Matching Patterns with Variables Under Simon's Congruence

Pamela Fleischmann[1], Sungmin Kim[3], Tore Koß[2], Florin Manea[2], Dirk Nowotka[1], Stefan Siemer[2], and Max Wiedenhöft[1]

[1] Department of Computer Science, Kiel University, Germany,
`{fpa,dn,maw}@informatik.uni-kiel.de`
[2] Department of Computer Science, University of Göttingen, Germany
`{tore.koss,florin.manea,stefan.siemer}@cs.uni-goettingen.de`
[3] Department of Computer Science, Yonsei University, Republic of Korea
`rena_rio@yonsei.ac.kr`

Abstract. We introduce and investigate a series of matching problems for patterns with variables under Simon's congruence. Our results provide a thorough picture of these problems' computational complexity.

1 Introduction

A *pattern with variables* is a string α consisting of *constant letters* (or *terminals*) from a finite alphabet $\Sigma = \{1, \ldots, \sigma\}$, of size $\sigma \geq 2$, and *variables* from a potentially infinite set $\mathcal{X}$, with $\Sigma \cap \mathcal{X} = \emptyset$. Such a pattern α is mapped by a function h, called *substitution*, to a word by substituting the variables occurring in α by arbitrary strings of constants, i.e., strings over Σ. For example, the pattern $\alpha = xx\mathsf{abab}yy$ can be mapped to the string of constants $\mathsf{aaaaababbb}$ by the substitution h defined by $h(x) = \mathsf{aa}, h(y) = \mathsf{b}$. In this framework, $h(\alpha)$ denotes the word obtained by substituting every occurrence of a variable x in α by $h(x)$ and leaving all the constants unchanged. If a pattern α can be mapped to a string of constants w, we say that α matches w; the problem of deciding, given a pattern α with variables and a string of constants w, whether there exists a substitution which maps α to w is called the *(exact) matching problem*, `Match`.

Exact Matching Problem: `Match`(α, w)	
Input:	Pattern α, $|\alpha| = m$, word w, $|w| = n$.
Question:	Is there a substitution h with $h(\alpha) = w$?

`Match` is a heavily studied problem, which appears frequently in various areas of theoretical computer science. Initially, this problem was considered in language theory (e.g., pattern languages [5]) or combinatorics on words (e.g., unavoidable patterns [52]), with connections to algorithmic learning theory (e.g., the theory of descriptive patterns for finite sets of words [5, 16, 63]), and has by now found interesting applications in string solving and the theory of word equations ([51]), stringology (e.g., generalised function matching [4]), the theory of extended regular expressions with backreferences [8, 23, 28, 29]), or database theory (mainly in relation to document spanners [15, 24, 26, 27, 46, 58, 59]).

Match is NP-complete in general [5], and a more detailed image of the parameterised complexity of the matching problem is revealed in [17–19, 55, 57, 62] and the references therein. A series of classes of patterns, defined by structural restrictions, for which Match is in P were identified [13, 17, 55]; moreover, for most of these classes, Match is $W[1]$-hard [14] with respect to the structural parameters used to define the respective classes. Recently, Gawrychowski et al. [34, 35] studied Match in an approximate setting: given a pattern α, a word w, and a natural number ℓ, one has to decide if there exists a substitution h such that $D(h(\alpha), w) \leq \ell$, where D is either the Hamming [34] or the edit distance [35]. Their results offered, once more, a detailed understanding of the approached matching problems' complexity (in general, and for classes of patterns defined by structural restrictions). The problems discussed in [34, 35] can be seen in a more general setting: given a pattern α and a word w, decide if there exists a substitution h such that $h(\alpha)$ is similar to w, with respect to some similarity measure (Hamming resp. edit distance in [34, 35] or string equality for exact Match). Thus, it seems natural to also consider various other string-equivalence relations as similarity measures, such as $(k\text{-})$abelian equivalence [41, 42] or k-binomial equivalence [25, 50, 56]. Here, we consider an approximate variant of Match using Simon's congruence $\sim_k$ [65].

Matching under Simon's Congruence: MatchSimon(α, w, k)
Input: Pattern α, $|\alpha| = m$, word w, $|w| = n$, and number $k \in [n]$.
Question: Is there a substitution h with $h(\alpha) \sim_k w$?

Let us recall the definition of Simon's congruence. A string u is a *subsequence* of a string v if u results from v by deleting some letters of v. Subsequences are well studied in the area of combinatorics of words and combinatorial pattern matching, and are well-connected to other areas of computer science (e.g., the handbook [51] or the survey [48] and the references therein). Let $\mathbb{S}_k(v)$ be the set of all subsequences of a given string v up to length $k \in \mathbb{N}_0$. Two strings v and v' are k-Simon congruent iff $\mathbb{S}_k(v) = \mathbb{S}_k(v')$. The problem of testing whether two given strings are k-Simon congruent, for a given k, was introduced by Imre Simon in his PhD thesis [64] as a similarity measure for strings, and was intensely studied in the combinatorial pattern matching community (see [11, 20, 30, 38, 66, 67] and the references therein), before being optimally solved in [6, 32]. Another interesting extension of these results, discussed in [45], brings us closer to the focus of this paper. There, the authors present an efficient solution for the following problem: given two words w, u and a natural number k, decide whether there exists a factor of w which is k-Simon congruent to u; this is MatchSimon with the input pattern $\alpha = xuy$ for variables x, y. Thus, it seems natural to consider, in a general setting, the problem of checking whether one can map a given pattern α to a string which is similar to w with respect to $\sim_k$. Moreover, there is another way to look at this problem, which seems interesting to us: the input word w and the number k are a succinct representation of $\mathbb{S}_k(w)$. So, MatchSimon(α, w, k) asks whether or not we can assign the variables of α in such a way that we reach a word describing the target set of subsequences of length k, as well.

One of the congurence-classes of Σ^* w.r.t. $\sim_k$ received a lot of attention: the class of k-subsequence universal words, those words which contain all k-length words as subsequences. This class was first studied in [40, 60], and further investigated in [1, 2, 6, 12, 21, 22, 47, 61] in contexts related to and motivated by formal languages, automata theory, or combinatorics, where the notion of universality is central (see [7, 9, 33, 36, 49, 53, 54] for examples in this direction). The motivation of studying k-subsequence universal words is thoroughly discussed in [12]. Here, we consider the following problem:

Matching a Target Universality: `MatchUniv`(α, k)
Input: Pattern α, $|\alpha| = m$, and $k \in \mathbb{N}_0$.
Question: Is there a substitution h with $\iota(h(\alpha)) = k$?

In this problem, $\iota(w)$ (the universality index of w) is the largest integer ℓ for which w is ℓ-subsequence universal. Note that `MatchUniv` can be formulated in terms of `MatchSimon`: the answer to `MatchUniv`(α, k) is yes if and only if the answer to `MatchSimon`$(\alpha, (1 \cdots \sigma)^k, k)$ is yes and the answer to `MatchSimon`$(\alpha, (1 \cdots \sigma)^{k+1}, k+1)$ is no. However, there is an important difference: for `MatchUniv` we are not explicitly given the target word w, whose set of k-length subsequences we want to reach; instead, we are given the number k which represents the target set more compactly (using only $\log k$ bits).

In the problems introduced above, we attempt to match (or reach), starting with a pattern α, the set of subsequences defined by a given word w (given explicitly or implicitly). A well-studied extension of `Match` is the satisfiability problem for word equations, where we are given two patterns α and β and are interested in finding an assignment of the variables that maps both patterns to the same word (see, e.g., [51]). This problem is central both to combinatorics on words and to the applied area of string solving [3, 37]. In this paper, we extend `MatchSimon` to the problem of solving word equations under $\sim_k$, defined as follows.

Word Equations under Simon's Congruence: `WESimon`(α, β, k)
Input: Patterns α, β, $|\alpha| = m$, $|\beta| = n$, and $k \in [m+n]$.
Question: Is there a substitution h with $h(\alpha) \sim_k h(\beta)$?

Besides introducing these natural problems, our paper presents a rather comprehensive picture of their computational complexity. We start with `MatchUniv`, the most particular of them and whose input is given in the most compact way. In Section 3 we show that `MatchUniv` is NP-complete, and also present a series of structurally restricted classes of patterns, for which it can be solved in polynomial time. In Section 4, we approach `MatchSimon` and show that it is also NP-complete; some other variants of this problem, both tractable and intractable, are also discussed. Finally, in Section 5, we discuss `WESimon` and its variants, and characterise their computational complexity. The paper ends with a section pointing to a series of future research directions.

2   Preliminaries

Let $\mathbb{N} = \{1, 2, \ldots\}$ be the set of natural numbers. Let $[n] = \{1, \ldots, n\}$ and $[m : n] = [n] \setminus [m-1]$, for $m, n \in \mathbb{N}, m < n$. $\mathbb{N}_0$ denotes $\mathbb{N} \cup \{0\}$.

For a finite set $\Sigma = [\sigma]$ called *alphabet*, Σ^* denotes the set of all words (or strings) over Σ, with ε denoting the empty word. For $w \in \Sigma^*$, $|w|$ denotes its length, while $|w|_{\mathtt{a}}$ denotes the number of occurrences of $\mathtt{a} \in \Sigma$ in w. Further, $\Sigma^{\leq k}$ (resp. Σ^k) denotes the set of all words over Σ up to (resp. of) length $k \in \mathbb{N}$. Let $w[i]$ denote the ith letter in the string w, and let $\mathtt{alph}(w) = \{\mathtt{a} \mid |w|_{\mathtt{a}} \geq 1\}$ denote the set of different letters in w. To access the first occurrence of a letter $\mathtt{a} \in \Sigma$ after a position $i \in [|w|]$ in a word $w \in \Sigma^*$, define the *X-ranker* as a mapping $\mathtt{X} : \Sigma^* \times ([|w|] \cup \{0, \infty\}) \times \Sigma \to [|w|] \cup \{\infty\}$ with $(w, i, \mathtt{a}) \mapsto \min(\{j \in [i+1 : |w|] \mid w[j] = \mathtt{a}\} \cup \{\infty\})$ (cf. [68]). Notice that a lookup table for all possible X-ranker evaluations for some given $w \in \Sigma^*$ can be computed in linear time in $|w|$, where each item can be accessed in constant time [6, 20]. In the special case of $\mathtt{X}(w, 0, \mathtt{a})$, we call this occurrence of $\mathtt{a}$ the *signature letter* $\mathtt{a}$ of w, for all $\mathtt{a} \in \mathtt{alph}(w)$. A *permutation* γ of an alphabet Σ is a string in Σ^σ with $\mathtt{alph}(\gamma) = \Sigma$. A string u is a *subsequence* of a string w if there exists a strictly increasing integer sequence $0 < i_1 < i_2 < \ldots < i_{|u|} \leq |w|$ with $w[i_j] = u[j]$ for all $j \in [|u|]$. For a given $k \in \mathbb{N}_0$, we use $\mathbb{S}_k(w)$ as the set of all subsequences of w with length at most k. A subsequence u of w is called a *substring* of w if there exists a position i of w such that $u = w[i]w[i+1]\cdots w[i+|u|-1]$. We write $w[i : j]$ for $w[i]w[i+1]\cdots w[j]$ for $1 \leq i \leq j \leq |w|$. Substrings $w[1 : j]$ (resp., $w[i : |w|]$) are called *prefixes* (resp., *suffixes*) of w.

Two words $w_1, w_2 \in \Sigma^*$ are called *Simon k-congruent* ($w_1 \sim_k w_2$) if $\mathbb{S}_k(w_1) = \mathbb{S}_k(w_2)$ [65]. A word $w \in \Sigma^*$ is called *k-subsequence universal* (or k-universal for short) for some $k \in \mathbb{N}$ if $\mathbb{S}_k(w) = \Sigma^{\leq k}$; this means that $w \sim_k (1 \cdots \sigma)^k$. The largest $k \in \mathbb{N}_0$ such that w is k-universal is the *universality index* of w, denoted by $\iota(w)$. In [38], Hébrard introduced the following unique factorisation of words.

Definition 1. *The* arch factorisation *of a word $w \in \Sigma^*$ is defined by $w = \mathtt{arch}_1(w) \cdots \mathtt{arch}_k(w)\mathtt{rest}(w)$ for some $k \in \mathbb{N}_0$ such that there exists a sequence $(i_j)_{j \leq k}$ with $i_0 = 0$, $i_j = \max\{\mathtt{X}(w, i_{j-1}, \mathtt{a}) \mid \mathtt{a} \in \Sigma\}$ for all $j \geq 1$, $\mathtt{arch}_j(w) = w[i_{j-1} + 1 : i_j]$ whenever $1 \leq i_j < \infty$, and $\mathtt{rest}(w) = w[i_j : |w|]$, if $i_{j+1} = \infty$.*

Clearly, the number of arches of $w \in \Sigma^*$ is exactly $\iota(w)$. Extending the notion of arch factorisation, we define the arches and rest of $w \in \Sigma^*$ for $\mathtt{a} \in \mathtt{alph}(w)$ (cf. the arch jumping functions introduced in [61]) as well as the universality index for the respective letter $\mathtt{a}$. That is, we perform the arch factorisation and obtain the universality index for the suffix of w that starts after the first occurrence of $\mathtt{a}$.

Definition 2. *Let $w \in \Sigma^*$, $\mathtt{a} \in \mathtt{alph}(w)$, and $j \in [\iota(w)]$. The arches of signature letters are defined by $\mathtt{arch}_{\mathtt{a},j}(w) = \mathtt{arch}_j(w[\mathtt{X}(w, 0, \mathtt{a}) + 1 : |w|])$ and $\mathtt{rest}_{\mathtt{a}}(w) = \mathtt{rest}(w[\mathtt{X}(w, 0, \mathtt{a}) + 1 : |w|])$. The universality index of $\mathtt{a}$ is $\iota_{\mathtt{a}}(w) = \iota(w[\mathtt{X}(w, 0, \mathtt{a}) + 1 : |w|])$. The last index with respect to w of $\mathtt{arch}_{\mathtt{a},j}(w)$ is defined as $\mathtt{archEnd}_{\mathtt{a},j}(w) = \mathtt{X}(w, 0, \mathtt{a}) + \sum_{i=1}^{j} |\mathtt{arch}_{\mathtt{a},i}(w)|$.*

Now, we are interested in the smallest substrings of w that allow the completion of rests of specific prefixes of w to full arches. Hence, we define *marginal sequences*, which are breadth-first orderings of σ parallel arch factorisations, each starting after a signature letter of the word.
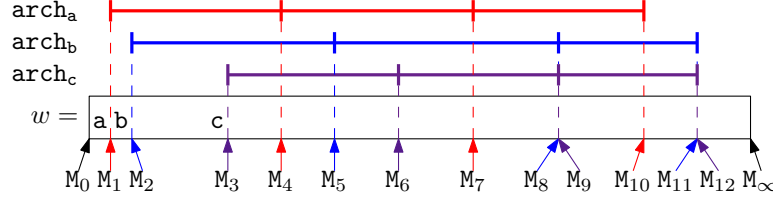

Fig. 1. The marginal sequence of a word.

Definition 3. *Let $w \in \Sigma^*$ and γ be a permutation of Σ such that $\mathtt{X}(w, 0, \gamma[i])$ is increasing w.r.t. $i \in [\sigma]$. From the arches for signature letters, we define the marginal sequence of integers of $w \in \Sigma^*$ inductively by $\mathtt{M}_0(w) = 0$, $\mathtt{M}_i(w) = \mathtt{X}(w, 0, \gamma[i])$ for all $i \in [\sigma]$, and $\mathtt{M}_{i\sigma+j}(w) = \mathtt{archEnd}_{\gamma[j],i}(w)$ for $j \in [\sigma]$, $i \in [\iota_{\gamma[j]}(w)]$. Let $\mathtt{M}_\infty(w) = |w|$ denote the last element of the sequence.*

The sequence is called *marginal* because, for $j \in [\sigma]$, $w[\mathtt{M}_{i\sigma+j-1}(w) + 1 : \mathtt{M}_{i\sigma+j}(w)]$ is the smallest prefix p of $w[\mathtt{M}_{i\sigma+j-1}(w) + 1 : |w|]$ such that $\iota_{\gamma[j]}(w[1 : \mathtt{M}_{i\sigma+j-1}(w)]p) = i$. Note that the marginal sequence $\mathtt{M}_i(w)$ is non-decreasing. In the following, we define a slight variation of the *subsequence universality signature* $\mathtt{s}(w)$ introduced in [61].

Definition 4. *1. For $w \in \Sigma^*$, the subsequence universality signature $\mathtt{s}(w)$ of w is defined as the 3-tuple $(\gamma, \mathcal{K}, \mathcal{R})$ with a permutation γ of $\mathtt{alph}(w)$, where $\mathtt{X}(w, 0, \gamma[i]) > \mathtt{X}(w, 0, \gamma[j]) \Leftrightarrow i > j$ (γ consists of the letters of $\mathtt{alph}(w)$ in order of their first appearance in w) and two arrays $\mathcal{K}$ and $\mathcal{R}$ of length σ with $\mathcal{K}[i] = \iota_{\gamma[i]}(w)$ and $\mathcal{R}[i] = \mathtt{alph}(\mathtt{rest}_{\gamma[i]}(w))$ for all $i \in [|\mathtt{alph}(w)|]$. For all $i \in [\sigma] \setminus \mathtt{alph}(w)$, we have $\mathcal{R}[i] = \Sigma$ and $\mathcal{K}[i] = -\infty$.*
2. Conversely, for a permutation γ' of Σ, an integer array $\mathcal{K}'$ and an alphabet array $\mathcal{R}'$ both of length σ, we say that the tuple $(\gamma', \mathcal{K}', \mathcal{R}')$ is a valid signature if there exists a string w that satisfies $\mathtt{s}(w) = (\gamma,' \mathcal{K}', \mathcal{R}')$.

Note that, for $k_i = \iota_{\gamma[i]}(w)$, we have $\mathcal{R}[i] = \mathtt{alph}(w[\mathtt{M}_{k_i\sigma+i}(w) + 1 : \mathtt{M}_\infty(w)])$, since $\mathtt{rest}_{\gamma[i]}(w) = w[\mathtt{M}_{k_i\sigma+i}(w) + 1 : \mathtt{M}_\infty(w)]$.

A central notion to this work is that of patterns with variables. From now on, we consider two alphabets: $\Sigma = [\sigma]$ is an alphabet of constants (or terminals), and $\mathcal{X}$ a (possibly infinite) alphabet of variables, with $\mathcal{X} \cap \Sigma = \emptyset$. A pattern α is a string from $(\mathcal{X} \cup \Sigma)^*$, i.e., a string containing both constants and variables. For a pattern α, $\mathtt{var}(\alpha) = \mathtt{alph}(\alpha) \cap \mathcal{X}$ denotes the set of *variables* in α, while $\mathtt{term}(\alpha) = \mathtt{alph}(\alpha) \cap \Sigma$ is the set of *constants (terminals)* in α.

Definition 5. *A* substitution $h : (\mathcal{X} \cup \Sigma)^* \to \Sigma^*$ *is a morphism that acts as the identity on* Σ *and maps each variable of* $\mathcal{X}$ *to a (potentially empty) string over* Σ. *That is,* $h(\mathtt{a}) = \mathtt{a}$ *for all* $\mathtt{a} \in \Sigma$ *and* $h(x) \in \Sigma^*$ *for all* $x \in \mathcal{X}$. *We say that pattern* α matches *string w over* Σ *under a binary relation* $\sim$ *if there exists a substitution h that satisfies* $h(\alpha) \sim w$.

In the above definition, if $\sim$ is the string equality $=$, we say that the pattern α *matches* the string w instead of saying that α *matches* w under $=$.

The problems addressed in this paper, introduced in Section 1, deal with matching patterns to words under Simon's congruence $\sim_k$. For these problems, the input consists of patterns, words, and a number k. In general, we assume that each letter of Σ appears at least once, in at least one of the input patterns or words. Therefore, for input pattern α and word w we assume that $\Sigma = \mathtt{term}(\alpha) \cup \mathtt{alph}(w)$. Hence, σ is upper bounded by the total length of the input words and patterns. Similarly, the total number of variables occurring in the input patterns is upper bounded by the total length of these patterns. However, in this paper, although the number of variables is not restricted, we assume that σ is a constant, i.e., $\sigma \in O(1)$. Clearly, the complexity lower bounds proven in this setting for the analysed problems are stronger while the upper bounds are weaker than in the general case, when no restriction is placed on σ. Note, however, that $\sigma \in O(1)$ is not an unusual assumption, being used in, e.g., [20].

The **computational model** we use is the Word RAM model with memory words of logarithmic size. This is a standard computational model in algorithm design in which, for an input of size n, the memory consists of memory-words consisting of $\Theta(\log n)$ bits. Basic operations (including arithmetic and bitwise Boolean operations) on memory-words take constant time, and any memory-word can be accessed in constant time. Numbers larger than n, with ℓ bits, are represented in $\Theta(\ell/\log n)$ memory words, and working with them takes time proportional to the number of memory words on which they are represented. In all the problems, we assume that we are given a pattern α, with $|\alpha| = n$, over a constant size alphabet of constants $\Sigma = \{1, 2, \ldots, \sigma\}$, with $\sigma \in O(1)$, and a set of variables $X := \{x_1, \ldots, x_n\}$ that can be encoded as integers between 1 and $\sigma + n$. That is, we assume that the processed patterns are sequences of integers (called letters or symbols), each fitting in $O(1)$ memory words. This is a common assumption in string algorithms: the input is said to be over *an integer alphabet*. For instance, the same assumption was also used for developing efficient algorithms for `Match` in [16, 34]. For a more detailed general discussion on this computational model see, e.g., [10].

3   `MatchUniv`

In this section, we discuss the `MatchUniv` problem. In this problem, we are given a pattern α and a natural number $k \leq n$, and we want to check the existence of a substitution h with $\iota(h(\alpha)) = k$. Note that $\iota(h(\alpha)) = k$ means both that $h(\alpha)$ is k-universal and that it is not $(k + 1)$-universal. A slightly relaxed version of

the problem, where we would only ask for $h(\alpha)$ to be k-universal is trivial (and, therefore, not interesting): the answer, in that case, is always positive, as it is enough to map one of the variables of α to $(1 \cdots \sigma)^k$. The main result of this section is that `MatchUniv` is NP-complete, which we will show in the following.

To show that `MatchUniv`(α, k) is NP-hard, we reduce `3CNFSAT` (3-satisfiability in conjunctive normal form) to `MatchUniv`(α, k). We provide several gadgets allowing us to encode a `3CNFSAT`-instance φ as an `MatchUniv`-instance (α, k). Finally, we show that we can find a substitution h for the instance (α, k), such that $\iota(h(\alpha)) = k$, if and only if φ is satisfiable. We begin by recalling `3CNFSAT`.

3-Satisfiability for formulas in conjunctive normal form, `3CNFSAT`.

Input: Clauses $\varphi := \{c_1, c_2, \ldots, c_m\}$, where $c_j = (y_j^1 \vee y_j^2 \vee y_j^3)$ for $1 \leq j \leq m$, and y_j^1, y_j^2, y_j^3 from a finite set of boolean variables $X := \{x_1, x_2, \ldots, x_n\}$ and their negations $\bar{X} := \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\}$.

Question: Is there an assignment for X, which satisfies all clauses of φ?

It is well-known that `3CNFSAT` is NP-complete (see [31, 43] for a proof). With this result at hand, we can prove the following lower bound.

Lemma 1. `MatchUniv` *is* NP-*hard.*

Proof. We reduce `3CNFSAT` to `MatchUniv`(α, k). Let us consider an instance of `3CNFSAT`: formula φ given by m clauses $\varphi := \{c_1, c_2, \ldots c_m\}$ over n variables $X := \{x_1, x_2, \ldots x_n\}$ (for simplicity in notation we define $N = n + m$). We map this `3CNFSAT` instance to an instance (α, k) of `MatchUniv`(α, k) with $k = 5n + m + 2$, the alphabet $\Sigma := \{0, 1, \#, \$\}$ and the variable set $\mathcal{X} := \{z_1, z_2, \ldots z_n, u_1, u_2, \ldots u_n\}$. More precisely, we want to show that there exists a substitution h to replace all the variables in α with constant words, such that $\iota(h(\alpha)) = 5n + m + 2$, if and only if the boolean formula φ is satisfiable. Our construction can be performed in polynomial time and is of polynomial size with respect to N. To present this construction, we will go through its building blocks, the so-called gadgets.

Before we start with these gadgets, let us introduce a renaming function for the variables $\rho : X \cup \bar{X} \to \mathcal{X}$ with $\rho(x_i) = z_i$ and $\rho(\bar{x}_i) = u_i$. Also, a substitution h which maps α to a string of universality index $5n + m + 2$ is called valid in the following.

The binarisation gadgets. We use the following gadgets to make the image of variables z_i and u_i under a valid substitution be strings over $\{0, 1\}$. Recall that we have the alphabet $\Sigma := \{0, 1, \#, \$\}$ and the set of variables $\mathcal{X} := \{z_1, z_2, \ldots, z_n, u_1, u_2, \ldots, u_n\}$.

At first, we construct the gadget $\pi_\# = (z_1 z_2 \cdots z_n u_1 u_2 \cdots u_n 01 \$)^{N^6} \#$, as shown in Figure 2. We observe that for all possible substitutions h, we have two cases for the universality of the image of this gadget. On the one hand, assume that any of the variables is substituted under h by a string that contains a $\#$. Then, the universality index of the image of this gadget will be $\iota(h(\pi_\#)) = k'$ with $k' \geq N^6 > k$, which is too big for a valid substitution. On the other hand, when all the variables are substituted under h by strings that do not contain $\#$, this

gadget is mapped to a string which is exactly one arch because there is only one # at its very end. Thus, under a valid substitution h, the images of the variables z_i and u_i do not contain #. Note also that, in the arch factorisation of such a string ($h(\pi_\#)$, where h is a valid substitution) we have one arch and no rest.

The gadget $\pi_\$ = (z_1 z_2 \cdots z_n u_1 u_2 \cdots u_n 01\#)^{N^6}\$$ is constructed analogously and can be seen in Figure 3. This enforces that under a valid substitution h, the images of the variables z_i and u_i do not contain \$.

In conclusion, the gadgets $\pi_\#$ and $\pi_\$$ enforce that under a valid substitution h, the image of the variables z_i and u_i contains only 0 and 1, i.e., they are binary strings.

$$\pi_\# = \quad (\quad \underbrace{z_1 z_2 \cdots z_n \quad u_1 u_2 \cdots u_n \quad 01\,\$}_{\text{No \# allowed}} \quad)^{N^6} \quad \#$$

Fig. 2. If any of the variables is substituted by a string that contains a #, then this gadget would add at least N^6 arches, which is already greater than the target universality $k = 5n + m + 2$.

$$\pi_\$ = \quad (\quad \underbrace{z_1 z_2 \cdots z_n \quad u_1 u_2 \cdots u_n \quad 01\,\#}_{\text{No \$ allowed}} \quad)^{N^6} \quad \$$$

Fig. 3. If any of the variables is substituted by a string that contains a \$, then this gadget would add at least N^6 arches, which is already greater than the target universality $k = 5n + m + 2$.

The Boolean gadgets. We use the following gadgets to force the image of each z_i and u_i to be either in 0^* or 1^*. Intuitively, mapping a variable z_i (respectively, u_i) to a string of the form 0^+ corresponds to mapping x_i (respectively, $\bar{x}_i$) to the Boolean value false (respectively, true). Similarly, mapping one of these string-variables to a string from 1^+ means mapping the corresponding boolean variable to true. For a beginning, these gadgets just have to enforce that the image of any string-variable does not contain both 0 and 1. We construct the gadget π_i^z (respectively π_i^u) for every string-variable z_i (respectively, u_i), according to Figure 4. More precisely, for all $i \in [n]$, we define two gadgets $\pi_i^z = (z_i\,\$\,\#)^{N^6} 1001\,\$\,\#$ and $\pi_i^u = (u_i\,\$\,\#)^{N^6} 1001\,\$\,\#$.

We now analyse the possible images of $\pi_i^z = (z_i\,\$\,\#)^{N^6} 1001\,\$\,\#$ under various substitutions h. There are three ways in which z_i can be mapped to a string by h. Firstly, if the image of z_i contains both 0 and 1, then for the universality

index of the image of π_i^z under the respective substitution is $\iota(h(\pi_i^z)) \geq N^6 > k$; such a substitution cannot be valid. Secondly, if the image of z_i is a string from 0^*, then the universality of this gadget is exactly $\iota(h(\pi_i^z)) = 2$ as shown in Figure 5. As a third option, if the image of z_i is a string from 1^*, then the universality of this gadget is exactly $\iota(h(\pi_i^z)) = 2$ as shown in Figure 6. As for the binarisation gadgets, in the arch factorisation of a string $h(\pi_i^z)$, where h is a valid substitution, we have exactly two arches (and no rest). A similar analysis can be performed for the gadgets $\pi_i^u = (u_i\,\$\,\#)^{N^6}\,1001\,\$\,\#$. In conclusion, the gadgets π_i^z and π_i^u enforce that under a valid substitution h, the image of the variables z_i and u_i contains either only 0s or only 1s (or is empty).

$$\pi_i^z = \ (\ z_i\ \$\,\#\ )^{N^6}\ \ 1\ \ 0\ \ 01\,\$\,\#$$

No 0 and 1 together allowed

Fig. 4. If any of the variables in the gadget π_i^z (analogous for π_i^u) is substituted by a string that contains both a 0 and a 1, then this gadget would add N^6 arches, which is already greater than the target universality $k = 5n + m + 2$.

$$\pi_i^z = \ (\ z_i\ \$\,\#\ )^{N^6}\ \ 1\ \ 0\ \ 01\,\$\,\#$$

arch if $z_i \in 0^+$

Fig. 5. If any of the variables π_i^z (analogous for π_i^u) consits of only 0's, this gadget would add 2 arches per variable.

$$\pi_i^z = \ (\ z_i\ \$\,\#\ )^{N^6}\ \ 1\ \ 0\ \ 01\,\$\,\#$$

arch if $z_i \in 1^*$

Fig. 6. If any of the variables π_i^z (analogous for π_i^u) consits of only 1's, this gadget would add 2 arches per variable.

The complementation gadgets. The role of these gadgets is to enforce the property that z_i and u_i are not both in 0^+ or not both in 1^+, for all $i \in [n]$. We construct the gadget $\xi_i = \$\,z_i u_i\,\#$, for every $i \in [n]$, according to Figure 7.

Let us now analyse the image of these gadgets under a valid substitution ($\pi_{\texttt{\#}}$ and $\pi_{\texttt{\$}}$ are mapped to exactly one arch each, and $\pi_{\texttt{i}}^{\texttt{z}}$ and $\pi_{\texttt{i}}^{\texttt{u}}$ are mapped to exactly two arches each). In this case, we observe that $\xi_{\texttt{i}}$ is mapped to exactly one complete arch ending on the rightmost symbol $\texttt{\#}$ if and only if the image of one of the variables $\texttt{z}_{\texttt{i}}$ and $\texttt{u}_{\texttt{i}}$ has at least one 0 and the image of the other one has at least one 1. Further, let us consider the concatenation of two consecutive such gadgets $\xi_{\texttt{i}}\xi_{\texttt{i+1}}$ and assume that both $\texttt{z}_{\texttt{i}}$ and $\texttt{u}_{\texttt{i}}$ are mapped to strings over the same letter or at least one of them is mapped to the empty word. In that case, the first arch must close to the right of the $\texttt{\$}$ letter in $\xi_{\texttt{i+1}}$, hence $\xi_{\texttt{i}}\xi_{\texttt{i+1}}$ could not contain two arches. Thus, the concatenation of the gadgets $\xi_1 \cdots \xi_n$ is mapped to a string which has exactly n arches if and only if each gadget ξ_i is mapped to exactly one arch, which holds if and only if the image of one of the variables $\texttt{z}_{\texttt{i}}$ and $\texttt{u}_{\texttt{i}}$ has at least one 0 and the image of the other one has at least one 1. When assembling together all the gadgets, we will ensure that, in a valid substitution, this property holds: $\texttt{z}_{\texttt{i}}$ and $\texttt{u}_{\texttt{i}}$ are mapped to repetitions of different letters.

$$\xi_{\texttt{i}} = \quad \texttt{\$} \quad \texttt{z}_{\texttt{i}} \quad \texttt{u}_{\texttt{i}} \quad \texttt{\#}$$

Fig. 7. In order for $\xi_{\texttt{i}}$ to contribute an arch, one of z_i and u_i has to be replaced by only 1's while the other must consist of only 0's.

The clause gadgets. Let $c_j = (y_j^1 \vee y_j^2 \vee y_j^3)$ be a clause, with $y_j^1, y_j^2, y_j^3 \in X \cup \bar{X}$. We construct the gadget $\delta_{\texttt{j}}$ for every clause c_j as $\texttt{\$}\,\texttt{0}\rho(\texttt{y}_{\texttt{j}}^1)\rho(\texttt{y}_{\texttt{j}}^2)\rho(\texttt{y}_{\texttt{j}}^3)\,\texttt{\#}$, as shown in Figure 8. Now, by all of the properties discussed for the previous gadgets, we can analyse the possible number of arches contained in the image of this gadget under a valid substitution. Firstly, note that if at least one of the variables $\rho(\texttt{y}_{\texttt{j}}^1), \rho(\texttt{y}_{\texttt{j}}^2), \rho(\texttt{y}_{\texttt{j}}^3)$ is mapped to a string containing at least one 1, then this gadget will contain exactly one arch ending on its rightmost symbol $\texttt{\#}$. Now consider the concatenation of two consecutive such gadgets $\delta_j\delta_{j+1}$, and assume that all the variables in δ_j are substituted by only 0s. In this case, the first arch must end to the right of the $\texttt{\$}$ symbol in δ_{j+1}, hence the string to which $\delta_{\texttt{j}}\delta_{\texttt{j+1}}$ is mapped could not contain two arches. The same argument holds if we look at the concatenation of the last complementation gadget and the first clause gadget, e.g. $\xi_n\delta_1$.

Thus, the concatenation of the gadgets $\delta_1 \cdots \delta_m$ is mapped to a string which has exactly m arches if and only if each gadget δ_i is mapped to exactly one arch. This holds if and only if at least one of the string-variables occurring in δ_i is mapped to a string of 1s. When assembling together all the gadgets, we will ensure that at least one of the variables occurring in each gadget δ_i, for all $i \in [m]$, is mapped to a string of 1s in a valid substitution.

$$\delta_\mathtt{j} = \quad \texttt{\$} \quad \texttt{0} \quad \rho(\mathtt{y}_\mathtt{j}^1) \quad \rho(\mathtt{y}_\mathtt{j}^2) \quad \rho(\mathtt{y}_\mathtt{j}^3) \quad \texttt{\#}$$

Fig. 8. In order for $\delta_\mathtt{j}$ to contribute an arch, at least one of $\rho(\mathtt{y}_\mathtt{j}^1)$, $\rho(\mathtt{y}_\mathtt{j}^2)$ and $\rho(\mathtt{y}_\mathtt{j}^3)$ has to be replaced by only 1's.

Final Assemblage. We finish the construction of the pattern α by concatenating all the gadgets. That is, $\alpha = \pi_\texttt{\#}\pi_\texttt{\$}\pi_1^\mathtt{z}\pi_1^\mathtt{u}\pi_2^\mathtt{z}\pi_2^\mathtt{u}\cdots\pi_\mathtt{n}^\mathtt{z}\pi_\mathtt{n}^\mathtt{u}\xi_1\xi_2\cdots\xi_\mathtt{n}\delta_1\delta_2\cdots\delta_\mathtt{m}$, as shown in Figure 9.

$$\alpha = \quad \pi_\texttt{\#} \quad \pi_\texttt{\$} \quad \underbrace{\pi_1^\mathtt{z}\pi_1^\mathtt{u}\pi_2^\mathtt{z}\pi_2^\mathtt{u}\cdots\pi_\mathtt{n}^\mathtt{z}\pi_\mathtt{n}^\mathtt{u}}_{\mathtt{4n}} \quad \underbrace{\xi_1\xi_2\cdots\xi_\mathtt{n}}_{\mathtt{n}} \quad \underbrace{\delta_1\delta_2\cdots\delta_\mathtt{m}}_{\mathtt{m}}$$

with $\pi_\texttt{\#}$ underbraced $\mathtt{1}$ and $\pi_\texttt{\$}$ underbraced $\mathtt{1}$.

Fig. 9. The concatenation of all gadgets and their respective amount of arches we expect, if we can find a substitution h with $\iota(h(\alpha)) = 5n + m + 2$.

The correctness of the reduction. We show that there exists a substitution h of the string variables of α with $\iota(h(\alpha)) = 5n + m + 2$ (i.e., a valid substitution) if and only if we can find an assignment for all Boolean-variables occurring in φ that satisfy all clauses $c_j \in \varphi$.

Let us first show that if there is a satisfying assignment for Boolean-variables of φ which makes the formula true, then there exists a substitution h of the string-variables of α such that $\iota(h(\alpha)) = 5n + m + 2$. In this case, we can give a canonical substitution h with $h(\rho(x_i)) = 1$ and $h(\rho(\bar{x}_i)) = 0$ if x_i is assigned true, and $h(\rho(x_i)) = 0$ and $h(\rho(\bar{x}_i)) = 1$ if x_i is assigned false. We can easily verify, by the definition of the gadgets, that under this substitution we have $\iota(h(\alpha)) = 5n + m + 2$. Indeed, in the images of each gadget $\pi_\texttt{\#}, \pi_\texttt{\$}, \pi_\mathtt{i}^\mathtt{z}, \xi_i$ and δ_i we have exactly one arch, ending on the last symbol of the respective strings, while in the image of each gadget $\pi_\mathtt{i}^\mathtt{z}$ under this substitution there will be exactly two arches, again ending on their last positions.

Conversely, we want to show that if we have a substitution h of the string-variables such that $\iota(h(\alpha)) = 5n + m + 2$, then there must be a satisfying assignment of the Boolean-variables for φ. The general idea is the following. We assume that we have a substitution of the string variables and compute the arch factorisation greedily and look at the properties enforced by the individual gadgets, as discussed above. Assume first, towards a contradiction, that the image of some variable $\mathtt{z_i}$ contains both 0 and 1 or that it contains $\texttt{\#}$ or $\texttt{\$}$. Then, as explained, the number of arches of the image of $\pi_\texttt{\#}\pi_\texttt{\$}\pi_1^\mathtt{z}\pi_1^\mathtt{u}\pi_2^\mathtt{z}\pi_2^\mathtt{u}\cdots\pi_\mathtt{n}^\mathtt{z}\pi_\mathtt{n}^\mathtt{u}$ will blow up to a value greater than $5n + m + 2$, a contradiction. The same reasoning holds for the variables $\mathtt{u_i}$. Therefore, each variable $\mathtt{z_i}$ is mapped to a string from $0^* \cup 1^*$, and the same holds for the variables $\mathtt{u_i}$. It follows that $h(\pi_\texttt{\#}\pi_\texttt{\$}\pi_1^\mathtt{z}\pi_1^\mathtt{u}\pi_2^\mathtt{z}\pi_2^\mathtt{u}\cdots\pi_\mathtt{n}^\mathtt{z}\pi_\mathtt{n}^\mathtt{u})$

contributes exactly $4n + 2$ arches to the arch factorisation of $h(\alpha)$, and the last arch of this factorisation (when identified greedily, from left to right) ends on the last letter of π_n^u (which is a # symbol). By this last property, we are guaranteed that we can look at the suffix $h(\xi_1\xi_2\cdots\xi_n\delta_1\delta_2\cdots\delta m)$ of our pattern's image under h separately, as no arch from the prefix $h(\pi_\#\pi_\$\pi_1^z\pi_1^u\pi_2^z\pi_2^u\cdots\pi_n^z\pi_n^u)$ extends in it. More precisely, this allows us to consider the subproblem of analysing h under the assumption that $\iota(h(\xi_1\xi_2\cdots\xi_n\delta_1\delta_2\cdots\delta m)) = m + n$, and, moreover, each string variable is mapped to strings from $0^* \cup 1^*$. In this subproblem, we have $n + m$ $ symbols in the pattern and we can not introduce new $ symbols in the image of the string-variables. Therefore, every $ symbol needs to be in exactly one arch. Now, as discussed when introducing the complementation and clause gadgets, we have to have the following properties, as otherwise we would have at least two $ symbols in the same arch and would only get to $k' < m + n$ arches overall. Firstly, one of each $h(\rho(x_i))$ and $h(\rho(\bar{x}_i))$ has to consist only of 0s while the other consists only of 1s, and both of them should have length at least 1. Secondly, at least one of each $\rho(y_j^1)$, $\rho(y_j^2)$ and $\rho(y_j^3)$ has to be substituted by a string from 1^+.

Given these properties, we can construct a satisfying assignment of the Boolean-variables from φ by setting a variable to be true if and only if their corresponding string-variable is mapped to a string from 1^*. As $h(\rho(x_i))$ and $h(\rho(\bar{x}_i))$ are mapped to strings over distinct alphabets, we get that x_i and $\bar{x}_i$ will be assigned distinct truth values. Moreover, at least one of each $\rho(y_j^1)$, $\rho(y_j^2)$ and $\rho(y_j^3)$ has to be substituted by a string from 1^+, so at least one variable per clause is assigned to true. Therefore, this assignment makes φ true.

This concludes our proof, and shows that $\mathtt{MatchUniv}(\alpha, k)$ is NP-hard.   □

In the following we show that $\mathtt{MatchUniv}(\alpha, k)$ is in NP. One natural approach is to guess the images of the variables occurring in the input pattern α under a substitution h and check whether or not $\iota(h(\alpha))$ is indeed k. However, it is difficult to bound the size of the images of the variables of α under h in terms of the size of α and $\log k$ (the size of our input), since the strings we look for may be exponentially long. For example, consider the pattern $\alpha = X_1$: the length of the shortest k-universal string is $k\sigma$ [6], which is already exponential in $\log k$. Therefore, we consider guessing only the subsequence universality signatures for the image of each variable under the substitution. We show that it is sufficient to guess $|\mathtt{var}(\alpha)|$ subsequence universality signatures, one for each variable, instead of the actual images of the variables under a substitution h using the following proposition by Schnoebelen and Veron [61].

Proposition 1 ([61]). *For $u, v \in \Sigma^*$, we can compute $\mathtt{s}(uv)$, given the subsequence universality signatures $\mathtt{s}(u) = (\gamma_u, \mathcal{K}_u, \mathcal{R}_u)$ and $\mathtt{s}(v) = (\gamma_v, \mathcal{K}_v, \mathcal{R}_v)$ of each string, in time polynomial in $|\mathtt{alph}(uv)|$ and $\log t$, where t is the maximum element of $\mathcal{K}_u$ and $\mathcal{K}_v$.*

Once we have guessed the subsequence universality signatures of all variables in $\mathtt{var}(\alpha)$ under substitution h, we can compute $\iota(h(\alpha))$ in the following way. We first compute the subsequence universality signature of the maximal prefix

of α that does not contain any variables. We then incrementally compute the subsequence universality signature of prefixes of the image of α. Let $\alpha = \alpha_1 \alpha_2$, where we already have $\mathbf{s}(h(\alpha_1))$ from induction. If $\alpha_2[1]$ is a variable, we compute $\mathbf{s}(h(\alpha_1 \alpha_2[1]))$ from $\mathbf{s}(h(\alpha_1))$ and the guessed subsequence universality signature for variable $\alpha_2[1]$, using Proposition 1. Otherwise, we take the maximal prefix w of α_2 that does not consist of any variables. We first compute $\mathbf{s}(w)$ and then compute $\mathbf{s}(h(\alpha_1 w))$ using Proposition 1. Once we have $\mathbf{s}(h(\alpha)) = (\gamma, \mathcal{K}, \mathcal{R})$, we compute $\iota(h(\alpha)) = \mathcal{K}[\sigma] + 1$. Note that the whole process can be done in a polynomial number of steps in $|\alpha|$, $\log k$, and σ due to Proposition 1, provided that the signatures are of polynomial size.

Thus, we now measure the encoding size of a subsequence universality signature and, as such, the overall size of the certificate for `MatchUniv` that we guess. We can use $\sigma!$ bits to encode a permutation γ of a subset of Σ. An integer between 1 and $\sigma - 1$ requires $\log \sigma$ bits. Naively, $\mathcal{R}$ requires $(2^\sigma)^\sigma$ bits because there can be 2^σ choices for each item. Finally, in the framework of our problem, note that $\mathcal{K}[1] - \mathcal{K}[|\gamma|] \leq 1$ by Schnoebelen and Veron [61], and that the values of $\mathcal{K}[i]$ are non-increasing in i. Therefore, we can encode $\mathcal{K}$ as a tuple (l, k') where $k' = \max\{\mathcal{K}[i] \mid 1 \leq i \leq |\gamma|\} \leq k$ and $l = |\{i \in [|\gamma|] \mid \mathcal{K}[i] = k'\}|$. This encoding scheme requires at most $\log \sigma + \log k$ bits. Summing up, the overall space required to encode a certificate that consists of $|\mathtt{var}(\alpha)|$ subsequence universality signatures takes at most $(1 + \sigma! + (2^\sigma)^\sigma + \log \sigma + \log k)|\mathtt{var}(\alpha)|$ bits. This is polynomial in the size of the input and the number of variables, because we assume a constant-sized alphabet, i.e. $\sigma \in O(1)$.
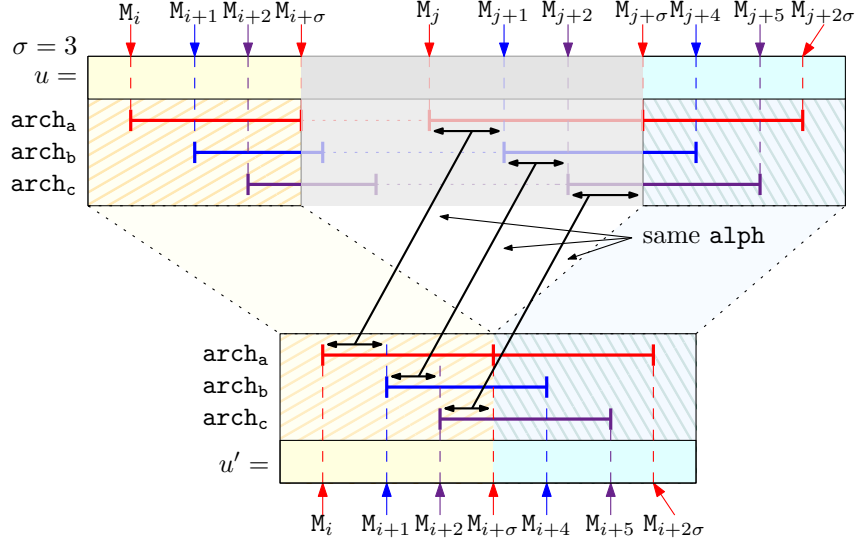
It remains to design a deterministic polynomial algorithm that tests the validity of the guessed subsequence universality signature. Assume that we have guessed the 3-tuple $(\gamma, \mathcal{K}, \mathcal{R})$. We claim that there are only constantly many strings we need to check to decide whether or not $(\gamma, \mathcal{K}, \mathcal{R})$ is a valid subsequence universality signature - allowing us a brute-force approach. Lemma 2 allows us to "pump down" strings with universality index greater than $(2^\sigma)^\sigma$, which is a constant.

Lemma 2. *The tuple $(\gamma, \mathcal{K}_1, \mathcal{R})$ is a valid subsequence universality signature iff there exists $w \in \Sigma^*$ with $\iota(w) \leq (2^\sigma)^\sigma$, $\mathbf{s}(w) = (\gamma, \mathcal{K}_2, \mathcal{R})$, and $\mathcal{K}_1[t] - \mathcal{K}_2[t] = c \in \mathbb{N}_0$ for all $t \in [|\gamma|]$.*

Proof. **Only if part.** Let $\mathbf{s}(w) = (\gamma, \mathcal{K}_2, \mathcal{R})$ with $\iota(w) \geq 1$. Then, we have $\mathbf{s}(\gamma^c w) = (\gamma, \mathcal{K}_1, \mathcal{R})$ where $\mathcal{K}_1[t] = \mathcal{K}_2[t] + c$ for all $t \in [\sigma]$ and all $c \in \mathbb{N}_0$.

If part. Let u be a string with $\mathbf{s}(u) = (\gamma, \mathcal{K}_1, \mathcal{R})$. If $\iota(u) \leq (2^\sigma)^\sigma$, the statement is already true. Otherwise, we have $\iota(u) > (2^\sigma)^\sigma$. Consider two integers i and j ($i < j$), which are multiples of σ. Assume that $\mathtt{alph}(u[\mathtt{M}_{i+l}(u) + 1 : \mathtt{M}_{i+l+1}(u)]) = \mathtt{alph}(u[\mathtt{M}_{j+l}(u) + 1 : \mathtt{M}_{j+l+1}(u)])$ for $l \in [\sigma - 1] \cup \{0\}$. Note that the endpoints of the arches after $\mathtt{M}_{j+\sigma}(u)$ depend exactly on the set of characters $\mathtt{alph}(w[\mathtt{M}_{j+l}(u)+1 : \mathtt{M}_{j+l+1}(u)])$ from $l = 0$ to $\sigma-1$, and the suffix $u[\mathtt{M}_{j+\sigma}(u)]$. Therefore, we can remove $u[\mathtt{M}_{i+\sigma}(u) + 1 : \mathtt{M}_{j+\sigma}(u)]$ without altering γ or $\mathcal{R}$. This argument is illustrated in Figure 10.

Now, let $\mathcal{K}_2$ be the array of integers obtained by subtracting $\frac{j-i}{\sigma}$ from all values in $\mathcal{K}_1$. Then, $\mathbf{s}(u[1 : \mathtt{M}_{i+\sigma}(u)]u[\mathtt{M}_{j+\sigma}(u) + 1 : |u|]) = (\gamma, \mathcal{K}_2, \mathcal{R})$. Since

Fig. 10. We can safely remove the substring $u[\mathtt{M}_{i+\sigma}(u) + 1 : \mathtt{M}_{j+\sigma}(u)]$ to obtain u' with the same γ and $\mathcal{R}$, and all $\mathcal{K}$ values are lower by $\frac{j-i}{\sigma}$.

there can be naïvely 2^σ choices of $\mathtt{alph}(u[\mathtt{M}_{i+l}(u) + 1 : \mathtt{M}_{i+l+1}(u)])$ for each $l \in [\sigma - 1] \cup \{0\}$, any string with $\iota(u) > (2^\sigma)^\sigma$ is guaranteed to have integers i and j that satisfy the above conditions by the pigeonhole principle. Therefore, we will reach a string w with $\iota(w) \leq (2^\sigma)^\sigma$ and $\mathtt{s}(w) = (\gamma, \mathcal{K}_2, \mathcal{R})$, where $\mathcal{K}_1[t] - \mathcal{K}_2[t] = c$ for some non-negative constant c and all $t \in [\sigma]$ if we repeatedly apply the same argument to remove arches.   □

Lemma 2 limits the search space for the candidate string corresponding to a tuple $(\gamma, \mathcal{K}, \mathcal{R})$ by mapping valid subsequence universality signatures to subsequence universality signatures for strings with universality index at most $(2^\sigma)^\sigma$. Therefore, we need to investigate those strings where there are up to $\sigma \cdot (1 + (2^\sigma)^\sigma) + 1$ terms in its marginal sequence. The following lemma bounds the length of the substring between two consecutive marginal sequence terms in such a string. The conclusion of this line of thought follows then, in Corollary 1.

Lemma 3. *For a given string w, let $w = uvx$ where $v = w[\mathtt{M}_i(w)+1 : \mathtt{M}_{i+1}(w)] \neq \varepsilon$, and $u = w[1 : \mathtt{M}_i(w)]$, and $x = w[\mathtt{M}_{i+1}(w) + 1 : |w|]$ for some integer $i \geq 1$. For a permutation v' of $\mathtt{alph}(v)$ that ends with $v[|v|]$, we have $\mathtt{s}(uvx) = \mathtt{s}(uv'x)$.*

Proof. Since v is between two consecutive marginal sequence terms, all arches for any signature letter in uvx must end at a position no more than $|u|$ or at a position no less than $|uv|$ Arches that end at a position no more than $|u|$ in uvx will end at the same positions in $uv'x$ because they only depend on u. Suppose that an arch for signature letter $\mathtt{a}$ starts at a position no more than $|u|$ and ends at a position no less than $|uv|$ in uvx. Let i be the minimum non-negative integer

that allows $\mathtt{alph}(\mathtt{rest_a}(u)vx[1:i]) = \Sigma$. If $i \geq 1$, we have $\mathtt{alph}(\mathtt{rest_a}(u)v) \neq \Sigma$. Since $\mathtt{alph}(\mathtt{rest_a}(u)v) = \mathtt{alph}(\mathtt{rest_a}(u)v')$, the minimum integer i' that allows $\mathtt{alph}(\mathtt{rest_a}(u)v'x[1:i']) = \Sigma$ is equal to i. On the other hand, if $i = 0$, then we have $\mathtt{alph}(\mathtt{rest_a}(u)v) = \Sigma$ and $\mathtt{alph}(\mathtt{rest_a}(u)v[1:|v|-1]) \neq \Sigma$. Since we have $v'[|v'|] = v[|v|]$ and $\mathtt{alph}(v'[1:|v'|-1]) = \mathtt{alph}(v[1:|v|-1])$, the arch ends exactly at $|uv'|$ in $uv'x$. The number of arches that continues afterwards and the corresponding letters in the rest are thus equal for uvx and $uv'x$. Finally, even if v is in the rest of the arch factorization for a signature letter $\mathtt{a}$, we still have $\mathtt{alph}(\mathtt{rest_a}(uvx)) = \mathtt{alph}(\mathtt{rest_a}(uv'x))$ because $\mathtt{alph}(v) = \mathtt{alph}(v')$. ☐

Corollary 1. *The tuple $(\gamma, \mathcal{K}, \mathcal{R})$ is a valid subsequence universality signature if and only if there exists a string w of length at most $\sigma \cdot (\sigma \cdot (1 + (2^\sigma)^\sigma) + 1)$ and a constant $c \in \mathbb{N}_0$ that satisfies $\mathtt{s}(w) = (\gamma, \mathcal{K} - c, \mathcal{R})$.*

We can now show the following result.

Lemma 4. $\mathtt{MatchUniv}(\alpha, k)$ *is in* NP.

Proof. Follows from Proposition 1 and Corollary 1. Firstly, for a guessed sequence of universality signatures $(\gamma_x, \mathcal{K}_x, \mathcal{R}_x)$, for $x \in \mathtt{var}(\alpha)$, we check their validity. For that, we enumerate all strings of length up to the constant $\sigma \cdot (\sigma \cdot (1 + (2^\sigma)^\sigma) + 1)$ over Σ and see if there exist strings w_x such that $\mathtt{s}(w_x) = (\gamma_x, \mathcal{K}_x - c_x, \mathcal{R}_x)$ for some constant $c_x \leq k$. Since σ is constant, this takes polynomial time. We then use Proposition 1 to check if the guessed signatures lead to an assignment h of the variables such that $\iota(h(\alpha)) = k$, as already explained. Since we have a polynomial size bound on the certificate and a deterministic verifier that runs in polynomial time, we obtain that $\mathtt{MatchUniv}(\alpha, k)$ is in NP. ☐

Based on Lemmas 1 and 4, the following theorem follows.

Theorem 1. $\mathtt{MatchUniv}$ *is* NP-*complete.*

Further, we describe two classes of patterns, defined by structural restrictions on the input patterns, for which $\mathtt{MatchUniv}$ can be solved in polynomial time.

Proposition 2. a) $\mathtt{MatchUniv}(\alpha, k)$ *is in* P *when there exists a variable that occurs only once in α. As such, $\mathtt{MatchUniv}(\alpha, k)$ is in P for the heavily studied class of regular patterns (see, e.g., [17] and the references therein), where each variable occurs only once.* b) $\mathtt{MatchUniv}(\alpha, k)$ *is in* P *when $|\mathtt{var}(\alpha)|$ is constant.*

Proof. a) Let x be the variable that occurs only once in α. Then, we can uniquely rewrite $\alpha = \alpha_1 x \alpha_2$. We will successively define three substitutions h_1, h_2, h_3, all of which map variables that are not x to the empty string, i.e., $h_1(x') = h_2(x') = h_3(x') = \varepsilon$ for all $x' \in \mathcal{X} \setminus \{x\}$. Now, let $h_1(x) = \varepsilon$ as well. We claim that $k \geq \iota(h_1(\alpha))$ if and only if $\mathtt{MatchUniv}(\alpha, k)$ is true. For any substitution h, we have $\iota(h_1(\alpha)) \leq \iota(h(\alpha))$ because $h_1(\alpha) \preceq h(\alpha)$. Therefore, the problem is false if $k < \iota(h_1(\alpha))$. Moreover, if $k = \iota(h_1(\alpha))$, the problem is true by definition. Now, assume $k > \iota(h_1(\alpha))$ and let $h_2(x)$ be a permutation of $\Sigma \setminus \mathtt{rest}(h_2(\alpha_1))$. Then, $\iota(h_2(\alpha)) = \iota(h_2(\alpha_1)) + 1 + \iota(h_2(\alpha_2))$, because $\mathtt{rest}(h_2(\alpha_1 x)) = \varepsilon$. Note that

we either have $\iota(h_2(\alpha)) = \iota(h_1(\alpha))$ or $\iota(h_1(\alpha)) + 1$. Finally, for an integer $i = k - \iota(h_2(\alpha))$, let $h_3(x) = h_2(x)\gamma^i$ where γ is a permutation of Σ. We now have $\iota(h_3(\alpha)) = \iota(h_3(\alpha_1)h_2(x)) + i + \iota(h_3(\alpha_2)) = i + \iota(h_2(\alpha))$. Thus, for any $k \geq \iota(h_1(\alpha))$, there exists a substitution h such that $k = \iota(h(\alpha))$. We therefore compute $\iota(h_1(\alpha))$, the universality index of the image, and then return true if and only if $\iota(h_1(\alpha)) \leq k$, which can be done in polynomial time.   □

b) The subsequence universality signature $\mathbf{s}(h(x_i))$ of the image of some variable $x_i \in \mathtt{var}(\alpha)$ under substitution h consists of three items, a permutation γ_i of a subset of Σ, an array $\mathcal{K}_i$ of σ integers, and an array $\mathcal{R}_i$ of σ subsets of Σ. Recall from the size estimation of such a universality signature that $\mathcal{K}_i$ can be represented with two integers l_i and k_i, where $\mathcal{K}_i[j] = k_i$ for all $j \in [l_i]$ and $\mathcal{K}_i[j] = k_i - 1$ for all $j \in [l_i + 1 : |\gamma_i|]$. Note that there are $\sum_{j=0}^{\sigma} j!$ choices for γ, at most σ choices for l_i, and $(2^\sigma)^\sigma$ choices for $\mathcal{R}$. Therefore, if we treat k_i as a variable whose value should be determined, we can enumerate for all possible assignments of γ_i, l_i, and $\mathcal{R}_i$ for all $i \in [|\mathtt{var}(\alpha)|]$ in constant time under the assumption that σ and $|\mathtt{var}(\alpha)|$ are constant.

Now, for a fixed set of γ_is, l_is, and $\mathcal{R}_i$s, we find the minimum value k_i' of k_i that validates $(\gamma_i, \mathcal{K}_i, \mathcal{R}_i)$ as a subsequence universality signature by enumerating all strings up to length $\sigma \cdot (\sigma \cdot (1 + (2^\sigma)^\sigma) + 1)$ using Corollary 1. If no such k_i' exists, we move on to the next set of γ_is, l_is, and $\mathcal{R}_i$s. Since Lemma 2 allows us to add an arbitrary number of arches while not altering γ_i and $\mathcal{R}_i$, we first assume that the number of additional arches is zero and compute how many more arches we need for $h(\alpha)$ to reach a universality index of k. We compute this number by counting the number of arches through an arch factorization on α. Specifically, for each rewriting $\alpha_1 x_i \alpha_2$, we compute the minimal $j \in [|\gamma|]$ that allows $\mathtt{alph}(\mathtt{rest}(h(\alpha_1))) \cup \mathtt{alph}(\gamma[1 : j]) = \Sigma$. If no such j exists, then we simply compute $\mathtt{alph}(\mathtt{rest}(h(\alpha_1 x_i))) = \mathtt{alph}(\mathtt{rest}(h(\alpha_1))) \cup \mathtt{alph}(\gamma)$ without incrementing the number of arches. Then, if $j \leq l_i$, we add k_i' arches to the total arch count. If $j > l_i$, we add $k_i' - 1$ arches instead. Finally, we continue the arch factorization process with $\mathtt{alph}(\mathtt{rest}(h(\alpha_1 x_i))) = \mathcal{R}_i[j]$. This way, we can compute the minimum number of arches the image of α can have for a fixed set of γ_is, l_is, and $\mathcal{R}_i$s.

Let d be the total number of additional arches we need for the image of α to reach a universality index of k. Note that an additional arch for each variable x_i will contribute to $|\alpha|_{x_i}$ more arches in the image of α. However, if the subsequence universality signature features γ_i with $\mathtt{alph}(\gamma_i) \neq \Sigma$, then we cannot add any more arches for each variable. Let $I = \{i \in [|\mathtt{var}(\alpha)|] \mid |\gamma_i| = \sigma\}$. Now, the problem boils down to finding how many additional arches we need for the image of each variable x_i with $i \in I$ while making the universality index of the image of α exactly k. Let d_i be the number of additional arches for each occurrence of variable x_i with $i \in I$. We can solve for d_is the following system

of linear inequalities:

$$\sum_{i \in I} |\alpha|_{x_i} d_i \leq d,$$
$$\sum_{i \in I} -|\alpha|_{x_i} d_i \leq -d,$$
$$- d_i \leq 0 \ \forall i \in I$$

Note that the first two inequalities imply $\sum_{i=1}^{|\text{var}(\alpha)|} |\alpha|_{x_i} d_i = d$ and the last $|\text{var}(\alpha)|$ inequalities enforce positive values for each d_i. If there is an integer solution for the system, then we can assign d_i more arches for the image of x_i, and the universality index of the image of α will be exactly k. Because $|\text{var}(\alpha)|$ is a constant, this system can be solved in time polynomial in $\log H$, where H is the maximum between k and the greatest coefficient of a variable in the above system (in absolute value) [39]. □

4   MatchSimon

Further, we discuss the `MatchSimon` problem. In the case of `MatchSimon`, we are given a pattern α, a word w, and a natural number $k \leq n$, and we want to check the existence of a substitution h with $h(\alpha) \sim_k w$. The first result is immediate: `MatchSimon` is NP-hard, because $\texttt{MatchSimon}(\alpha, w, |w|)$ is equivalent to $\texttt{Match}(\alpha, w)$, and `Match` is NP-complete.

Lemma 5. `MatchSimon` *is* NP*-hard.*

Proof. We note that $\texttt{MatchSimon}(\alpha, w, |w|)$ is equivalent to the NP-complete $\texttt{Match}(\alpha, w)$. □

To understand why this results followed much easier than the corresponding lower bound for `MatchUniv`, we note that in `MatchSimon` we only ask for $h(\alpha) \sim_k w$ and allow for $h(\alpha) \sim_{k+1} w$, while in `MatchUniv` $h(\alpha)$ has to be k-universal but not $(k + 1)$-universal. So, in a sense, `MatchSimon` is not strict, while `MatchUniv` is strict. So, we can naturally consider the following problem.

Matching under Strict Simon's Congruence: $\texttt{MatchStrictSimon}(\alpha, w, k)$
Input: Pattern α, $|\alpha| = m$, word w, $|w| = n$, and $k \in [n]$.
Question: Is there a substitution h with $h(\alpha) \sim_k w$ and $h(\alpha) \not\sim_{k+1} w$?

Adapting the reduction from Lemma 1, we can show that `MatchStrictSimon` is NP-hard.

Lemma 6. `MatchStrictSimon` *is* NP*-hard.*

Proof. We refer to the notations from Lemma 1. We use the same reduction from `3CNFSAT` and note that α can either be mapped to a string $h(\alpha)$ with $\iota(h(\alpha)) \leq 5n + m + 2$ (with equality only if the input instance of `3CNFSAT` is

satisfiability) or to a string $h(\alpha)$ with $\iota(h(\alpha)) \leq (n + m)^6$. Therefore, consider the instance of `MatchStrictSimon` with input the pattern α constructed in the reduction, $w = (\mathtt{10\$\#})^{5n+m+3}$, and $k = 5n + m + 2$. Clearly, there exists a substitution h with $h(\alpha) \sim_k w$ and $h(\alpha) \not\sim_{k+1} w$ if and only if there exists a substitution h with $\iota(h(\alpha)) = k$. Such a substitution exists if and only if the given instance of `3CNFSAT` is satisfiable.   □

We can also show an NP-upper bound: it is enough to consider as candidates for the images of the variables under the substitution h only strings of length $O((k + 1)^\sigma)$; longer strings can be replaced with shorter, $\sim_k$-congruent ones, which have the same impact on the sets $\mathbb{S}_k(h(\alpha))$. The following holds.

Theorem 2. `MatchSimon` *and* `MatchStrictSimon` *are* NP-*complete.*

Proof. By Lemmas 5 and 6, it is enough to show that both problems are in NP.

We make some observations first. Note that $\mathbb{S}_k(w_1 w_2) = \Sigma^{\leq k} \cap \mathbb{S}_k(w_1)\mathbb{S}_k(w_2)$. Thus, for a pattern α and two substitutions h_1 and h_2 where $h_1(x) \sim_k h_2(x)$ for all variables $x \in \mathcal{X}$, we have $w \sim_k h_1(\alpha)$ if and only if $w \sim_k h_2(\alpha)$. Moreover, Kim et al. [44] showed that, for a given string w, the length of the shortest string in the set $\{u \in \Sigma^* \mid u \sim_k w\}$ is at most $\binom{k+\sigma}{\sigma} \leq k^\sigma$.

Based on these observations, we can now give NP-algorithms for both problems. We note that these problems reduce to the normal pattern matching problem when $k \geq |w|$. For `MatchSimon`, if $k \geq |w|$, we answer `MatchSimon`(α, w, k) positively if and only if α matches w. For `MatchStrictSimon`, if $k \geq |w|$, we always answer `MatchStrictSimon`(α, w, k) negatively. Indeed, if there exists h such that $h(\alpha) \sim_k w$, then $h(\alpha) \sim_{|w|} w$. It follows that $h(\alpha) = w$ and $h(\alpha) \sim_{k+1} w$, as well; the answer to `MatchStrictSimon`(α, w, k) should therefore be no.

Hence, from now on, we can assume that $k < |w|$. Let us consider first the problem `MatchStrictSimon`. From the observations we have made at the beginning of this proof, and taking into account that we need to consider strings congruent under $\sim_{k+1}$, we can conclude that there exists a substitution h such that $h(\alpha) \sim_k w$ and $h(\alpha) \not\sim_{k+1} w$ if and only if there exists such a substitution h where the length of the image of each variable is $(k + 1)^\sigma$.

Since k is at most $|w|$ and σ is a constant, we only need to test certificates of polynomial length which encode the substitution. The verifier can then simply substitute the variables in the pattern, which will yield a string of length at most $(k + 1)^\sigma |\alpha|$. Now, we can compute the largest ℓ for which $h(\alpha) \sim_\ell w$ in $O(|h(\alpha)| + |w|) = O((|w| + 1)^\sigma |\alpha| + |w|)$ time [32], which is polynomial in the size of the input, under the assumption that σ is constant. If $\ell = k$, then we answer the respective instance positively. Therefore, the problem is in NP.

A similar argument holds for `MatchSimon` (but, in that case, it is enough to look for substitutions where the image of the variables is at most k^σ, as we only deal with $\sim_k$). On the other hand, note that the same argument cannot be applied to `MatchUniv`, because there is no bound on the size of k. This makes the size of the certificate, k^σ, exponentially large in $\log k$, which is the size of the encoding for a binary representation of k.   □

Finally, note that `MatchSimon` and `MatchStrictSimon` are in P when the input pattern is regular.

Proposition 3. *If α is a regular pattern, then both problems* `MatchSimon`(α, w, k) *and* `MatchStrictSimon`(α, w, k) *are in* P.

Proof. We consider the problem `MatchSimon`(α, w, k). We assume that α is a regular pattern $\alpha = w_0 x_1 w_1 \cdots x_\ell w_\ell$, where, for $i \in [\ell]$, x_i is a variable and, for $i \in [\ell] \cup \{0\}$, w_i is a string of constants. The language $L(\alpha)$ of all words which can be obtained by replacing the variables of α by constant strings is regular, and we can construct in polynomial time a non-deterministic finite automaton N_α accepting it (it is the automaton accepting the language described by the regular expression $w_0 \Sigma^* w_1 \cdots \Sigma^* w_\ell$). Now, using the results of [44], we can construct in polynomial time (when the size of the input alphabet σ is constant) a deterministic finite automaton $D_{w,k}$ accepting the words which are $\sim_k$ equivalent to w. Now, we simply check if there is a word accepted by both these automata (N_α and $D_{w,k}$), which can be done in polynomial time. We return the answer to this check as the answer to `MatchSimon`(α, w, k).

Further, we consider the problem `MatchStrictSimon`(α, w, k). Just as before, we construct the NFA N_α and the DFA $D_{w,k}$. Moreover, we construct the DFA $D_{w,k+1}$ and its complement $D'_{w,k+1}$ (which accepts the words which are not $\sim_{k+1}$ equivalent to w). Now, we see if there is a word accepted by N_α and $D_{w,k}$ and $D'_{w,k+1}$. Clearly, all steps can be done in polynomial time. We return the answer to this check as the answer to the problem `MatchStrictSimon`(α, w, k). $\qquad\square$

5 WESimon

In this section, we address the `WESimon` problem, where we are given two patterns α and β, and a natural number $k \leq n$, and we want to check the existence of a substitution h with $h(\alpha) \sim_k h(\beta)$. The first result is immediate: this problem is NP-hard because `MatchSimon`, which is a particular case of `WESimon`, is NP-hard.

To show that the problem is in NP, we need a more detailed analysis. If $k \leq |\alpha| + |\beta|$, the same proof as for the NP-membership of `MatchSimon` works: it is enough to look for substitutions of the variables with the image of each variable having length at most k^σ, and this is polynomial in the size of the input. If $k > |\alpha| + |\beta|$, and $\beta = w$ contains no variable, then this is an input for `MatchSimon` with k greater than the length of the input word w, and we have seen previously how this can be decided. Finally, if both α and β contain variables, then the problem is trivial, irrespective of k: the answer to any input is positive, as we simply have to map all variables to $(1 \cdots \sigma)^k$ and obtain two $\sim_k$-congruent words. Therefore, we have the following result.

Theorem 3. `WESimon` *is* NP-*complete*.

To avoid the trivial cases arising in the above analysis for `WESimon`, we can also consider a stricter variant of this problem:

Word Equations under Strict Simon's Congruence: `WEStrictSimon`(α, β, k)
Input: Patterns α, β, $|\alpha| = m$, $\beta = n$, and $k \in [m + n]$.
Question: Is there a substitution h with $h(\alpha) \sim_k h(\beta)$ and $h(\alpha) \not\sim_{k+1} h(\beta)$?

Differently from `WESimon`, we can show that this problem is NP-hard, even in the case when both sides of the pattern contain variables.

Lemma 7. `WEStrictSimon` *is* NP-*hard, even if both patterns contain variables.*

Proof. We refer to the notations from Lemma 1. We use the same reduction from `3CNFSAT` and note that α can either be mapped to a string $h(\alpha)$ with $\iota(h(\alpha)) \leq 5n + m + 2$ (with equality only if the input instance of `3CNFSAT` is satisfiability) or to a string $h(\alpha)$ with $\iota(h(\alpha)) \leq (n + m)^6$. Therefore, consider the instance of `WEStrictSimon` with input the pattern α constructed in the reduction, the second pattern $\beta = (\texttt{10}\,\$\,\texttt{\#})^{5n+m+3}x$, where x is a fresh string-variable, and $k = 5n + m + 2$. Clearly, there exists a substitution h with $h(\alpha) \sim_k h(\beta)$ and $h(\alpha) \not\sim_{k+1} h(\beta)$ if and only if there exists a substitution h with $\iota(h(\alpha)) = k$. Such a substitution exists if and only if the given instance of `3CNFSAT` is satisfiable.   □

Regarding the membership in NP: if k is upper bounded by a polynomial function in $|\alpha| + |\beta|$ (or, alternatively, if k is given in unary representation), then the fact that `WEStrictSimon` is in NP follows as in the case of `MatchStrictSimon`. The case when k is not upper bounded by a polynomial in $|\alpha| + |\beta|$ remains open. We can show the following theorem.

Theorem 4. `WEStrictSimon` *is* NP-*complete, for $k \leq |\alpha| + |\beta|$.*

6   Conclusions

In this paper, we have considered the problem of matching patterns with variables under Simon's congruence. More precisely, we have considered three main problems `MatchUniv`, `MatchSimon`, and `WESimon` and we have given a rather comprehensive image of their computational complexity. These problems are NP-complete, in general, but have interesting particular cases which are in P. Interestingly, our NP or P algorithms work in (non-deterministic) polynomial time only in the case of constant input alphabet (their complexity being, in fact, exponential in the size σ of the input alphabet). It seems very interesting to characterize the parameterised complexity of these problems w.r.t. the parameter σ. In the light of Proposition 2, another interesting parameter to be considered in such a parameterised complexity analysis would be the number of variables. We conjecture that the problems are $W[1]$-hard with respect to both these parameters.

References

1. Adamson, D.: Ranking and unranking k-subsequence universal words. In: Frid, A., Mercaş, R. (eds.) WORDS. pp. 47–59. Springer Nature Switzerland (2023)
2. Adamson, D., Kosche, M., Koß, T., Manea, F., Siemer, S.: Longest common subsequence with gap constraints. In: Frid, A., Mercaş, R. (eds.) WORDS. pp. 60–76 (2023)
3. Amadini, R.: A survey on string constraint solving. ACM Computing Surveys (CSUR) **55**(1), 1–38 (2021)
4. Amir, A., Nor, I.: Generalized function matching. J. Discrete Algorithms **5**, 514–523 (2007)
5. Angluin, D.: Finding patterns common to a set of strings. J. Comput. Syst. Sci. **21**(1), 46–62 (1980)
6. Barker, L., Fleischmann, P., Harwardt, K., Manea, F., Nowotka, D.: Scattered factor-universality of words. In: Jonoska, N., Savchuk, D. (eds.) DLT 2020, Proceedings. Lecture Notes in Computer Science, vol. 12086, pp. 14–28. Springer (2020)
7. de Bruijn, N.G.: A combinatorial problem. Koninklijke Nederlandse Akademie v. Wetenschappen **49**, 758–764 (1946)
8. Câmpeanu, C., Salomaa, K., Yu, S.: A formal study of practical regular expressions. Int. J. Found. Comput. Sci. **14**, 1007–1018 (2003)
9. Chen, H.Z.Q., Kitaev, S., Mütze, T., Sun, B.Y.: On universal partial words. Electronic Notes in Discrete Mathematics **61**, 231–237 (2017)
10. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on strings. Cambridge University Press (2007)
11. Crochemore, M., Melichar, B., Tronícek, Z.: Directed acyclic subsequence graph - overview. J. Discrete Algorithms **1**(3-4), 255–280 (2003)
12. Day, J., Fleischmann, P., Kosche, M., Koß, T., Manea, F., Siemer, S.: The edit distance to k-subsequence universality. In: STACS. vol. 187, pp. 25:1–25:19 (2021)
13. Day, J.D., Fleischmann, P., Manea, F., Nowotka, D.: Local patterns. In: Proc. 37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017. LIPIcs, vol. 93, pp. 24:1–24:14 (2017)
14. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science, Springer (1999)
15. Fagin, R., Kimelfeld, B., Reiss, F., Vansummeren, S.: Document spanners: A formal approach to information extraction. J. ACM **62**(2), 12:1–12:51 (2015)
16. Fernau, H., Manea, F., Mercas, R., Schmid, M.L.: Revisiting Shinohara's algorithm for computing descriptive patterns. Theor. Comput. Sci. **733**, 44–54 (2018)
17. Fernau, H., Manea, F., Mercas, R., Schmid, M.L.: Pattern matching with variables: Efficient algorithms and complexity results. ACM Trans. Comput. Theory **12**(1), 6:1–6:37 (2020)
18. Fernau, H., Schmid, M.L.: Pattern matching with variables: A multivariate complexity analysis. Inf. Comput. **242**, 287–305 (2015)
19. Fernau, H., Schmid, M.L., Villanger, Y.: On the parameterised complexity of string morphism problems. Theory Comput. Syst. **59**(1), 24–51 (2016)
20. Fleischer, L., Kufleitner, M.: Testing Simon's congruence. In: Potapov, I., Spirakis, P.G., Worrell, J. (eds.) MFCS 2018. LIPIcs, vol. 117, pp. 62:1–62:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018)
21. Fleischmann, P., Germann, S., Nowotka, D.: Scattered factor universality–the power of the remainder. preprint arXiv:2104.09063 (published at RuFiDim) (2021)

22. Fleischmann, P., Höfer, J., Huch, A., Nowotka, D.: α-β-factorization and the binary case of Simon's congruence (2023)
23. Freydenberger, D.D.: Extended regular expressions: Succinctness and decidability. Theory of Comput. Syst. **53**, 159–193 (2013)
24. Freydenberger, D.D.: A logic for document spanners. Theory Comput. Syst. **63**(7), 1679–1754 (2019)
25. Freydenberger, D.D., Gawrychowski, P., Karhumäki, J., Manea, F., Rytter, W.: Testing k-binomial equivalence. CoRR **abs/1509.00622** (2015)
26. Freydenberger, D.D., Holldack, M.: Document spanners: From expressive power to decision problems. Theory Comput. Syst. **62**(4), 854–898 (2018)
27. Freydenberger, D.D., Peterfreund, L.: The theory of concatenation over finite models. In: Bansal, N., Merelli, E., Worrell, J. (eds.) ICALP 2021, Proceedings. LIPIcs, vol. 198, pp. 130:1–130:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
28. Freydenberger, D.D., Schmid, M.L.: Deterministic regular expressions with backreferences. J. Comput. Syst. Sci. **105**, 1–39 (2019)
29. Friedl, J.E.F.: Mastering Regular Expressions. O'Reilly, Sebastopol, CA, third edn. (2006)
30. Garel, E.: Minimal separators of two words. In: Proc. CPM 1993. Lecture Notes in Computer Science, vol. 684, pp. 35–53. Springer (1993)
31. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1979)
32. Gawrychowski, P., Kosche, M., Koß, T., Manea, F., Siemer, S.: Efficiently testing Simon's congruence. In: STACS 2021. LIPIcs, vol. 187, pp. 34:1–34:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
33. Gawrychowski, P., Lange, M., Rampersad, N., Shallit, J.O., Szykula, M.: Existential length universality. In: Proc. STACS 2020. LIPIcs, vol. 154, pp. 16:1–16:14 (2020)
34. Gawrychowski, P., Manea, F., Siemer, S.: Matching patterns with variables under Hamming distance. In: 46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021. LIPIcs, vol. 202, pp. 48:1–48:24 (2021)
35. Gawrychowski, P., Manea, F., Siemer, S.: Matching patterns with variables under edit distance. In: Arroyuelo, D., Poblete, B. (eds.) String Processing and Information Retrieval - 29th International Symposium, SPIRE 2022, Concepción, Chile, November 8-10, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13617, pp. 275–289. Springer (2022)
36. Goeckner, B., Groothuis, C., Hettle, C., Kell, B., Kirkpatrick, P., Kirsch, R., Solava, R.W.: Universal partial words over non-binary alphabets. Theor. Comput. Sci **713** (2018)
37. Hague, M.: Strings at MOSCA. ACM SIGLOG News **6**(4), 4–22 (2019)
38. Hébrard, J.: An algorithm for distinguishing efficiently bit-strings by their subsequences. Theoretical Computer Science **82**(1), 35–49 (1991)
39. Jr., H.W.L.: Integer programming with a fixed number of variables. Mathematics of Operations Research **8**(4), 538–548 (1983)
40. Karandikar, P., Schnoebelen, P.: The height of piecewise-testable languages with applications in logical complexity. In: CSL (2016)
41. Karhumäki, J., Saarela, A., Zamboni, L.Q.: On a generalization of abelian equivalence and complexity of infinite words. J. Comb. Theory, Ser. A **120**(8), 2189–2206 (2013)
42. Karhumäki, J., Saarela, A., Zamboni, L.Q.: Variations of the Morse-Hedlund theorem for k-abelian equivalence. Acta Cybern. **23**(1), 175–189 (2017)

43. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Proceedings of a symposium on the Complexity of Computer Computations. pp. 85–103. The IBM Research Symposia Series, Plenum Press, New York (1972). https://doi.org/10.1007/978-1-4684-2001-2_9

44. Kim, S., Han, Y., Ko, S., Salomaa, K.: On Simon's congruence closure of a string. In: DCFS 2022, Proceedings. Lecture Notes in Computer Science, vol. 13439, pp. 127–141. Springer (2022)

45. Kim, S., Ko, S., Han, Y.: Simon's congruence pattern matching. In: Bae, S.W., Park, H. (eds.) 33rd International Symposium on Algorithms and Computation, ISAAC 2022, December 19-21, 2022, Seoul, Korea. LIPIcs, vol. 248, pp. 60:1–60:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022)

46. Kleest-Meißner, S., Sattler, R., Schmid, M.L., Schweikardt, N., Weidlich, M.: Discovering event queries from traces: Laying foundations for subsequence-queries with wildcards and gap-size constraints. In: 25th International Conference on Database Theory, ICDT 2022. LIPIcs, vol. 220, pp. 18:1–18:21 (2022)

47. Kosche, M., Koß, T., Manea, F., Siemer, S.: Absent subsequences in words. In: RP. pp. 115–131. Springer (2021)

48. Kosche, M., Koß, T., Manea, F., Siemer, S.: Combinatorial algorithms for subsequence matching: A survey. In: Bordihn, H., Horváth, G., Vaszil, G. (eds.) NCMA. vol. 367, pp. 11–27 (2022)

49. Krötzsch, M., Masopust, T., Thomazo, M.: Complexity of universality and related problems for partially ordered NFAs. Inf. Comput. **255**, 177–192 (2017)

50. Lejeune, M., Rigo, M., Rosenfeld, M.: The binomial equivalence classes of finite words. Int. J. Algebra Comput. **30**(07), 1375–1397 (2020)

51. Lothaire, M.: Combinatorics on Words. Cambridge University Press (1997)

52. Lothaire, M.: Algebraic Combinatorics on Words. Cambridge University Press (2002)

53. Martin, M.H.: A problem in arrangements. Bull. Amer. Math. Soc. **40**(12), 859–864 (12 1934)

54. Rampersad, N., Shallit, J., Xu, Z.: The computational complexity of universality problems for prefixes, suffixes, factors, and subwords of regular languages. Fundam. Inf. **116**(1-4), 223–236 (Jan 2012)

55. Reidenbach, D., Schmid, M.L.: Patterns with bounded treewidth. Inf. Comput. **239**, 87–99 (2014), https://doi.org/10.1016/j.ic.2014.08.010

56. Rigo, M., Salimov, P.: Another generalization of abelian equivalence: Binomial complexity of infinite words. Theor. Comput. Sci. **601**, 47–57 (2015)

57. Schmid, M.L.: A note on the complexity of matching patterns with variables. Inf. Process. Lett. **113**(19), 729–733 (2013)

58. Schmid, M.L., Schweikardt, N.: A purely regular approach to non-regular core spanners. In: Proc. 24th International Conference on Database Theory, ICDT 2021. LIPIcs, vol. 186, pp. 4:1–4:19 (2021)

59. Schmid, M.L., Schweikardt, N.: Document spanners - A brief overview of concepts, results, and recent developments. In: PODS '22: International Conference on Management of Data. pp. 139–150. ACM (2022)

60. Schnoebelen, P., Karandikar, P.: The height of piecewise-testable languages and the complexity of the logic of subwords. Logical Methods in Computer Science **15** (2019)

61. Schnoebelen, P., Veron, J.: On arch factorization and subword universality for words and compressed words. In: WORDS 2023, Proceedings. Lecture Notes in Computer Science, vol. 13899, pp. 274–287 (2023)

62. Shinohara, T.: Polynomial time inference of pattern languages and its application. In: Proc. 7th IBM Symposium on Mathematical Foundations of Computer Science, MFCS. pp. 191–209 (1982)
63. Shinohara, T., Arikawa, S.: Pattern inference. In: Algorithmic Learning for Knowledge-Based Systems, GOSLER Final Report. LNAI, vol. 961, pp. 259–291 (1995)
64. Simon, I.: Hierarchies of events with dot-depth one. Ph.D. thesis, University of Waterloo (1972)
65. Simon, I.: Piecewise testable events. In: Barkhage, H. (ed.) Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 20-23, 1975. Lecture Notes in Computer Science, vol. 33, pp. 214–222. Springer (1975)
66. Simon, I.: Words distinguished by their subwords (extended abstract). In: Proc. WORDS 2003. TUCS General Publication, vol. 27, pp. 6–13 (2003)
67. Tronícek, Z.: Common subsequence automaton. In: Proc. CIAA 2002 (Revised Papers). Lecture Notes in Computer Science, vol. 2608, pp. 270–275 (2002)
68. Weis, P., Immerman, N.: Structure theorem and strict alternation hierarchy for FOˆ2 on words. Log. Methods Comput. Sci. **5**(3) (2009)