contained in these sets indicate strings v_i or v_i', respectively, which are simply contained in an arch, and all the letters of these strings already appeared in that arch before the start of v_i and v_i', respectively.

As $|M_\ell| + |M_r| \leq 4\sigma - 4$, we get that there exists $i \in [1, 4\sigma]$ such that $i \notin M_\ell \cup M_r$. It is now immediate that the derivation $S \Rightarrow^* v_0 A v_0' \Rightarrow^* v_0 v_1 A v_1' v_0' \Rightarrow^* \ldots \Rightarrow^* v_0 v_1 \cdots v_{i-1} A v_{i-1}' \cdots v_1' v_0' \Rightarrow^* v_0 \cdots v_{i-1} v_{i+1} A v_{i+1}' v_{i-1}' \cdots v_0' \Rightarrow^* v_0 \cdots v_{i-1} v_{i+1} \cdots v_{4\sigma-1} w_0' v_{4\sigma-1}' \cdots v_{i+1}' v_{i-1}' \cdots v_0' = w_1$ produces a string w_1 such that $\iota(w_1) = \iota(w_0)$; let T_1 be the tree corresponding to this derivation. Clearly, the total length of the simple-paths connecting the root to leaves in the derivation tree T_1 is strictly smaller than the total length of the simple-paths connecting the root to leaves in the tree T_0. If T_1 still has root-to-leaf simple-paths of length at least $4n\sigma$, we can repeat this process and obtain a tree where the total length of the simple-paths connecting the root to leaves is even smaller. This process is repeated as long as we obtain trees having at least one root-to-leaf simple-path of length at least $4n\sigma$. Clearly, this is a finite process, whose number of iterations is bounded by, e.g., the sum of the length of root-to-leaf simple-paths of T_0. When we obtain a tree T where all root-to-leaf simple paths are of length at most $4n\sigma$, we stop and note that the border of this tree is a string w, with $\iota(w) = \iota_\exists(L)$. This concludes our proof. ◀

We now come to the algorithmic consequences of our combinatorial lemmas. For both considered problems the language given as input is in the form of a CFG $G = (V, \Sigma, P, S)$ in CNF with $|\Sigma| = \sigma \geq 2$ and $|V| = n$. Firstly, we show that Problem 5 can be decided in polynomial time.

▶ **Theorem 22.** *Problem 5 can be solved in $\mathcal{O}(\max(n^3, n^2\sigma))$ time.*

Proof Sketch. By Lemma 20, it is enough to check whether G contains a non-terminal $X \in V$ such that X has a 1-universal cycle. More precisely, we want to check if there exists a non-terminal X such that $X \Rightarrow^* wXw'$, where $\mathrm{alph}(w) = \Sigma$ or $\mathrm{alph}(w') = \Sigma$. We only show how to decide if there is a non-terminal X such that $X \Rightarrow^* wXw'$, where $\mathrm{alph}(w) = \Sigma$ (the case when $\mathrm{alph}(w') = \Sigma$ is similar). The main observation is that such a non-terminal $X \in V$ exists if and only if G contains, for some non-terminal X, derivations $X \Rightarrow^* w_a X w_a'$, with $w_a \in \Sigma^* a \Sigma^*$ and $w_a' \in \Sigma^*$, for all $a \in \Sigma$. Determining the existence of such a non-terminal is done in several steps. Firstly, we identify in $\mathcal{O}(n^3)$-time all pairs of non-terminals $A, B \in V$ with $A \Rightarrow^* \alpha B \beta$, for some $\alpha, \beta \in \Sigma^*$. Then, using the previously computed pairs, in $\mathcal{O}(n^2\sigma)$, we identify all pairs A, a, with $A \in V$ and $a \in \Sigma$, with $A \Rightarrow^* \alpha a \beta$, for some $\alpha, \beta \in \Sigma^*$. Now, in $\mathcal{O}(n^3)$ time, we identify all pairs of non-terminals $A, B \in V$, such that there exist a production $A \to BC$ in G and a derivation $C \Rightarrow^* \alpha A \beta$, with $\alpha, \beta \in \Sigma^*$. Finally, using all the sets of pairs that we have computed, we can identify all pairs of A, a, of non-terminals and terminals of G, respectively, such that there exists a derivation $A \Rightarrow^* \alpha a \beta A \gamma$, with $\alpha, \beta, \gamma \in \Sigma^*$. We conclude that there exists a non-terminal $X \in V$ for which we have derivations $X \Rightarrow^* w_a X w_a'$, with $w_a \in \Sigma^* a \Sigma^*$ and $w_a' \in \Sigma^*$, for all $a \in \Sigma$, if and only if there exists such a non-terminal X where the pairs X, a were found in the last step of our approach, for all $a \in \Sigma$. ◀

Further, we show that Problem 3 is FPT w.r.t. the parameter σ; this also means that the respective problem is solvable in polynomial time for constant-size alphabets. Recall that there is an ETH-conditional lower bound of $2^{o(\sigma)}\mathrm{poly}(n, \sigma)$ for the time complexity of algorithms solving this problem.

▶ **Theorem 23.** *Problem 3 can be solved in $\mathcal{O}(2^{4\sigma} n^5 \sigma^2)$ time.*

Proof Sketch. Recall that now we also get as input an integer $k > 0$ (given in binary representation).

To solve Problem 3, we check, using the algorithm from Theorem 22, whether $\iota_\exists(L)$ is finite. If $\iota_\exists(L)$ is infinite, then we answer the given instance positively. Otherwise, we proceed as follows.

We use a dynamic programming approach to compute the maximal universality index of a string of L. This essentially uses the result of Lemma 21 which states that such a string is the border of a derivation tree of depth at most $N = 4n\sigma$. More precisely, we construct a 4-dimensional matrix $M[\cdot,\cdot,\cdot,\cdot]$, with elements $M[i, A, S_p^A, S_s^A]$ with $A \in V$, $S_p^A, S_s^A \subsetneq \Sigma$ and $i \le N$. By definition, $M[i, A, S_p^A, S_s^A] = \ell$ if ℓ is the maximum number with the property that there exists a string w, which labels the border of a derivation tree of height at most i rooted in A, so that w has a prefix x, with $\mathrm{alph}(x) = S_p^A$, followed by ℓ arches, and a suffix y with $\mathrm{alph}(y) = S_s^A$.

To compute the elements $M[i,\cdot,\cdot,\cdot]$ for $i = 1$ it is enough to consider the productions of the form $A \to a$. For each such production, we only have to set $M[1, A, \{a\}, \emptyset] \leftarrow 0$ and $M[1, A, \emptyset, \{a\}] \leftarrow 0$.

To compute $M[i,\cdot,\cdot,\cdot]$ for $i > 1$, we consider every production $A \to BC$, and try to combine derivation trees of height at most $i-1$ and obtain derivation trees of height i. This computation is structured in two phases (corresponding to two cases).

The first phase corresponds to first of the cases we need to consider. Namely, in this case, we produce trees of height at most i whose borders have 0 arches, by combining trees of height at most $i-1$ with the same property. For that, we iterate over the productions $A \to BC$, and sets $S_1, S_2, S_3, S_4 \subsetneq \Sigma$ such that $M[i-1, B, S_1, S_2] = 0$ and $M[i-1, C, S_3, S_4] = 0$. If $S_1 \cup S_2 \cup S_3 \cup S_4 \subsetneq \Sigma$, we set $M[i, A, S_1 \cup S_2 \cup S_3 \cup S_4, \emptyset] = 0$ and $M[i, A, \emptyset, S_1 \cup S_2 \cup S_3 \cup S_4] = 0$. If $S_1 \cup S_2 \cup S_3 \subsetneq \Sigma$, we set $M[i, A, S_1 \cup S_2 \cup S_3, S_4] = 0$. If $S_1 \cup S_2 \subsetneq \Sigma$ and $S_3 \cup S_4 \subsetneq \Sigma$, we set $M[i, A, S_1 \cup S_2, S_3 \cup S_4] = 0$. Finally, if $S_2 \cup S_3 \cup S_4 \subsetneq \Sigma$, we set $M[i, A, S_1, S_2 \cup S_3 \cup S_4] = 0$. Note that the elements of $M[i,\cdot,\cdot,\cdot]$ set in this step might still be updated in the following. Moreover, the case when we can join two trees of height at most $i-1$ whose borders have 0 arches, and obtain a tree of height i whose border has one arch is also considered in the following.

The second case (and corresponding phase of our computation) is, therefore, the one where we produce trees of height at most i whose borders have at least one arch, by combining trees of height at most $i-1$. In this case, we iterate over the productions $A \to BC$, and sets $S_p^A, S_s^A \subsetneq \Sigma$. Now, for every pair $R, R' \subsetneq \Sigma$, with $R \cup R' = \Sigma$, a possible candidate for $M[i, A, S_p^A, S_s^A]$, corresponding to the production $A \to BC$, is obtained by adding $M[i-1, B, S_p^A, R]$ and $M[i-1, C, R', S_s^A]$, and add one for the new arch. We then take the maximum over all these combinations of alphabets R and R'. We get

$$c_{A\to BC} \leftarrow \max(\{-\infty\} \cup \{M[i-1, B, S_p^A, R] + M[i-1, C, R', S_s^A] + 1 \mid R, R' \subsetneq \Sigma, R \cup R' = \Sigma\}).$$

If A has t productions $p_1, \ldots, p_t$ we compute all values $c_{p_1}, \ldots, c_{p_t}$. Then $M[i, A, S_p^A, S_s^A]$ is set to be the maximum of the current value of $M[i, A, S_p^A, S_s^A]$ (as potentially computed in the first phase), $c_{p_1}, \ldots, c_{p_t}$, and $M[i-1, A, S_p^A, S_s^A]$.

For each i, this process (covering both cases) requires $\mathcal{O}(n^3 2^{4\sigma})$ algorithm-steps, in the worst case. So the entire matrix M is computed in $\mathcal{O}(2^{4\sigma} n^4 \sigma)$ algorithm-steps, where each algorithm-step might require $\mathcal{O}(n\sigma)$-time (as it can involve arithmetical operations on numbers with $\mathcal{O}(n\sigma)$ bits). We obtain, in the end, the complexity from the statement. Note that the matrix $M[\cdot,\cdot,\cdot,\cdot]$ computed by our algorithm has $O(2^{2\sigma} n^2)$ entries, so the space used by our algorithm is exponential.

Then, $\iota_\exists(L)$ equals the maximum over the entries of $M[4n\sigma, S, \emptyset, R]$, over all subsets $R \subsetneq \Sigma$, as we only consider strings w that lie in L, so strings that can be derived from S. The answer of the given instance of Problem 3 is positive if and only if $k \le \iota_\exists(L)$. ◀

The algorithm from Theorem 23 uses exponential space (due to the usage of the matrix M). However, there is also a simple (non-deterministic) PSPACE-algorithm solving this problem. Such an algorithm constructs non-deterministically the left derivation (where, at each step, the leftmost non-terminal is rewritten) of a string $w \in L$ with $\iota(w) \geq k$; w is non-deterministically guessed, and it is never constructed or stored explicitly by our algorithm. During this derivation of w the number of non-terminals in each sentential form is upper bounded by the depth of its derivation tree [35]; due to Lemma 21, we thus can have only $4n\sigma$ such non-terminals (if this number becomes larger, we stop and reject: the derivation tree of the guessed derivation is too deep for our purposes). During the simulation of the leftmost derivation, at step i, we also do not keep track of the maximal prefix w'_i consisting only of terminals of the sentential form, but only of $\iota(w'_i)$, $\mathrm{alph}(r(w'_i))$, and of the maximal suffix w''_i consisting of non-terminals only (i.e., the part we still need to process); this is enough for computing the universality of the derived string. The information stored by our algorithm clearly fits in polynomial space. If, and only if, at the end of the derivation, the maintained universality index is at least k, we accept the input grammar and number k.

6 Problem 4

Let us note that deciding Problem 4 for some input language L and integer k is equivalent to deciding whether $\iota_\forall(L) \geq k$. In [4], it was shown that for a regular language L over an alphabet with σ letters, accepted by an NFA with s states, Problem 4 can be decided in $\mathcal{O}(s^3\sigma)$.

For the rest of this section, we consider Problem 4 for the class CFL, and we assume that we are given a CFL L by a CFG G in CNF, with n non-terminals, over an alphabet Σ, with $\sigma \geq 2$ letters. Recall that our approach is to compute $\iota_\forall(L)$ and compare it with the input integer k.

As before, we start with a combinatorial observation. Intuitively, when we try to find a word with the lowest universality index, it is enough to consider words w, whose derivation trees do not contain root-to-leaf paths which contain twice the same non-terminal (otherwise, such a tree could be reduced, to a derivation tree of a word with potentially lower universality index).

▶ **Lemma 24.** *If $w \in L$ is a string with $\iota(w) \leq \iota(w')$, for all $w' \in L$, then there exists a string $w'' \in L$ with $\iota(w'') = \iota(w)$ and the derivation tree of w'' has depth at most n.*

We now show that we can compute $\iota_\forall(L)$ in polynomial time, when the input language is a CFL.

▶ **Theorem 25.** *Problem 4 can be solved in $\mathcal{O}(n^4\sigma^2)$ time.*

Proof Sketch. In order to compute $\iota_\forall(L)$, it is enough to compute the smallest $\ell \in \mathbb{N}$ for which there exists $w \in L$ having an absent subsequence of length ℓ (and then we conclude that $\iota_\forall(L) = \ell - 1$).

Our approach to computing $\iota_\forall(L)$ is, therefore, to define a 4-dimensional matrix M whose elements are $M[i, A, a, b]$, with $i \in [n]$, $A \in V$, $a \in \Sigma \cup \{\varepsilon\}, b \in \Sigma$. We define $M[i, A, a, b] = \ell$ if and only if there exists a word $w \in \Sigma^*$ such that $A \Rightarrow^* w$ and this derivation has an associated tree of depth at most i, and any $\mathrm{SAS}_{a,b}(w)$ has length ℓ. Based on Lemma 24, the elements of M can be computed by dynamic programming, by considering i from 1 to n, in $\mathcal{O}(n^4\sigma^2)$.

Once all elements of M are computed, we note that $\iota_\forall(L)$ is obtained by subtracting 1 from the minimum element of the form $M[n, S, a, b]$, with $a \in \Sigma \cup \{\varepsilon\}$ and $b \in \Sigma$. ◀

7 What Next? Conclusions and First Steps Towards Future Work

A conclusion of this work is that the complexity of the approached problems is, to a certain extent, similar when the input language is from the classes REG and CFL and they all become undecidable for CSL. So, a natural question is whether there are classes of languages (defined by corresponding classes of grammars or automata) between REG and CSL which exhibit a different, interesting behaviour.

We commence here this investigation by considering the class of languages accepted by a model of automata, namely, the deterministic finite automata with translucent letters (or, for short, translucent (finite) automaton – TFA), which generalizes the classical DFA by allowing the processing of the input string in an order which is not necessarily the usual sequential left-to-right order (without the help of an explicit additional storage unit). These automata, first considered in [60] (see also the survey [63] for a discussion on their properties and motivations), are strictly more powerful than classical finite automata and are part of a class of automata-models that are allowed to jump symbols in their processing, e.g., see [58] or [15]. From our perspective, these automata and the class of languages they accept are interesting because, on the one hand, they seem to be a generalization of regular languages which is orthogonal to the classical generalization provided by context-free languages, and, on the other hand, initial results suggest that the problems considered in this paper, not only become harder for them, but also their decidability fills the gap between the polynomial time solubility in the case of CFLs and that of undecidability for the class of CSL.

So, in what follows, we discuss some problems from Section 1 in relation to the TFA model, following the formalization from [59].

▶ **Definition 26.** *A TFA M is a tuple $M = (Q, \Sigma, q_0, F, \delta)$, just as in the case of DFA. However, the processing of inputs is not necessarily sequential. We define the partial relation $\circlearrowright$ on the set $Q \times \Sigma^*$ of configurations of M: $(p, xay) \circlearrowright_M (q, xy)$ if $\delta(p, a) = q$, and $\delta(p, b)$ is not defined for any $b \in alph(x)$, where $p, q \in Q$, $a, b \in \Sigma$, $x, y \in \Sigma^*$. The subscript M is omitted when it is understood from the context. The reflexive and transitive closure of $\circlearrowright$ is $\circlearrowright^*$ and the language accepted by M is defined as $L(M) = \{w \in \Sigma^* \mid (q_0, w) \circlearrowright^* (f, \varepsilon) \text{ for some } f \in F\}$.*

In this model, letters a such that $\delta(p, a)$ is not defined are called translucent for p, hence the name of the model. The machine reads and erases from the tape the letters of the input one-by-one. Note that the definition requires that every letter of the input is read before it can be accepted. This is slightly different from the original definition [60], which did not require all of the letters read, and used an unerasable endmarker on the tape. TFA by our definition can be trivially simulated by a machine with the original definition, and our results stand for the original model, too. We chose to follow the definitions in [59], because in our opinion it is simpler (and simpler to argue), and illustrates the difficulty of the subsequence matching problems for nonsequential machine models just as well.

A first observation is that, in terms of execution, in each step a TFA reads (and consumes) the leftmost unconsumed symbol which allows a transition (i.e., that has not been previously read, and there is a transition labeled with it from the current state). Therefore, for every individual letter, the order of the processing of its occurrences in the TFA is that in which they appear in a string. The non-deterministic version of this automata model accepts all rational trace languages, and all accepted languages have semi-linear Parikh images. Moreover, the class of languages accepted by this model is incomparable to the class of CFL, while still being CS. The class of languages accepted by the more restrictive deterministic finite automata with translucent letters, for short TFA, strictly includes the class REG and

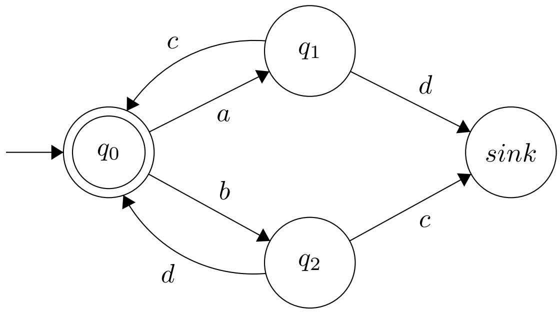

Figure 1 TFA that accepts the language $w \sqcup\!\sqcup\, h(w)$, where $w \in \{a,b\}^*$ and h is a morphism of the form $h(a) = c$, $h(b) = d$.

is still incomparable with CFL and the above mentioned class of rational trace languages. The recent survey [63] overviews the extensive literature regarding variations of these types of machines.

▶ **Example 27.** The TFA in Figure 1 accepts the language $L = w \sqcup\!\sqcup\, h(w)$, where $w \in \{a,b\}^*$ and $h : \{a,b\}^* \to \{c,d\}^*$ is a morphism given by $h(a) = c$, $h(b) = d$. Here $\sqcup\!\sqcup$ denotes the usual shuffle operation for words over some alphabet Σ, i.e., $u \sqcup\!\sqcup\, v = \{u_1v_1 \cdots u_\ell v_\ell \mid u = u_1 \cdots u_\ell, v = v_1 \cdots v_\ell, u_i \in \Sigma^* \text{ for } i \in [\ell], v_i \in \Sigma^* \text{ for } i \in [\ell]\}$; in our case, $\Sigma = \{a,b,c,d\}$.

Coming back to the TFA in Figure 1: in state q_0, the machine can read only the first a or b remaining on the tape and immediately matches it with the first c or d, respectively. If it reads a and in the remaining input the first d comes before the first c, it goes into the sink state. Similarly, if it reads b but the first remaining c is before the first remaining d, it goes to sink, because the projection of the input to the $\{a,b\}$ alphabet does not match the projection to the $\{c,d\}$ alphabet. The language L is not context-free. This can be, indeed, seen by intersecting it with the regular language $(a+b)^*(c+d)^*$, which yields the language $\{w \cdot h(w) \mid w \in \{a,b\}^*\}$, a variant of the so called "copy language". This language is non-context-free, by an easy application of the Bar-Hillel pumping lemma, so L is not context-free.

We first note that the class of languages accepted by TFA becomes incomparable to that of CFLs only starting from the ternary alphabet case (see [61]), since, for a TFA over a binary alphabet, one can construct a push-down automaton accepting the same language.

▶ **Theorem 28.** *The languages accepted by TFA over binary alphabets are CF.*

As a consequence of this and of the results shown in the previous sections we get the following.

▶ **Theorem 29.** *Over binary alphabets, all problems of Section 1 are decidable, and except for Problem 3, all are decidable in polynomial time for a TFA A given as input.*

Thus, our interest now shifts to languages accepted by TFAs, over alphabets Σ of size $\sigma \geq 3$. We report here a series of initial results, which suggest this to be a worthwhile direction of investigation. We first note that we cannot apply the approach from the general Theorem 15 to solve the problems considered, since one can encode the solution set of any Post Correspondence Problem (for short, PCP) instance as the intersection of a regular language and a language accepted by a TFA. This is a first significant difference w.r.t. the status of the approached problems for the case of REG and CFL.

▶ **Theorem 30.** *The emptiness problem for languages defined as the intersection of the language accepted by a TFA with a regular language (given as finite automaton) is undecidable.*

The decidability of Problem 1 for larger alphabets in the case of TFA is settled by an exponential time brute force algorithm, after establishing that if the input language contains a supersequence of w, then it also contains one whose length is bounded by a polynomial in the size of the input. By a reduction from the well-known NP-complete Hamiltonian Cycle Problem [27], we can also show that Problem 1 for TFA is NP-hard over unbounded alphabets (containment in NP follows from the same length upper bound mentioned earlier). This is again a significant deviation w.r.t. the status of this problem for the case when the input language is given by a finite automaton or by a CFG.

▶ **Theorem 31.** *Problem 1 is NP-complete over unbounded alphabets.*

Since our initial results deviate from the corresponding results obtained for CFL, without suggesting that the considered problems become undecidable, completing this investigation for all other problems seems worthwhile to us. While we have excluded the approach from the general Theorem 15, we cannot yet say anything about the approach in Theorem 14. It remains an interesting open problem (also of independent interest w.r.t. to our research) to obtain an algorithm for computing the downward closure of a TFA-language, or show that such an algorithm does not exist.

While studying the problems discussed in this paper for TFAs seems an interesting way to understand their possible further intricacies, which cause the huge gap between their status for CFL and CSL, respectively, another worthwhile research direction is to consider them in the context of other well-motivated classes of languages, for which all these problems are decidable, and try to obtain optimised algorithms in those cases.

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015. doi:10.1109/FOCS.2015.14.
2. Amir Abboud and Aviad Rubinstein. Fast and deterministic constant factor approximation algorithms for LCS imply new circuit lower bounds. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 35:1–35:14, 2018. doi:10.4230/LIPIcs.ITCS.2018.35.
3. Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 39–51, 2014. doi:10.1007/978-3-662-43948-7_4.
4. Duncan Adamson, Pamela Fleischmann, Annika Huch, Tore Koß, Florin Manea, and Dirk Nowotka. k-universality of regular languages. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation (ISAAC 2023)*, volume 283 of *LIPIcs*, pages 4:1–4:21, 2023. Full version: https://arxiv.org/abs/2311.10658. doi:10.4230/LIPIcs.ISAAC.2023.4.
5. Duncan Adamson, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Longest common subsequence with gap constraints. In Anna E. Frid and Robert Mercas, editors, *Combinatorics on Words - 14th International Conference, WORDS 2023, Umeå, Sweden, June 12-16, 2023, Proceedings*, volume 13899 of *Lecture Notes in Computer Science*, pages 60–76. Springer, 2023. doi:10.1007/978-3-031-33180-0_5.

6 Ashwani Anand and Georg Zetzsche. Priority downward closures. In Guillermo A. Pérez and Jean-François Raskin, editors, *34th International Conference on Concurrency Theory, CONCUR 2023, September 18-23, 2023, Antwerp, Belgium*, volume 279 of *LIPIcs*, pages 39:1–39:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CONCUR.2023.39.

7 Alexander Artikis, Alessandro Margara, Martín Ugarte, Stijn Vansummeren, and Matthias Weidlich. Complex event recognition languages: Tutorial. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017*, pages 7–10, 2017. doi:10.1145/3093742.3095106.

8 Ricardo A. Baeza-Yates. Searching subsequences. *Theoretical Computer Science*, 78(2):363–376, 1991. doi:10.1016/0304-3975(91)90358-9.

9 Laura Barker, Pamela Fleischmann, Katharina Harwardt, Florin Manea, and Dirk Nowotka. Scattered factor-universality of words. In Nataša Jonoska and Dmytro Savchuk, editors, *Developments in Language Theory*, LNCS, pages 14–28, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-48516-0_2.

10 Lasse Bergroth, Harri Hakonen, and Timo Raita. A survey of longest common subsequence algorithms. In Pablo de la Fuente, editor, *Seventh International Symposium on String Processing and Information Retrieval, SPIRE 2000, A Coruña, Spain, September 27-29, 2000*, pages 39–48, , 2000. doi:10.1109/SPIRE.2000.878178.

11 Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and David Kofoed Wind. String matching with variable length gaps. *Theoretical Computer Science*, 443:25–34, 2012. doi:10.1016/j.tcs.2012.03.029.

12 Karl Bringmann and Bhaskar Ray Chaudhury. Sketching, Streaming, and Fine-Grained Complexity of (Weighted) LCS. In Sumit Ganguly and Paritosh Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*, volume 122 of *LIPIcs*, pages 40:1–40:16, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.40.

13 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proc. SODA 2018*, pages 1216–1235, 2018. doi:10.1137/1.9781611975031.79.

14 Sam Buss and Michael Soltys. Unshuffling a square is NP-hard. *Journal of Computer and System Sciences*, 80(4):766–776, 2014. doi:10.1016/j.jcss.2013.11.002.

15 Hiroyuki Chigahara, Szilárd Zsolt Fazekas, and Akihiro Yamamura. One-way jumping finite automata. *International Journal of Foundations of Computer Science*, 27(3):391–405, 2016.

16 Vaclav Chvatal and David Sankoff. Longest common subsequences of two random sequences. *Journal of Applied Probability*, 12(2):306–315, 1975. URL: http://www.jstor.org/stable/3212444.

17 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.

18 Maxime Crochemore, Borivoj Melichar, and Zdeněk Troníček. Directed acyclic subsequence graph — overview. *Journal of Discrete Algorithms*, 1(3-4):255–280, 2003. doi:10.1016/S1570-8667(03)00029-7.

19 Joel D. Day, Pamela Fleischmann, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. The Edit Distance to k-Subsequence Universality. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*, volume 187 of *LIPIcs*, pages 25:1–25:19, 2021. doi:10.4230/LIPIcs.STACS.2021.25.

20 Joel D. Day, Maria Kosche, Florin Manea, and Markus L. Schmid. Subsequences with gap constraints: Complexity bounds for matching and analysis problems. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation, ISAAC 2022, December 19-21, 2022, Seoul, Korea*, volume 248 of *LIPIcs*, pages 64:1–64:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ISAAC.2022.64.

21 Lukas Fleischer and Manfred Kufleitner. Testing Simon's congruence. In Igor Potapov, Paul Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117 of *LIPIcs*, pages 62:1–62:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.62.

22 Pamela Fleischmann, Sungmin Kim, Tore Koß, Florin Manea, Dirk Nowotka, Stefan Siemer, and Max Wiedenhöft. Matching patterns with variables under simon's congruence. In Olivier Bournez, Enrico Formenti, and Igor Potapov, editors, *Reachability Problems - 17th International Conference, RP 2023, Nice, France, October 11-13, 2023, Proceedings*, volume 14235 of *Lecture Notes in Computer Science*, pages 155–170. Springer, 2023. doi:10.1007/978-3-031-45286-4_12.

23 Michael L. Fredman and Dan E. Willard. BLASTING through the information theoretic barrier with FUSION TREES. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 1–7. ACM, 1990. doi:10.1145/100216.100217.

24 Dominik D. Freydenberger, Pawel Gawrychowski, Juhani Karhumäki, Florin Manea, and Wojciech Rytter. Testing k-binomial equivalence. *https://arxiv.org/abs/1509.00622*, pages 239–248, 2015.

25 André Frochaux and Sarah Kleest-Meißner. Puzzling over subsequence-query extensions: Disjunction and generalised gaps. In Benny Kimelfeld, Maria Vanina Martinez, and Renzo Angles, editors, *Proceedings of the 15th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW 2023), Santiago de Chile, Chile, May 22-26, 2023*, volume 3409 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3409/paper3.pdf.

26 Emmanuelle Garel. Minimal separators of two words. In Alberto Apostolico, Maxime Crochemore, Zvi Galil, and Udi Manber, editors, *Combinatorial Pattern Matching, 4th Annual Symposium, CPM 93, Padova, Italy, June 2-4, 1993, Proceedings*, volume 684 of *LNCS*, pages 35–53. Springer, 1993. doi:10.1007/BFB0029795.

27 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., USA, 1990.

28 Pawel Gawrychowski, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Efficiently testing Simon's congruence. In *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, pages 34:1–34:18, 2021. doi:10.4230/LIPIcs.STACS.2021.34.

29 Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. Complex event recognition in the big data era: a survey. *The VLDB Journal*, 29(1):313–352, 2020. doi:10.1007/s00778-019-00557-w.

30 Hermann Gruber, Markus Holzer, and Martin Kutrib. More on the size of Higman-Haines sets: Effective constructions. *Fundamenta Informaticae*, 91(1):105–121, 2009. doi:10.3233/FI-2009-0035.

31 Simon Halfon, Philippe Schnoebelen, and Georg Zetzsche. Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '17, pages 1–12, 2017. doi:10.1109/LICS.2017.8005141.

32 Jean-Jacques Hébrard. An algorithm for distinguishing efficiently bit-strings by their subsequences. *Theoretical Computer Science*, 82:35–49, 1991.

33 Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 3(1):326–336, 1952.

34 Daniel S. Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM*, 24(4):664–675, 1977. doi:10.1145/322033.322044.

35 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

36 James W. Hunt and Thomas G. Szymanski. A fast algorithm for computing longest subsequences. *Communications of the ACM*, 20(5):350–353, 1977. doi:10.1145/359581.359603.

37 P. Karandikar, M. Kufleitner, and P. Schnoebelen. On the index of Simon's congruence for piecewise testability. *Information Processing Letters*, 115(4):515–519, 2015. doi:10.1016/J.IPL.2014.11.008.

38 Prateek Karandikar and Philippe Schnoebelen. The height of piecewise-testable languages and the complexity of the logic of subwords. *Logical Methods in Computer Science*, 15(2), 2019. doi:10.23638/LMCS-15(2:6)2019.

39 Sarah Kleest-Meißner, Rebecca Sattler, Markus L. Schmid, Nicole Schweikardt, and Matthias Weidlich. Discovering event queries from traces: Laying foundations for subsequence-queries with wildcards and gap-size constraints. In Dan Olteanu and Nils Vortmeier, editors, *25th International Conference on Database Theory, ICDT 2022, March 29 to April 1, 2022, Edinburgh, UK (Virtual Conference)*, volume 220 of *LIPIcs*, pages 18:1–18:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICDT.2022.18.

40 Sarah Kleest-Meißner, Rebecca Sattler, Markus L. Schmid, Nicole Schweikardt, and Matthias Weidlich. Discovering multi-dimensional subsequence queries from traces - from theory to practice. In Birgitta König-Ries, Stefanie Scherzinger, Wolfgang Lehner, and Gottfried Vossen, editors, *Datenbanksysteme für Business, Technologie und Web (BTW 2023), 20. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 06.-10, März 2023, Dresden, Germany, Proceedings*, volume P-331 of *LNI*, pages 511–533. Gesellschaft für Informatik e.V., 2023. doi:10.18420/BTW2023-24.

41 Maria Kosche, Tore Koß, Florin Manea, and Viktoriya Pak. Subsequences in bounded ranges: Matching and analysis problems. In Anthony W. Lin, Georg Zetzsche, and Igor Potapov, editors, *Reachability Problems - 16th International Conference, RP 2022, Kaiserslautern, Germany, October 17-21, 2022, Proceedings*, volume 13608 of *Lecture Notes in Computer Science*, pages 140–159. Springer, 2022. doi:10.1007/978-3-031-19135-0_10.

42 Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Absent subsequences in words. In Paul C. Bell, Patrick Totzke, and Igor Potapov, editors, *Reachability Problems - 15th International Conference, RP 2021, Liverpool, UK, October 25-27, 2021, Proceedings*, volume 13035 of *LNCS*, pages 115–131, 2021. doi:10.1007/978-3-030-89716-1_8.

43 Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Absent subsequences in words. *Fundam. Informaticae*, 189(3-4):199–240, 2022. doi:10.3233/FI-222159.

44 Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Combinatorial algorithms for subsequence matching: A survey. In Henning Bordihn, Géza Horváth, and György Vaszil, editors, *Proceedings 12th International Workshop on Non-Classical Models of Automata and Applications, NCMA 2022, Debrecen, Hungary, August 26-27, 2022*, volume 367 of *EPTCS*, pages 11–27, , 2022. doi:10.4204/EPTCS.367.2.

45 Dietrich Kuske. The subtrace order and counting first-order logic. In Henning Fernau, editor, *Computer Science – Theory and Applications*, volume 12159 of *LNCS*, pages 289–302, 2020. doi:10.1007/978-3-030-50026-9_21.

46 Dietrich Kuske and Georg Zetzsche. Languages ordered by the subword order. In Mikołaj Bojańczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures*, LNCS, pages 348–364, 2019. doi:10.1007/978-3-030-17127-8_20.

47 Martin Lange and Hans Leiß. To CNF or not to CNF? an efficient yet presentable version of the CYK algorithm. *Informatica Didact.*, 8, 2009. URL: http://ddi.cs.uni-potsdam.de/InformaticaDidactica/LangeLeiss2009.

48 Marie Lejeune, Julien Leroy, and Michel Rigo. Computing the k-binomial complexity of the Thue-Morse word. In Piotrek Hofman and Michał Skrzypczak, editors, *Developments in Language Theory*, LNCS, pages 278–291, 2019. doi:10.1007/978-3-030-24886-4_21.

49 Julien Leroy, Michel Rigo, and Manon Stipulanti. Generalized Pascal triangle for binomial coefficients of words. *The Electronic Journal of Combinatorics*, 24(1.44):36 pp., 2017.

50 Chun Li and Jianyong Wang. *Efficiently Mining Closed Subsequences with Gap Constraints*, pages 313–322. SIAM, 2008. doi:10.1137/1.9781611972788.28.

51 Chun Li, Qingyan Yang, Jianyong Wang, and Ming Li. Efficient mining of gap-constrained subsequences and its various applications. *ACMTransactions on Knowledge Discovery from Data*, 6(1):2:1–2:39, 2012. doi:10.1145/2133360.2133362.

52 Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complex. Cryptol.*, 4(2):241–299, 2012. doi:10.1515/GCC-2012-0016.

53 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bull. EATCS*, 105:41–72, 2011. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/92.

54 David Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25(2):322–336, April 1978. doi:10.1145/322063.322075.

55 Florin Manea, Jonas Richardsen, and Markus L. Schmid. Subsequences with generalised gap constraints: Upper and lower complexity bounds. In Shunsuke Inenaga and Simon J. Puglisi, editors, *35th Annual Symposium on Combinatorial Pattern Matching, CPM 2024, June 25-27, 2024, Fukuoka, Japan*, volume 296 of *LIPIcs*, pages 22:1–22:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.CPM.2024.22.

56 William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980. doi:10.1016/0022-0000(80)90002-1.

57 Alexandru Mateescu, Arto Salomaa, and Sheng Yu. Subword histories and Parikh matrices. *Journal of Computer and System Sciences*, 68(1):1–21, 2004. doi:10.1016/J.JCSS.2003.04.001.

58 Alexander Meduna and Petr Zemek. Jumping finite automata. *International Journal of Foundations of Computer Science*, 23(7):1555–1578, 2012. doi:10.1142/S0129054112500244.

59 Victor Mitrana, Andrei Păun, Mihaela Păun, and José-Ramón Sánchez-Couso. Jump complexity of finite automata with translucent letters. *Theoretical Computer Science*, 992:114450, 2024. doi:10.1016/J.TCS.2024.114450.

60 Benedek Nagy and Friedrich Otto. Finite-state acceptors with translucent letters. In A.O. De La Puente G. Bel-Enguix, V. Dahl, editor, *BILC 2011: AI Methods for Interdisciplinary Research in Language and Biology*, pages 3–13, 2011.

61 Benedek Nagy and Friedrich Otto. Globally deterministic CD-systems of stateless R-automata with window size 1. *International Journal of Computer Mathematics*, 90(6):1254–1277, 2013. doi:10.1080/00207160.2012.688820.

62 Narao Nakatsu, Yahiko Kambayashi, and Shuzo Yajima. A longest common subsequence algorithm suitable for similar text strings. *Acta Informatica*, 18:171–179, 1982. doi:10.1007/BF00264437.

63 Friedrich Otto. A survey on automata with translucent letters. In Benedek Nagy, editor, *Implementation and Application of Automata*, pages 21–50, Cham, 2023. doi:10.1007/978-3-031-40247-0_2.

64 Rohit Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966. doi:10.1145/321356.321364.

65 William E. Riddle. An approach to software system modelling and analysis. *Computer Languages*, 4(1):49–66, 1979. doi:10.1016/0096-0551(79)90009-2.

66 Michel Rigo and Pavel Salimov. Another generalization of abelian equivalence: Binomial complexity of infinite words. *Theoretical Computer Science*, 601:47–57, 2015. doi:10.1016/J.TCS.2015.07.025.

67 Arto Salomaa. Connections between subwords and certain matrix mappings. *Theoretical Computer Science*, 340(2):188–203, 2005. doi:10.1016/J.TCS.2005.03.024.

68 Philippe Schnoebelen and Julien Veron. On arch factorization and subword universality for words and compressed words. In Anna Frid and Robert Mercaş, editors, *Combinatorics on Words*, volume 13899, pages 274–287, 2023. doi:10.1007/978-3-031-33180-0_21.

69 Shinnosuke Seki. Absoluteness of subword inequality is undecidable. *Theoretical Computer Science*, 418:116–120, 2012. doi:10.1016/J.TCS.2011.10.017.

70 Alan C. Shaw. Software descriptions with flow expressions. *IEEE Transactions on Software Engineering*, 4(3):242–254, 1978. doi:10.1109/TSE.1978.231501.

71 Imre Simon. *Hierarchies of events with dot-depth one - Ph.D. thesis*. University of Waterloo, 1972.

72 Imre Simon. Piecewise testable events. In H. Brakhage, editor, *Automata Theory and Formal Languages*, LNCS, pages 214–222, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg. doi:10.1007/3-540-07407-4_23.

73 Imre Simon. Words distinguished by their subwords (extended abstract). In *Proc. WORDS 2003*, volume 27 of *TUCS General Publication*, pages 6–13, 2003.

74 Zdeněk Troníček. Common subsequence automaton. In Jean-Marc Champarnaud and Denis Maurel, editors, *Conference on Implementation and Application of Automata*, volume 2608 of *LNCS*, pages 270–275, 2002. doi:10.1007/3-540-44977-9_28.

75 Jan van Leeuwen. Effective constructions in well-partially- ordered free monoids. *Discrete Mathematics*, 21(3):237–252, 1978. doi:10.1016/0012-365X(78)90156-5.

76 Georg Zetzsche. The Complexity of Downward Closure Comparisons. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *LIPIcs*, pages 123:1–123:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.123.

77 Georg Zetzsche. Separability by piecewise testable languages and downward closures beyond subwords. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 929–938. ACM, 2018. doi:10.1145/3209108.3209201.

78 Haopeng Zhang, Yanlei Diao, and Neil Immerman. On complexity and optimization of expensive queries in complex event processing. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 217–228, 2014. doi:10.1145/2588555.2593671.