Scattered Factor-Universality of Words

Laura Barker[1], Pamela Fleischmann[1(✉)], Katharina Harwardt[1], Florin Manea[2], and Dirk Nowotka[1]

[1] Kiel University, Kiel, Germany
{stu97347,stu120568}@mail.uni-kiel.de, {fpa,dn}@informatik.uni-kiel.de
[2] University of Göttingen, Göttingen, Germany
florin.manea@informatik.uni-goettingen.de

Abstract. A word $u = u_1 \dots u_n$ is a scattered factor of a word w if u can be obtained from w by deleting some of its letters: there exist the (potentially empty) words $v_0, v_1, .., v_n$ such that $w = v_0 u_1 v_1 ... u_n v_n$. The set of all scattered factors up to length k of a word is called its full k-spectrum. Firstly, we show an algorithm deciding whether the k-spectra for given k of two words are equal or not, running in optimal time. Secondly, we consider a notion of scattered-factors universality: the word w, with $\mathrm{alph}(w) = \Sigma$, is called k-universal if its k-spectrum includes all words of length k over the alphabet Σ; we extend this notion to k-circular universality. After a series of preliminary combinatorial results, we present an algorithm computing, for a given k'-universal word w the minimal i such that w^i is k-universal for some $k > k'$. Several other connected problems are also considered.

1 Introduction

A scattered factor (also called subsequence or subword) of a given word w is a word u such that there exist (possibly empty) words $v_0, \dots, v_n, u_1, \dots, u_n$ with $u = u_1 \dots u_n$ and $w = v_0 u_1 v_1 u_2 \dots u_n v_n$. Thus, scattered factors of a word w are imperfect representations of w, obtained by removing some of its parts. As such, there is considerable interest in the relationship between a word and its scattered factors, both from a theoretical and practical point of view (cf. e.g., the chapter *Subwords* by J. Sakarovitch and I. Simon in [27, Chapter 6] for an introduction to the combinatorial properties). Indeed, in situations where one has to deal with input strings in which errors may occur, e.g., sequencing DNA or transmitting a digital signal, scattered factors form a natural model for the processed data as parts of the input may be missing. This versatility of scattered factors is also highlighted by the many contexts in which this concept appears. For instance, in [16,24,37], various logic-theories were developed around the notion of scattered factors which are analysed mostly with automata theory tools and discussed in connection to applications in formal verification. On an even

F. Manea—Supported by the DFG grant MA 5725/2-1. F.M. thanks Paweł Gawrychowski for his comments and suggestions.

N. Jonoska and D. Savchuk (Eds.): DLT 2020, LNCS 12086, pp. 14–28, 2020.
https://doi.org/10.1007/978-3-030-48516-0_2

more fundamental perspective, there have been efforts to bridge the gap between the field of combinatorics on words, with its usual non-commutative tools, and traditional linear algebra, via, e.g., subword histories or Parikh matrices (cf. e.g., [30,33,34]) which are algebraic structures in which the number of specific scattered factors occurring in a word are stored. In an algorithmic framework, scattered factors are central in many classical problems, e.g., the longest common subsequence or the shortest common supersequence problems [1,28], the string-to-string correction problem [36], as well as in bioinformatics-related works [10].

In this paper we focus, for a given word, on the sets of scattered factors of a given length: the (full) k-spectrum of w is the set containing all scattered factors of w of length exactly k (up to k resp.). The total set of scattered factors (also called downward closure) of $w = \mathtt{aba}$ is $\{\varepsilon, \mathtt{a}, \mathtt{aa}, \mathtt{ab}, \mathtt{aba}, \mathtt{b}, \mathtt{ba}\}$ and the 2-spectrum is $\{\mathtt{aa}, \mathtt{ab}, \mathtt{ba}\}$. The study of scattered factors of a fixed length of a word has its roots in [35], where the relation $\sim_k$ (called Simon's congruence) defines the congruence of words that have the same full k-spectra. Our main interest here lies in a special congruence class w.r.t. $\sim_k$: the class of words which have the largest possible k-spectrum. A word w is called k-*universal* if its k-spectrum contains all the words of length k over a given alphabet. That is, k-universal words are those words that are as rich as possible in terms of scattered factors of length k (and, consequently, also scattered factors of length at most k): the restriction of their downward closure to words of length k contains all possible words of the respective length, i.e., is a *universal* language. Thus $w = \mathtt{aba}$ is not 2-universal since $\mathtt{bb}$ is not a scattered factor of w, while $w' = \mathtt{abab}$ is 2-universal. Calling a words *universal* if its k-spectrum contains all possible words of length k, is rooted in formal language theory. The classical universality problem (cf. e.g., [18]) is whether a given language L (over an alphabet Σ) is equal to Σ^*, where L can be given, e.g., as the language accepted by an automaton. A variant of this problem, called length universality, asks, for a natural number ℓ and a language L (over Σ), whether L contains all strings of length ℓ over Σ. See [14] for a series of results on this problem and a discussion on its motivation, and [14,23,31] and the references therein for more results on the universality problem for various types of automata. The universality problem was also considered for words [6,29] and, more recently, for partial words [2,15] w.r.t. their factors. In this context, the question is to find, for a given ℓ, a word w over an alphabet Σ, such that each word of length ℓ over Σ occurs exactly once as a contiguous factor of w. De Bruijn sequences [6] fulfil this property, and have been shown to have many applications in various areas of computer science or combinatorics, see [2,15] and the references therein. As such, our study of scattered factor-universality is related to, and motivated by, this well developed and classical line of research.

While $\sim_k$ is a well studied congruence relation from language theoretic, combinatorial, or algorithmic points of view (see [11,27,35] and the references therein), the study of universality w.r.t. scattered factors seems to have been mainly carried out from a language theoretic point of view. In [20] as well as in [21,22] the authors approach, in the context of studying the height of piecewise testable languages, the notion of ℓ-rich words, which coincides with the

ℓ-universal words we define here; we will discuss the relation between these notions, as well as our preference to talk about universality rather than richness, later in the paper. A combinatorial study of scattered factors universality was started in [5], where a simple characterisation of k-universal binary words was given. In the combinatorics on words literature, more attention was given to the so called binomial complexity of words, i.e., a measure of the multiset of scattered factors that occur in a word, where each occurrence of such a factor is considered as an element of the respective multiset (see, e.g., [12,25,26,32]). As such, it seemed interesting to us to continue the work on scattered factor universality: try to understand better (in general, not only in the case of binary alphabets) their combinatorial properties, but, mainly, try to develop an algorithmic toolbox around the concept of (k-)universal words.

Our Results. In the preliminaries we give the basic definitions and recall the arch factorisation introduced by Hebrard [17]. Moreover we explain in detail the connection to richness introduced in [20].

In Sect. 3 we show one of our main results: testing whether two words have the same full k-spectrum, for given $k \in \mathbb{N}$, can be done in optimal linear time for words over ordered alphabets and improve and extend the results of [11]. They also lead to an optimal solution over general alphabets.

In Sect. 4 we prove that the arch factorisation can be computed in time linear w.r.t. the word-length and, thus, we can also determine whether a given word is k-universal. Afterwards, we provide several combinatorial results on k-universal words (over arbitrary alphabets); while some of them follow in a rather straightforward way from the seminal work of Simon [35], other require a more involved analysis. One such result is a characterisation of k-universal words by comparing the spectra of w and w^2. We also investigate the similarities and differences of the universality if a word w is repeated or w^R and $\pi(w)$ resp. are appended to w, for a morphic permutation of the alphabet π. As consequences, we get a linear run-time algorithm for computing a minimal length scattered factor of ww that is not a scattered factor of w. This approach works for arbitrary alphabets, while, e.g., the approach of [17] only works for binary ones. We conclude the section by analysing the new notion of k-circular universality, connected to the universality of repetitions.

In Sect. 5 we consider the problem of modifying the universality of a word by repeated concatenations or deletions. Motivated by the fact that, in general, starting from an input word w, we could reach larger sets of scattered factors of fixed length by iterative concatenations of w, we show that, for a word w a positive integer k, we can compute efficiently the minimal ℓ such that w^ℓ is k-universal. This result is extensible to sets of words. Finally, the shortest prefix or suffix we need to delete to lower the universality index of a word to a given number can be computed in linear time. Interestingly, in all of the algorithms where we are concerned with reaching k-universality we never effectively construct a k-universal word (which would take exponential time, when k is given as input via its binary encoding, and would have been needed when solving these problems using, e.g., [10,11]). Our algorithms run in polynomial time w.r.t. $|w|$, the length of the input word, and $\log_2 k$, the size of the representation of k.

2 Preliminaries

Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. Define $[n]$ as the set $\{1, \ldots, n\}$, $[n]_0 = [n] \cup \{0\}$ for an $n \in \mathbb{N}$, and $\mathbb{N}_{\geq n} = \mathbb{N}\backslash[n-1]$. An alphabet Σ is a nonempty finite set of symbols called *letters*. A *word* is a finite sequence of letters from Σ, thus an element of the free monoid Σ^*. Let $\Sigma^+ = \Sigma^*\backslash\{\varepsilon\}$, where ϵ is the empty word. The *length* of a word $w \in \Sigma^*$ is denoted by $|w|$. For $k \in \mathbb{N}$ define $\Sigma^k = \{w \in \Sigma^* | |w| = k\}$ and $\Sigma^{\leq k}, \Sigma^{\geq k}$ analogously. A word $u \in \Sigma^*$ is a *factor* of $w \in \Sigma^*$ if $w = xuy$ for some $x, y \in \Sigma^*$. If $x = \varepsilon$ (resp. $y = \epsilon$), u is called a *prefix* (resp. *suffix* of w). Let $\mathrm{Pref}_k(w)$ be the prefix of w of length $k \in \mathbb{N}_0$. The i^{th} letter of $w \in \Sigma^*$ is denoted by $w[i]$ for $i \in [|w|]$ and set $w[i..j] = w[i]w[i+1]\ldots w[j]$ for $1 \leq i \leq j \leq |w|$. Define the *reversal* of $w \in \Sigma^n$ by $w^R = w[n]\ldots w[1]$. Set $|w|_{\mathtt{a}} = |\{i \in [|w|] | w[i] = \mathtt{a}\}|$ and $\mathrm{alph}(w) = \{\mathtt{a} \in \Sigma | |w|_{\mathtt{a}} > 0\}$ for $w \in \Sigma^*$. For a word $u \in \Sigma^*$ we define $u^0 = \varepsilon, u^{i+1} = u^i u$, for $i \in \mathbb{N}$. A word $w \in \Sigma^*$ is called *power* (repetition) of a word $u \in \Sigma^*$, if $w = u^t$ for some $t \in \mathbb{N}_{\geq 2}$. A word $u \in \Sigma^*$ is a *conjugate* of $w \in \Sigma^*$ if there exist $x, y \in \Sigma^*$ with $w = xy$ and $u = yx$. A function $\pi : \Sigma^* \to \Sigma^*$ is called *morphic permutation* if π is bijective and $\pi(uv) = \pi(u)\pi(v)$ for all $u, v \in \Sigma^*$.

Definition 1. *A word $v = v_1 \ldots v_k \in \Sigma^*$ is a* scattered factor *of $w \in \Sigma^*$ if there exist $x_1, \ldots, x_{k+1} \in \Sigma^*$ such that $w = x_1 v_1 \ldots x_k v_k x_{k+1}$. Let* $\mathrm{ScatFact}(w)$ *be the set of all scattered factors of w and define* $\mathrm{ScatFact}_k(w)$ *(resp.,* $\mathrm{ScatFact}_{\leq k}(w)$*) as the set of all scattered factors of w of length (resp., up to) $k \in \mathbb{N}$. A word $u \in \Sigma^*$ is a* common scattered factor *of $w, v \in \Sigma^*$, if $u \in \mathrm{ScatFact}(w) \cap \mathrm{ScatFact}(v)$; the word u is an* uncommon scattered factor *of w and v (and* distinguishes *them) if u is a scattered factor of exactly one of them.*

For $k \in \mathbb{N}_0$, the sets $\mathrm{ScatFact}_k(w)$ and $\mathrm{ScatFact}_{\leq k}(w)$ are also known as the k-spectrum and the full-k-spectrum of w resp.. Simon [35] defined the congruence $\sim_k$ in which $u, v \in \Sigma^*$ are congruent if they have the same full k-spectrum and thus the same k-spectrum. The *shortlex normal form* of a word $w \in \Sigma^*$ w.r.t. $\sim_k$, where Σ is an ordered alphabet, is the shortest word u with $u \sim_k w$ which is also lexicographically smallest (w.r.t. the given order on Σ) amongst all words $v \sim_k w$ with $|v| = |u|$. The maximal cardinality of a word's k-spectrum is $|\Sigma|^k$ and as shown in [5] this is equivalent in the binary case to $w \in \{\mathtt{ab}, \mathtt{ba}\}^k$. The following definition captures this property of a word in a generalised setting.

Definition 2. *A word $w \in \Sigma^*$ is called k-*universal *(w.r.t. Σ), for $k \in \mathbb{N}_0$, if* $\mathrm{ScatFact}_k(w) = \Sigma^k$. *We abbreviate 1-universal by* universal. *The* universality-index $\iota(w)$ *of $w \in \Sigma^*$ is the largest k such that w is k-universal.*

Remark 3. Notice that k-universality is always w.r.t. a given alphabet Σ: the word $\mathtt{abcba}$ is 1-universal for $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ but it is not universal for $\Sigma \cup \{\mathtt{d}\}$. If it is clear from the context, we do not explicitly mention Σ. The universality of the factors of a word w is considered w.r.t. $\mathrm{alph}(w)$.

Karandikar and Schnoebelen introduced in [21,22] the notion of richness of words: $w \in \Sigma^*$ is *rich* (w.r.t. Σ) if $\mathrm{alph}(w) = \Sigma$ (and poor otherwise) and w is ℓ-rich if w is the concatenation of $\ell \in \mathbb{N}$ rich words. Immediately we get that a word is universal iff it is rich and moreover that a word is ℓ-rich iff it is ℓ-universal and a rich-factorisation, i.e., the factorisation of an ℓ-rich word into ℓ rich words, can be efficiently obtained. However, we will use the name *ℓ-universality* rather than *ℓ-richness*, as richness defines as well, e.g. the property of a word $w \in \Sigma^n$ to have $n+1$ distinct palindromic factors, see, e.g., [7,9]. As w is ℓ-universal iff w is the concatenation of $\ell \in \mathbb{N}$ universal words it follows immediately that, if w is over the ordered alphabet $\Sigma = \{1 < 2 < \ldots < \sigma\}$ and it is ℓ-universal then its shortlex normal form w.r.t. $\sim_\ell$ is $(1 \cdot 2 \cdots \sigma)^\ell$ (as this is the shortest and lexicographically smallest ℓ-universal word).

The following observation leads to the next definition: the word $w = \mathtt{abc} \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*$ is 1-universal and w^s is s-universal for all $s \in \mathbb{N}$. But, $v^2 = (\mathtt{ababcc})^2 \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*$ is 3-universal even though v is only 1-universal. Notice that the conjugate $\mathtt{abccab}$ of v is 2-universal.

Definition 4. *A word $w \in \Sigma^*$ is called k-*circular universal *if a conjugate of w is k-universal (abbreviate 1-circular universal by circular universal). The* circular universality index $\zeta(w)$ *of w is the largest k such that w is k-circular universal.*

Remark 5. It is worth noting that, unlike the case of factor universality of words and partial words [2,6,15,29], in the case of scattered factors it does not make sense to try to identify a k-universal word $w \in \Sigma^*$, for $k \in \mathbb{N}_0$, such that each word from Σ^k occurs *exactly once* as scattered factor of w. Indeed for $|\Sigma| = \sigma$, if $|w| \geq k + \sigma$ then there exists a word from Σ^k which occurs at least twice as a scattered factor of w. Moreover, the shortest word which is k-universal has length $k\sigma$ (we need $\mathtt{a}^k \in \mathrm{ScatFact}_k(w)$ for all $\mathtt{a} \in \Sigma$). As $k\sigma \geq k + \sigma$ for $k, \sigma \in \mathbb{N}_{\geq 2}$, all k-universal words have scattered factors occurring more than once: there exists $i, j \in [\sigma + 1]$ such that $w[i] = w[j]$ and $i \neq j$. Then $w[i]w[\sigma+2..\sigma+k], w[j]w[\sigma+2..\sigma+k] \in \mathrm{ScatFact}_k(w)$ and $w[i]w[\sigma+2.\sigma+k] = w[j]w[\sigma+2..\sigma+k]$.

We now recall the arch factorisation, introduced by Hebrard in [17].

Definition 6 ([17]**).** *For $w \in \Sigma^*$ the* arch factorisation *of w is given by $w = \mathrm{ar}_w(1) \ldots \mathrm{ar}_w(k) r(w)$ for a $k \in \mathbb{N}_0$ with $\mathrm{ar}_w(i)$ is universal and $\mathrm{ar}_w(i)[|\mathrm{ar}_w(i)|] \notin \mathit{alph}(\mathrm{ar}_w(i)[1 \ldots |\mathrm{ar}_w(i)| - 1])$ for all $i \in [n]$, and $\mathit{alph}(r(w)) \subset \Sigma$. The words $\mathrm{ar}_w(i)$ are called* archs *of w, $r(w)$ is called the* rest. *Set $m(w) = \mathrm{ar}_w(1)[|\mathrm{ar}_w(1)|] \ldots \mathrm{ar}_w(k)[|\mathrm{ar}_w(k)|]$ as the word containing the unique last letters of each arch.*

Remark 7. If the arch factorisation contains $k \in \mathbb{N}_0$ archs, the word is k-universal, thus the equivalence of k-richness and k-universality becomes clear. Moreover if a factor v of $w \in \Sigma^*$ is k-universal then w is also k-universal: if v has an arch factorisation with k archs then w's arch factorisation has at least k archs (in which the archs of v and w are not necessarily related).

Finally, our main results are of algorithmic nature. The computational model we use is the standard unit-cost RAM with logarithmic word size: for an input of size n, each memory word can hold $\log n$ bits. Arithmetic and bitwise operations with numbers in $[n]$ are, thus, assumed to take $O(1)$ time. Arithmetic operations on numbers larger than n, with ℓ bits, take $O(\ell/\log n)$ time. For simplicity, when evaluating the complexity of an algorithm we first count the number of steps we perform (e.g., each arithmetic operation is counted as 1, no matter the size of the operands), and then give the actual time needed to implement these steps in our model. In our algorithmic problems, we assume that the processed words are sequences of integers (called letters or symbols, each fitting in $O(1)$ memory words). In other words, we assume that the alphabet of our input words is *an integer alphabet*. In general, after a linear time preprocessing, we can assume that the letters of an input word of length n over an integer alphabet Σ are in $\{1, \ldots, |\Sigma|\}$ where, clearly, $|\Sigma| \leq n$. For a more detailed discussion see, e.g., [4].

3 Testing Simon's Congruence

Our first result extends and improves the results of Fleischer and Kufleitner [11].

Theorem 8. *(1) Given a word w over an integer alphabet Σ, with $|w| = n$, and a number $k \leq n$, we can compute the shortlex normal form of w w.r.t. $\sim_k$ in time $O(n)$. (2) Given two words w', w'' over an integer alphabet Σ, with $|w'| \leq |w''| = n$, and a number $k \leq n$, we can test if $w' \sim_k w''$ in time $O(n)$.*

Proof. The main idea of the algorithm is that checking $w' \sim_k w''$ is equivalent to checking whether the shortlex normal forms w.r.t. $\sim_k$ of w' and w'' are equal. To compute the shortlex normal form of a word $w \in \Sigma^n$ w.r.t. $\sim_k$ the following approach was used in [11]: firstly, for each position of w the x- and y-coordinates were defined. The x-coordinate of i, denoted x_i, is the length of the shortest sequence of indices $1 \leq i_1 < i_2 < \ldots < i_t = i$ such that i_1 is the position where the letter $w[i_1]$ occurs w for the first time and, for $1 < j \leq t$, i_j is the first position where $w[i_j]$ occurs in $w[i_{j-1} + 1..i]$. Obviously, if $\mathtt{a}$ occurs for the first time on position i in w, then $x_i = 1$ (see [11] for more details). A crucial property of the x-coordinates is that if $w[\ell] = w[i] = \mathtt{a}$ for some $i > \ell$ such that $w[j] \neq \mathtt{a}$ for all $\ell + 1 \leq j \leq i - 1$, then $x_i = \min\{x_\ell, x_{\ell+1}, \ldots, x_{i-1}\} + 1$. The y-coordinate of a position i, denoted y_i, is defined symmetrically: y_i is the length of the shortest sequence of indices $n \geq i_1 > i_2 > \ldots > i_t = i$ such that i_1 is the position where the letter $w[i_1]$ occurs last time in w and, for $1 < j \leq t$, i_j is the last position where $w[i_j]$ occurs in $w[i..i_{j-1} - 1]$. Clearly, if $w[\ell] = w[i] = \mathtt{a}$ for some $i < \ell$ such that $w[j] \neq \mathtt{a}$ for all $\ell - 1 \geq j \geq i + 1$, then $y_i = \min\{y_{i+1}, \ldots, y_{\ell-1}, y_\ell\} + 1$.

Computing the coordinates is done in two phases: the x-coordinates are computed and stored (in an array x with elements $x_1, \ldots, x_n$) from left to right in phase 1a, and the y-coordinates are stored in an array y with elements $y_1, \ldots, y_n$ and computed from right to left in phase 1b (while dynamically deleting a position whenever the sum of its coordinates is greater then $k + 1$ (cf. [11, Prop. 2])). Then, to compute the shortlex normal form, in a third phase, labelled phase 2, if letters $\mathtt{b} > \mathtt{a}$ occur consecutively in this order, they are interchanged whenever they have the same x- and y-coordinates and the sum of these coordinates is $k+1$ (until this situation does not occur anymore).

We now show how these steps can be implemented in $O(n)$ time for input words over integer alphabets. For simplicity, let $x[i..j]$ denote the sequence of coordinates $x_i, x_{i+1}, \ldots, x_j$; $\min(x[i..j])$ denotes $\min\{x_i, \ldots, x_j\}$. It is clear that in $O(n)$ time we can compute all values $\text{last}[i] = \max(\{0\} \cup \{j < i | w[j] = w[i]\})$.

Firstly, phase 1a. For simplicity, assume that $x_0 = 0$. While going with i from 1 to n, we maintain a list L of positions $0 = i_0 < i_1 < i_2 < \ldots < i_t = i$ such that the following property is invariant: $x_{i_{\ell-1}} < x_{i_\ell}$ for $1 \leq \ell \leq t$ and $x_p \geq x_{i_\ell}$ for all $i_{\ell-1} < p \leq i_\ell$. After each i is read, if $\text{last}[i] = 0$ then set $x_i = 1$; otherwise, determine $x_i = \min(x[\text{last}[i]..i-1]) + 1$ by L, then append i to L and update L accordingly so that its invariant property holds. This is done as follows: we go through the list L from right to left (i.e., inspect the elements $i_t, i_{t-1}, \ldots$) until we reach a position $i_{j-1} < \text{last}[i]$ or completely traverse the list (i.e., $i_{j-1} = 0$). Let us note now that all elements x_ℓ with $i-1 \geq \ell \geq \text{last}[i]$ fulfill $x_\ell \geq x_{i_j}$ and $i_j \geq \text{last}[i]$. Consequently, $x_i = x_{i_j} + 1$. Moreover, $x_{i_{j+1}} \geq x_{i_j} + 1$. As such, we update the list L so that it becomes $i_1, \ldots, i_j, i$ (and x_i is stored in the array x).

Note that each position of w is inserted once in L and once deleted (but never reinserted). Also, the time needed for the update of L caused by the insertion of i is proportional to the number of elements removed from the list in that step. Accordingly, the total time needed to process L, for all i, is $O(n)$. Clearly, this procedure computes the x-coordinates of all the positions of w correctly.

Secondly, phase 1b. We cannot proceed exactly like in the previous case, because we need to dynamically delete a position whenever the sum of its coordinates is greater than $k+1$ (i.e., as soon as we finished computing its y-coordinate and see that it is $> k+1$; this position does not influence the rest of the computation). If we would proceed just as above (right to left this time), it might be the case that after computing some y_i we need to delete position i, instead of storing it in our list and removing some of the elements of the list. As such, our argument showing that the time spent for inspecting and updating the list in the steps where the y-coordinates are computed amortises to $O(n)$ would not work.

So, we will use an enhanced approach. For simplicity, assume that $y_{n+1} = 0$ and that every time we should eliminate position i we actually set y_i to $+\infty$. Also, let $y[i..j]$ denote the sequence of coordinates $y_i, y_{i+1}, \ldots, y_j$; note that some of these coordinates can be $+\infty$. Let $min(y[i..j])$ denote the minimum in the sequence $y[i..j]$. Similarly to what we did in phase 1a, while going with i from n to 1, we maintain a list L' of positions $n+1 = i_0 > i_1 > i_2 > \ldots > i_t \geq i$ such that the following property is invariant: $y_{i_{\ell-1}} < y_{i_\ell}$ for $1 \leq \ell \leq t$ and $y_p \geq y_{i_\ell}$ for all $i_{\ell-1} > p \geq i_\ell$. In the current case, we also have that $y_p = +\infty$ for all $i_t > p \geq i$. The numbers $i_0, i_1, i_2, \ldots, i_t \geq i$ contained in the list L' at some moment in our computation define a partition of the universe $[1, n]$ in intervals: $\{1\}, \{2\}, \ldots, \{i-1\}, [i, i_{t-1} - 1], [i_{t-1}, i_{t-2} - 1], \ldots, [i_1, i_0 - 1]$ for which we define an *interval union-find* data structure [13,19]; here the singleton $\{a\}$ is seen as the interval $[a, a]$. According to [19], in our model of computation, such a structure can be initialized in $O(n)$ time such that we can perform a sequence of $O(n)$ `union` and `find` operations on it in $O(n)$ time, with the crucial restriction that one can only unite neighbouring intervals. We assume that `find(j)` returns the bounds of the interval stored in our data structure to which j belongs. From the definition of the list L', it is clear that, before processing position i (and after finishing processing position $i+1$), $y_{i_\ell} = \min(y[i+1..i_{\ell-1}-1])$ holds. We maintain a new array next$[\cdot]$ with $|\Sigma|$ elements: before processing position i, next$[w[i]]$ is the smallest position $j > i$ where $w[i]$ occurs after position i, which was not eliminated (i.e., smallest $j > i$ with $y_j \neq +\infty$), or 0 if there is no such position. Position i is now processed as follows: let $[a, b]$ be the interval returned by `find`(next$[i]$). If $a = i + 1$ then let min $= y_{i_t}$; if $a > i + 1$ then there exists j such that $[a, b] = [i_j, i_{j-1} - 1]$ and $t > j > 0$, so let min $= y_j$. Let now $y = \min +1$, and note that we should set $y_i = y$, but only if $x_i + i \leq k + 1$. So, we check whether $x_i + i \leq k + 1$ and, if yes, let $y_i = y$ and set next$[w[i]] = i$; otherwise, set $y_i = +\infty$ (note that position i becomes, as such, irrelevant when the y-coordinate is computed for other positions). If $y_i = +\infty$ then make the union of the intervals $\{i\}$ and $[i + 1, i_{t-1} - 1]$ and start processing $i - 1$; L' remains unchanged. If $y_i \neq +\infty$ then make the union of the intervals $\{i\}, [i + 1, i_{t-1} - 1], \ldots, [i_{j+1}, i_j - 1]$ and start processing $i - 1$; L' becomes $i, i_j, i_{j-1}, \ldots, i_0$.

As each position of w is inserted at most once in L', and then deleted once (never reinserted), the number of list operations is $O(n)$. The time needed for the update of L', caused by the insertion of i in L', is proportional to the number of elements removed from L' in that step, so the total time needed (exclusively) to process L is $O(n)$. On top of that, for each position i, we run one `find` operation and a number of `union` operations proportional to the number of elements removed from L' in that step. Overall we do $O(n)$ `union` and `find` operations on the *union-find* data structure. This takes in total, for all i, $O(n)$ time (including the initialisation). Thus, the time complexity of phase 1b is linear.

Thirdly, phase 2. Assume that w_0 is the input word *of this phase.* Clearly, $|w_0| = m \leq n$, and we have computed the coordinates for all its positions (and maybe eliminated some positions of the initial input word w). We partition in linear time $O(n)$ the interval $[1, m]$ into $2t+1$ (possibly empty) lists of positions $L_1, \ldots, L_{2t+1}$ such that the following conditions hold. Firstly, all elements of L_i are smaller than those of L_{i+1} for $1 \leq i \leq 2t$. Secondly, for i odd, the elements j in L_i have $x_j + y_j < k + 1$; for each i even, there exist a_i, b_i such that $a_i + b_i = k + 1$ and for all j in L_i we have $x_j = a_i, y_j = b_i$. Thirdly, we want t to be minimal with these properties. We now produce, also in linear time, a new list U: for each $i \leq t$ and $j \in L_{2i}$ we add the triplet $(i, w[j], j)$ in U. We sort the list of triples U (cf. [11, Prop. 10]) with radix sort in linear time [3]. After sorting it, U can be decomposed in t consecutive blocks $U_1, U_2, \ldots, U_t$, where U_i contains the positions of L_{2i} sorted w.r.t. the order on Σ (i.e., determined by the second component of the pair). As such, U_i induces a new order on the positions of w_0 stored in L_{2i}. We can now construct a word w_1 by just writing in order the letters of w_0 corresponding to the positions stored in L_i, for i from 1 to $2t + 1$, such that the letters of L_i are written in the original order, for i odd, and in the order induced by U_i, for i even. Clearly, this is a correct implementation of phase 2 which runs in linear time. The word w_1 is the shortlex normal form of w.

Summing up, we have shown how to compute the shortlex normal form of a word in linear time (for integer alphabets). Both our claims follow. $\square$

This improves the complexity of the algorithm reported in [11], where the problem was solved in $O(n|\Sigma|)$ time. As such, over integer alphabets, testing Simon's congruence for a given k can be done in optimal time, that does not depend on the input alphabet or on k. When no restriction is made on the input alphabet, we can first sort it, replace the letters by their ranks, and, as such, reduce the problem to the case of integer alphabets. In that case, testing Simon's congruence takes $O(|\Sigma| \log |\Sigma| + n)$ time which is again optimal: for $k = 1$, testing if $w_1 \sim_1 w_2$ is equivalent (after a linear time processing) to testing whether two subsets of Σ are equal, and this requires $\Theta(|\Sigma| \log |\Sigma|)$ time [8].

4 Scattered Factor Universality

In this section we present several algorithmic and combinatorial results.

Remark 9. Theorem 8 allows us to decide in linear time $O(n)$ whether a word w over $\Sigma = \{1 < 2 < \ldots < \sigma\}$ is k-universal, for a given $k \leq n, \sigma \in \mathbb{N}$. We compute the shortlex normal form of w w.r.t. $\sim_k$ and check whether it is $(1 \cdot 2 \cdots \sigma)^k$.

We can actually compute $\iota(w)$ efficiently by computing its arch factorisation in linear time in $|w|$. Moreover this allows us to check whether w is k-universal for some given k by just checking if $\iota(w) \geq k$ or not.

Proposition 10. *Given a word $w \in \Sigma^n$, we can compute $\iota(w)$ in time $O(n)$.*

Proof. We actually compute the number ℓ of archs in the arch factorisation. For a lighter notation, we use $u_i = \mathrm{ar}_w(i)$ for $i \in [\ell]_0$. The factors u_i can be computed in linear time as follows. We maintain an array C of $|\Sigma|$ elements, whose all elements are initially 0, and a counter h, which is initially $|\Sigma|$. For simplicity, let $m_0 = 0$. We go through the letters $w[j]$ of $w[m_{i-1}+1..n]$, from left to right, and if $C[w[j]]$ equals 0, we decrement h by 1 and set $C[w[j]] = 1$. Intuitively, we keep track of which letters of Σ we meet while traversing $w[m_{i-1}+1..n]$ using the array C, and we store in h how many letters we still need to see. As soon as $h = 0$ or $j = n$, we stop: set $m_i = j$ (the position of the last letter of w we read), $u_i = w[m_{i-1}+1..m_i]$ (the i^{th} arch), and $h = |\Sigma|$ again. If $j < n$ then reinitialise all elements of C to 0 and restart the procedure for $i+1$. Note that if $j = n$ then u_i is $r(w)$ as introduced in the definition of the arch factorization. The time complexity of computing u_j is $O(|u_j|)$, because we process each symbol of $u_i = w[m_{i-1}+1..m_i]$ in $O(1)$ time, and, at the end of the procedure, we reinitialise C in $O(|\Sigma|)$ time iff u_i contained all letters of Σ, so $|u_i| \geq |\Sigma|$. The conclusion follows. $\square$

The following combinatorial result characterise universality by repetitions.

Theorem 11 (∗). *A word $w \in \Sigma^{\geq k}$ with alph$(w) = \Sigma$ is k-universal for $k \in \mathbb{N}_0$ iff* $\mathrm{ScatFact}_k(w^n) = \mathrm{ScatFact}_k(w^{n+1})$ *for an $n \in \mathbb{N}$. Moreover we have $\iota(w^n) \geq kn$ if $\iota(w) = k$.*

As witnessed by $w = \mathtt{aabb} \in \{\mathtt{a}, \mathtt{b}\}^*$, $\iota(w^n)$ can be greater than $n \cdot \iota(w)$: w is universal, not 2-universal but $w^2 = \mathtt{aab.ba.ab.b}$ is 3-universal. We study this phenomenon at the end of this section. Theorem 11 can also be used to compute an uncommon scattered factor of w and ww over arbitrary alphabets; note that the shortest such a factor has to have length $k+1$ if $\iota(w) = k$.

Proposition 12 (∗). *Given a word $w \in \Sigma^*$ we can compute in linear time $O(|w|)$ one of the uncommon scattered factors of w und ww of minimal length.*

Remark 13. By Proposition 12, computing the shortest uncommon scattered factor of w and ww takes optimal $O(n)$ time, which is more efficient than running an algorithm computing the shortest uncommon scattered factor of two arbitrary words (see, e.g., [10,11], and note that we are not aware of any linear-time algorithm performing this task for integer alphabets). In particular, we can use Theorem 8 to find by binary search the smallest k for which two words have distinct k-spectra in $O(n \log n)$ time. In [17] a linear time algorithm solving this problem is given for binary alphabets; an extension seems non-trivial.

Continuing the idea of Theorem 11, we investigate even-length palindromes, i.e. appending w^R to w. The first result is similar to Theorem 11 for $n = 1$. Notice that $\iota(w) = \iota(w^R)$ follows immediately with the arch factorisation.

Corollary 14. *A word w is k-universal iff* $\mathrm{ScatFact}_k(w) = \mathrm{ScatFact}_k(ww^R)$.

In contrast to $\iota(w^2)$, $\iota(ww^R)$ is never greater than $2\iota(w)$.

Proposition 15 (∗). *Let $w \in \Sigma^*$ be a palindrome and $u = \mathrm{Pref}_{\lfloor \frac{|w|}{2} \rfloor}(w)$ with $\iota(u) = k \in \mathbb{N}$. For $|w|$ even we have $\iota(w) = 2k$ if $|w|$ even and for $|w|$ odd we get $\iota(w) = 2k + 1$ iff $w[\frac{n+1}{2}] \cup$ alph$(r(u)) = \Sigma$.*

Remark 16. If we consider the universality of a word $w = w_1 \dots w_m$ for $m \in \mathbb{N}$ with $w_i \in \{u, u^R\}$ for a given word $u \in \Sigma^*$, then a combination of the previous results can be applied. Each time either u^2 or $(u^R)^2$ occurs Theorem 11 can be applied (and the results about circular universality that finish this section). Whenever uu^R or u^Ru occur in w, the results of Proposition 15 are applicable.

Another generalisation of Theorem 11 is to investigate concatenations under permutations: for a morphic permutation π of Σ can we compute $\iota(w\pi(w))$?

Lemma 17 (∗). *Let $\pi : \Sigma^* \to \Sigma^*$ be a morphic permutation. Then $\iota(w) = \iota(\pi(w))$ for all $w \in \Sigma^*$ and especially the factors of the arch factorisation of w are mapped by π to the factors of the arch factorisation of $\pi(w)$.*

By Lemma 17 we have $2\iota(w) \leq \iota(w\pi(w)) \leq 2\iota(w)+1$. Consider the universal word $w = \mathtt{abcba}$. For $\pi(\mathtt{a}) = \mathtt{c}$, $\pi(\mathtt{b}) = \mathtt{b}$, and $\pi(\mathtt{c}) = \mathtt{a}$ we obtain $w\pi(w) =$ $\mathtt{abc.bac.babc}$. which is 3-universal. However, for the identity id on Σ we get that $w\,\mathrm{id}(w)$ is 2-universal. We can show exactly the case when $\iota(w\pi(w)) = 2\iota(w)+1$.

Proposition 18 (∗). *Let $\pi : \Sigma^* \to \Sigma^*$ be a morphic permutation and $w \in \Sigma^*$ with the arch factorisation $w = \mathrm{ar}_w(1)\ldots\mathrm{ar}_w(k)r(w)$ and $\pi(w)^R = \mathrm{ar}_{\pi(w)^R}(1)\ldots\ \mathrm{ar}_{\pi(w)^R}(k)r(\pi(w)^R)$ for an appropriate $k \in \mathbb{N}_0$. Then $\iota(w\pi(w)) = 2\iota(w)+1$ iff alph$(r(w)r(\pi(w)^R)) = \Sigma$, i.e. the both rests together are 1-universal.*

Proposition 18 ensures that, for a given word with a non-empty rest, we can raise the universality-index of $w\pi(w)$ by one if π is chosen accordingly.

Remark 19. Appending permutations of the word instead of its images under permutations of the alphabet, i.e. appending to w abelian equivalent words, does not lead to immediate results as the universality depends heavily on the permutation. If w is k-universal, a permutation π may arrange the letters in lexicographical order, so $\pi(w)$ would only be 1-universal. On the other hand, the universality can be increased by sorting the letters in 1-universal factors: $\mathtt{a}_1^m\mathtt{a}_2^m\ldots\mathtt{a}_{|\Sigma|}^m$ for $\Sigma = \{\mathtt{a}_1,\ldots,\mathtt{a}_{|\Sigma|}\}$ is 1-universal but $(\mathtt{a}_1\ldots\mathtt{a}_{|\Sigma|})^m$ is m-universal, for $m \in \mathbb{N}$.

In the rest of this section we present results regarding circular universality. Recall that a word w is k-circular universal if a conjugate of w is k-universal. Consider $\Sigma = \{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}\}$ and $w = \mathtt{abbccdabacdbdc}$. Note that w is not 3-universal ($\mathtt{dda} \notin \mathrm{ScatFact}_3(w)$) but 2-universal. Moreover, the conjugate $\mathtt{bbccdabacdbdca}$ of w is 3-universal; accordingly, w is 3-circular universal.

Lemma 20 (∗). *Let $w \in \Sigma^*$. If $\iota(w) = k \in \mathbb{N}$ then $k \leq \zeta(w) \leq k+1$. Moreover if $\zeta(w) = k+1$ then $\iota(w) \geq k$.*

Lemma 21 (∗). *Let $w \in \Sigma^+$. If $\iota(w) = k$ and $\zeta(w) = k+1$ then there exists $v, z, u \in \Sigma^*$ such that $w = vzu$, with $u, v \neq \varepsilon$ and $\iota(z) = k$.*

The following theorem connects the circular universality index of a word with the universality index of the repetitions of that word.

Theorem 22 (∗). *Let $w \in \Sigma^*$. If $\iota(w) = k$ and $\zeta(w) = k+1$ then $\iota(w^s) = sk+s-1$, for all $s \in \mathbb{N}$.*

The other direction of Theorem 22 does not hold for arbitrary alphabets: Consider the 2-universal word $w = \mathtt{babccaabc}$. We have that w^2 is 5-universal but w is not 3-circular universal. Nevertheless, Lemma 21 helps us show that the converse of Theorem 22 holds for binary alphabets:

Theorem 23 (∗). *Let $w \in \{\mathtt{a},\mathtt{b}\}^*$ with $\iota(w) = k$ and $s \in \mathbb{N}$. Then $\iota(w^s) = sk+s-1$ if $\zeta(w) = k+1$ and sk otherwise.*

5 On Modifying the Universality Index

In this section we present algorithms answering the for us most natural questions regarding universality: is a specific factor v of $w \in \Sigma^*$ universal? what is the minimal $\ell \in \mathbb{N}$ such that w^ℓ is k-universal for a given $k \in \mathbb{N}$? how many (and which) words from a given set do we have to concatenate such that the resulting word is k-universal for a given $k \in \mathbb{N}$? what is the longest (shortest) prefix (suffix) of a word being k-universal for a given $k \in \mathbb{N}$? In the following lemma we establish some preliminary data structures.

Lemma 24 **($*$).** *Given a word $x \in \Sigma^n$ with alph$(x) = \Sigma$, we can compute in $O(n)$ and for all $j \in [n]$*

- *the shortest 1-universal prefix of $x[j..n]$: $u_x[j] = \min\{i \mid x[j..i] \text{ is universal}\}$,*
- *the value $\iota(x[j..n])$: $t_x[j] = \max\{t \mid \mathrm{ScatFact}_t(x[j..n]) = \Sigma^t\}$, and*
- *the minimal $\ell \in [n]$ with $\iota(x[j..\ell]) = \iota(x[j..|x|])$: $m_x[j] = \min\{i \mid \mathrm{ScatFact}_{t_x[j]}(x[j..i]) = \Sigma^{t_x[j]}\}$.*

The data structures constructed in Lemma 24 allow us to test in $O(1)$ time the universality of factors $w[i..j]$ of a given word w, w.r.t. alph$(w) = \Sigma$: $w[i..j]$ is Σ-universal iff $j \geq u_w[i]$. The combinatorial results of Sect. 4 give us an initial idea on how the universality of repetitions of a word relates to the universality of that word: Theorem 22 shows that in order to compute the minimum s such that w^s is ℓ-universal, for a given *binary* word w and a number ℓ, can be reduced to computing the circular universality of w. Unfortunately, this is not the case for all alphabets, as also shown in Sect. 4. However, this number s can be computed efficiently, for input words over alphabets of all sizes. While the main idea for binary alphabets was to analyse the universality index of the conjugates of w (i.e., factors of length $|w|$ of ww), in the general case we can analyse the universality index of the suffixes of ww, by constructing the data structures of Lemma 24 for $x = ww$. The problem is then reduced to solving an equation over integers in order to identify the smallest ℓ such that w^ℓ is k-universal.

Proposition 25 **($*$).** *Given a word $w \in \Sigma^n$ with alph$(w) = \Sigma$ and $k \in \mathbb{N}$, we can compute the minimal ℓ such that w^ℓ is k-universal in $O(n + \frac{\log k}{\log n})$ time.*

We can extend the previous result to the more general (but less motivated) case of arbitrary concatenations of words from a given set, not just repetitions of the same word. The following preliminary results can be obtained. In all cases we give the number of steps of the algorithms, including arithmetic operations on $\log k$-bit numbers; the time complexities of these algorithms is obtained by multiplying these numbers by $O(\frac{\log k}{\log n})$.

1. Given the words $w_1, \ldots, w_p \in \Sigma^*$ with $|w_1 \cdots w_p| = n$ and alph$(w_1 \cdots w_p) = \Sigma$, and $k \in \mathbb{N}$, we can compute the minimal ℓ for which there exist $\{i_1, \ldots, i_\ell\} \subseteq [k]$ such that $w_{i_1} \cdots w_{i_\ell}$ is k-universal in $O(2^{3|\Sigma|}p^2 \log \ell + n)$ steps.

2. Given $k \in \mathbb{N}$ and $w_1, \ldots, w_p \in \{\mathtt{a}, \mathtt{b}\}^*$ with $\mathrm{alph}(w_1 \cdots w_p) = \{\mathtt{a}, \mathtt{b}\}$ and $|w_1 \cdots w_p| = n$, we can compute the minimal ℓ for which there exist $\{i_1, \ldots, i_\ell\} \subseteq [k]$ such that $w_{i_1} \cdots w_{i_\ell}$ is k-universal in $O(n + \log \ell)$ steps.
3. Given $w_1, \ldots, w_p \in \Sigma^*$, with $\mathrm{alph}(w_i) = \Sigma$ for all $i \in [p]$ and $|w_1 \cdots w_p| = n$, and $k \in \mathbb{N}$, we can compute in $O(n + p^3|\Sigma| \log \ell)$ steps the minimal ℓ for which there exist $\{i_1, \ldots, i_\ell\} \subseteq [k]$ with $w_{i_1} \cdots w_{i_\ell}$ is k-universal.

Finally, we consider the case of decreasing the universality of a word by an operation opposed to concatenation, namely the deletion of a prefix or a suffix.

Theorem 26 (∗). *Given $w \in \Sigma^n$ with $\iota(w) = m$ and a number $\ell < m$, we can compute in linear time the shortest prefix (resp., suffix) $w[1..i]$ (resp., $w[i..n]$) such that $w[i+1..n]$ (resp., $w[1..i-1]$) has universality index ℓ.*

Theorem 26 allows us to compute which is the shortest prefix (suffix) we should delete so that we get a string of universality index ℓ. Its proof is based on the data structures of Lemma 24. For instance, to compute the longest prefix $w[1..i-1]$ of w which has universality index ℓ, we identify the first $\ell + 1$ factors of the decomposition of Theorem 10, assume that their concatenation is $w[1..i]$, and remove the last symbol of this string. A similar approach works for suffixes.

References

1. Bringman, K., Künnemann, M.: Multivariate fine-grained complexity of longest common subsequence. In: Proceedings of the SODA 2018, pp. 1216–1235. SIAM (2018)
2. Chen, H.Z.Q., Kitaev, S., Mütze, T., Sun, B.Y.: On universal partial words. Electron. Notes Discrete Math. **61**, 231–237 (2017)
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press, Cambridge (2009)
4. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on Strings. Cambridge University Press, Cambridge (2007)
5. Day, J.D., Fleischmann, P., Manea, F., Nowotka, D.: k-spectra of weakly-c-balanced words. In: Hofman, P., Skrzypczak, M. (eds.) DLT 2019. LNCS, vol. 11647, pp. 265–277. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24886-4_20
6. de Bruijn, N.G.: A combinatorial problem. Koninklijke Nederlandse Akademie v. Wetenschappen **49**, 758–764 (1946)
7. de Luca, A., Glen, A., Zamboni, L.Q.: Rich, Sturmian, and trapezoidal words. Theor. Comput. Sci. **407**(1–3), 569–573 (2008)
8. Dobkin, D.P., Lipton, R.J.: On the complexity of computations under varying sets of primitives. J. Comput. Syst. Sci. **18**(1), 86–91 (1979)
9. Droubay, X., Justin, J., Pirillo, G.: Episturmian words and some constructions of de Luca and Rauzy. Theor. Comput. Sci. **255**(1–2), 539–553 (2001)
10. Elzinga, C.H., Rahmann, S., Wang, H.: Algorithms for subsequence combinatorics. Theor. Comput. Sci. **409**(3), 394–404 (2008)
11. Fleischer, L., Kufleitner, M.: Testing Simon's congruence. In: Proceedings of the MFCS 2018, volume 117 of LIPIcs, pp. 62:1–62:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2018)

12. Freydenberger, D.D., Gawrychowski, P., Karhumäki, J., Manea, F., Rytter, W.: Testing k-binomial equivalence. CoRR, abs/1509.00622 (2015)
13. Gabow, H.N., Tarjan, R.E.: A linear-time algorithm for a special case of disjoint set union. In: Proceedings of the 15th STOC, pp. 246–251 (1983)
14. Gawrychowski, P., Lange, M., Rampersad, N., Shallit, J., Szykula, M.: Existential length universality. To appear at STACS, abs/1702.03961 (2020)
15. Goeckner, B., et al.: Universal partial words over non-binary alphabets. Theor. Comput. Sci. **713**, 56–65 (2018)
16. Halfon, S., Schnoebelen, P., Zetzsche, G.: Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In: Proceedings of the LICS 2017, pp. 1–12 (2017)
17. Hebrard, J.-J.: An algorithm for distinguishing efficiently bit-strings by their subsequences. Theor. Comput. Sci. **82**(1), 35–49 (1991)
18. Holzer, M., Kutrib, M.: Descriptional and computational complexity of finite automata - a survey. Inf. Comput. **209**(3), 456–470 (2011)
19. Imai, H., Asano, T.: Dynamic segment intersection search with applications. In: Proceedings of the 25th Annual Symposium on Foundations of Computer Science, FOCS, pp. 393–402. IEEE Computer Society (1984)
20. Karandikar, P., Kufleitner, M., Schnoebelen, P.: On the index of Simon's congruence for piecewise testability. Inf. Process. Lett. **115**(4), 515–519 (2015)
21. Karandikar, P., Schnoebelen, P.: The height of piecewise-testable languages with applications in logical complexity. In: Proceedings of the CSL 2016, volume 62 of LIPIcs, pp. 37:1–37:22 (2016)
22. Karandikar, P., Schnoebelen, P.: The height of piecewise-testable languages and the complexity of the logic of subwords. Logic. Methods Comput. Sci. **15**(2) (2019)
23. Krötzsch, M., Masopust, T., Thomazo, M.: Complexity of universality and related problems for partially ordered NFAs. Inf. Comput. **255**, 177–192 (2017)
24. Kuske, D., Zetzsche, G.: Languages ordered by the subword order. In: Bojańczyk, M., Simpson, A. (eds.) FoSSaCS 2019. LNCS, vol. 11425, pp. 348–364. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17127-8_20
25. Lejeune, M., Leroy, J., Rigo, M.: Computing the k-binomial complexity of the Thue-Morse word. CoRR, abs/1812.07330 (2018)
26. Leroy, J., Rigo, M., Stipulanti, M.: Generalized Pascal triangle for binomial coefficients of words. CoRR, abs/1705.08270 (2017)
27. Lothaire, M.: Combinatorics on Words. Cambridge University Press, Cambridge (1997)
28. Maier, D.: The complexity of some problems on subsequences and supersequences. J. ACM **25**(2), 322–336 (1978)
29. Martin, M.H.: A problem in arrangements. Bull. Amer. Math. Soc. **40**(12), 859–864 (1934)
30. Mateescu, A., Salomaa, A., Sheng, Y.: Subword histories and Parikh matrices. J. Comput. Syst. Sci. **68**(1), 1–21 (2004)
31. Rampersad, N., Shallit, J., Zhi, X.: The computational complexity of universality problems for prefixes, suffixes, factors, and subwords of regular languages. Fundam. Inf. **116**(1–4), 223–236 (2012)
32. Rigo, M., Salimov, P.: Another generalization of abelian equivalence: binomial complexity of infinite words. Theor. Comput. Sci. **601**, 47–57 (2015)
33. Salomaa, A.: Connections between subwords and certain matrix mappings. Theor. Comput. Sci. **340**(2), 188–203 (2005)
34. Seki, S.: Absoluteness of subword inequality is undecidable. Theor. Comput. Sci. **418**, 116–120 (2012)

35. Simon, I.: Piecewise testable events. In: Brakhage, H. (ed.) GI-Fachtagung 1975. LNCS, vol. 33, pp. 214–222. Springer, Heidelberg (1975). https://doi.org/10.1007/3-540-07407-4_23
36. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. J. ACM **21**(1), 168–173 (1974)
37. Zetzsche, G.: The complexity of downward closure comparisons. In: Proceedings of the ICALP 2016, volume 55 of LIPIcs, pp. 123:1–123:14 (2016)