Scattered Factor-Universality of Words

Laura Barker\textsuperscript{1}, Pamela Fleischmann\textsuperscript{1}, Katharina Harwardt\textsuperscript{1}, Florin Manea\textsuperscript{*2}, and Dirk Nowotka\textsuperscript{1}

\textsuperscript{1} Kiel University, Germany, stu97347@mail.uni-kiel.de, fpa@informatik.uni-kiel.de, stu120568@mail.uni-kiel.de, dn@informatik.uni-kiel.de

\textsuperscript{2} University of Göttingen, Germany, florin.manea@informatik.uni-goettingen.de

Abstract. A word $u = u_1 \ldots u_n$ 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 $v_0, v_1, \ldots, v_n$ such that $w = v_0 u_1 v_1 u_2 \ldots u_n v_n$. 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 $\text{alph}(w) = \Sigma$, is called $k$-universal if its $k$-spectrum includes all words of length $k$ over the alphabet $\Sigma$; 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 $v_0, \ldots, v_n, u_1, \ldots, u_n$ with $u = u_1 \ldots u_n$ and $w = v_0 u_1 v_1 u_2 \ldots u_n v_n$. 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 [37,16,24], 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 more fundamental

* Supported by the DFG grant MA 5725/2-1. F.M. thanks Paweł Gawrychowski for his comments and suggestions.
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 [28,1], 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 $w = \text{aba}$ is $\{\varepsilon, a, aa, ab, b, ba\}$ and the 2-spectrum is $\{aa, ab, ba\}$. 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 $w = \text{aba}$ is not 2-universal since $\text{bb}$ is not a scattered factor of $w$, while $w' = \text{abab}$ 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 $\Sigma$) is equal to $\Sigma^*$, 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 $\ell$ and a language $L$ (over $\Sigma$), whether $L$ contains all strings of length $\ell$ over $\Sigma$. See [14] for a series of results on this problem and a discussion on its motivation, and [31,23,14] and the references therein for more results on the universality problem for various types of automata. The universality problem was also considered for words [29,6] and, more recently, for partial words [21,5] w.r.t. their factors. In this context, the question is to find, for a given $\ell$, a word $w$ over an alphabet $\Sigma$, such that each word of length $\ell$ over $\Sigma$ occurs exactly once as a contiguous factor of $w$. De Bruijn sequences [6] fulfill this property, and have been shown to have many applications in various areas of computer science or combinatorics, see [21,5] 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 [35,27,11] 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 $\ell$-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., \([32,12,20,24]\)). 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 Section 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 Section 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 \( \pi \). 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 Section 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 \( \ell \) such that \( w^\ell \) 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., \[\text{I} \text{I} \text{I} \text{I} \text{O}\). 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 \(\mathbb{N}_{\geq n} = \mathbb{N} \setminus [n-1]\). An alphabet \(\Sigma\) is a nonempty finite set of symbols called letters. A word is a finite sequence of letters from \(\Sigma\), thus an element of the free monoid \(\Sigma^*\). Let \(\Sigma^+ = \Sigma^\ast \setminus \{\varepsilon\}\), where \(\varepsilon\) is the empty word. The \textit{length} of a word \(w \in \Sigma^\ast\) is denoted by \(|w|\).

For \(k \in \mathbb{N}\) define \(\Sigma^k = \{w \in \Sigma^\ast | |w| = k\}\) and \(\Sigma^{\leq k}, \Sigma^{\geq k}\) analogously. A word \(u \in \Sigma^\ast\) is a \textit{factor} of \(w \in \Sigma^\ast\) if \(w = xuy\) for some \(x, y \in \Sigma^\ast\). If \(x = \varepsilon\) (resp. \(y = \varepsilon\)), \(u\) is called a \textit{prefix} (resp. \textit{suffix} of \(w\)). Let \(\text{Pref}_k(w)\) be the prefix of \(w\) of length \(k \in \mathbb{N}_0\). The \(i^{\text{th}}\) letter of \(w \in \Sigma^\ast\) 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 \textit{reversal} of \(w \in \Sigma^n\) by \(w^R = w[n] \ldots w[1]\). Set \(|w|_a = |\{i \in [|w|] | w[i] = a\}\} and \(\text{alph}(w) = \{a \in \Sigma | \forall a \geq 0\} \) for \(w \in \Sigma^\ast\). For a word \(u \in \Sigma^\ast\) we define \(u^\ast = \varepsilon, u^{i+1} = u^i u\) for \(i \in \mathbb{N}\). A word \(w \in \Sigma^\ast\) is called \textit{power} (repetition) of a word \(u \in \Sigma^\ast\) if \(w = u^t\) for some \(t \in \mathbb{N}_{\geq 2}\). A word \(u \in \Sigma^\ast\) is a \textit{conjugate} of \(w \in \Sigma^\ast\) if there exist \(x, y \in \Sigma^\ast\) with \(w = xy\) and \(u = yx\). A function \(\pi : \Sigma^\ast \rightarrow \Sigma^\ast\) is called \textit{morphic permutation} if \(\pi\) is bijective and \(\pi(uv) = \pi(u)\pi(v)\) for all \(u, v \in \Sigma^\ast\).

**Definition 1.** A word \(v = v_1 \ldots v_k \in \Sigma^\ast\) is a scattered factor of \(w \in \Sigma^\ast\) if there exist \(x_1, \ldots, x_{k+1} \in \Sigma^\ast\) such that \(w = x_1v_1 \ldots x_kv_kv_{k+1}\). Let \(\text{ScatFact}(w)\) be the set of all scattered factors of \(w\) and define \(\text{ScatFact}_k(w)\) (resp., \(\text{ScatFact}_{\leq k}(w)\)) as the set of all scattered factors of \(w\) of length (resp., up to) \(k \in \mathbb{N}\). A word \(u \in \Sigma^\ast\) is a common scattered factor of \(w, v \in \Sigma^\ast\), if \(u \in \text{ScatFact}(w) \cap \text{ScatFact}(v)\); 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 \(\text{ScatFact}_k(w)\) and \(\text{ScatFact}_{\leq k}(w)\) are also known as the \(k\)-spectrum and the full-\(k\)-spectrum of \(w\) resp.. Simon [35] defined the congruence \(\sim_k\) in which \(u, v \in \Sigma^\ast\) are congruent if they have the same full \(k\)-spectrum and thus the same \(k\)-spectrum. The \textit{shortlex normal form} of a word \(w \in \Sigma^\ast\) w.r.t. \(\sim_k\), where \(\Sigma\) 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 \(\Sigma\)) 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 [3] this is equivalent in the binary case to \(w \in \{ab, ba\}^k\). The following definition captures this property of a word in a generalised setting.

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

**Remark 3.** Notice that \(k\)-universality is always w.r.t. a given alphabet \(\Sigma\); the word \(abcbaba\) is 1-universal for \(\Sigma = \{a, b, c\}\) but it is not universal for \(\Sigma \cup \{d\}\). If it is clear from the context, we do not explicitly mention \(\Sigma\). The universality of the factors of a word \(w\) is considered w.r.t. \(\text{alph}(w)\).
Definition 6 (\cite{17}). A factor \( w \) is called \( \ell \)-rich if it is rich and moreover that a word is \( \ell \)-rich iff it is \( \ell \)-universal and a rich-factorisation, i.e., the factorisation of an \( \ell \)-rich word into \( \ell \) rich words, can be efficiently obtained. However, we will use the name \( \ell \)-universality rather than \( \ell \)-richness, as richness defines as well, e.g. the property of a word \( w \) to have \( n + 1 \) distinct palindromic factors, see, e.g., \cite{97}. As \( w \) is \( \ell \)-universal iff \( w \) is the concatenation of \( \ell \) rich words it follows immediately that, if \( w \) is over the ordered alphabet \( \Sigma = \{1 < 2 < \ldots < \sigma\} \) and \( \ell \)-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 \( \ell \)-universal word).

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

Definition 4. A word \( w \in \Sigma^* \) is called \( k \)-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 \cite{29,21,15}, 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 \( \Sigma^k \) occurs exactly once as scattered factor of \( w \). Indeed for \( |\Sigma| = \sigma \), if \( |w| \geq k + \sigma \) then there exists a word from \( \Sigma^k \) 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 \( a^k \in \text{ScatFact}_k(w) \) for all \( a \in \Sigma \)). 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 \( w[i]w[\sigma + 2..\sigma + k] \in \text{ScatFact}_k(w) \) 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 \cite{17}.

Definition 6 (\cite{17}). For \( w \in \Sigma^* \) the arch factorisation of \( w \) is given by \( w = \text{ar}_w(1) \ldots \text{ar}_w(k)r(w) \) for a \( k \in \mathbb{N}_0 \) with \( \text{ar}_w(i) \) is universal and \( \text{ar}_w(i)[\text{ar}_w(i)] \notin \text{alph}(\text{ar}_w(i)[1 \ldots \text{ar}_w(i)] - 1) \) for all \( i \in [n] \), and \( \text{alph}(r(w)) \subset \Sigma \). The words \( \text{ar}_w(i) \) are called archs of \( w \), \( r(w) \) is called the rest. Set \( m(w) = \text{ar}_w(1)[\text{ar}_w(1)] \ldots \text{ar}_w(k)[\text{ar}_w(k)] \) 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 \( \ell \) 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 \( \Sigma \) 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 \( \Sigma \), 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 \( \Sigma \), 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 \( a \) 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 \( w[\ell] = w[i] = a \) for some \( i > \ell \) such that \( w[j] \neq a \) 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 \( w[i_{j-1}+1..i] \). Clearly, if \( w[\ell] = w[i] = a \) for some \( i < \ell \) such that \( w[j] \neq a \) 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 than \( k + 1 \) (cf. [11 Prop. 2])). Then, to compute the shortlex normal form, in a third phase, labelled phase 2, if letters \( b > a \) 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 $\text{last}[i] = \max(\{0\} \cup \{j < i \mid w[j] = w[i]\})$.

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 $x_p \geq x_{i_t}$ 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 $i_1, i_{t-1}, \ldots$) 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 $x_\ell$ with $i - 1 \geq \ell \geq \text{last}[i]$ fulfill $x_\ell \geq x_1$ and $i_\ell \geq \text{last}[i]$. 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 $n + 1 = i_0 > i_1 > i_2 > \ldots > i_t = i$ such that the following property is invariant: $y_{i_{\ell-1}} < y_{i_\ell}$ for $1 \leq \ell \leq t$ and $y_p \geq y_{i_t}$ 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: $\{1\}, \{2\}, \ldots, \{i-1\}, [i, i_t-1], [i_t-1, i_t-2], \ldots, [i_1, i_0-1]$ for which we define an interval union-find data structure. 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
\textbf{find} operations on it in }O(n)\text{ time, with the crucial restriction that one can only unite neighboring intervals. We assume that }\textbf{find}(j)\text{ returns the bounds of the interval stored in our data structure to which }j\text{ belongs. From the definition of the list }L',\text{ it is clear that, before processing position }i\text{ (and after finishing processing position }i+1), y_i = \min(y[i+1..i_{t-1}-1])\text{ holds. We maintain a new array }\text{next}[\cdot]\text{ with }|\Sigma|\text{ elements: before processing position }i, \text{next}[w[i]]\text{ is the smallest position }j > i\text{ where }w[i]\text{ occurs after position }i,\text{ which was not eliminated (i.e., smallest }j > i\text{ with }y_j \neq +\infty),\text{ or }0\text{ if there is no such position. Position }i\text{ is now processed as follows: let }[a,b]\text{ be the interval returned by }\textbf{find(next}[i])\text{. If }a = i + 1\text{ then let }\min = y_i;\text{ if }a > i + 1\text{ then there exists }j\text{ such that }[a,b] = [i, i_{j-1} - 1]\text{ and }t > j > 0,\text{ so let }\min = y_j.\text{ Let now }y = \min + 1,\text{ and note that we should set }y_i = y,\text{ but only if }x_i + i \leq k + 1.\text{ So, we check whether }x_i + i < k + 1\text{ and, if yes, let }y_i = y\text{ and set }\text{next}[w[i]] = i;\text{ otherwise, set }y_i = +\infty\text{ (note that position }i\text{ becomes, as such, irrelevant when the }y\text{-coordinate is computed for other positions). If }y_i = +\infty\text{ then make the union of the intervals }\{i\}\text{ and }[i+1, i_{t-1} - 1]\text{ and start processing }i - 1; L'\text{ remains unchanged. If }y_i \neq +\infty\text{ then make the union of the intervals }\{i\}, [i+1, i_{t-1} - 1], \ldots, [i_{j+1}, i_j - 1]\text{ and start processing }i - 1; L'\text{ becomes }i, i_j, i_{j-1}, \ldots, i_0.\text{ As each position of }w\text{ is inserted at most once in }L',\text{ and then deleted once (never reinserted), the number of list operations is }O(n).\text{ The time needed for the update of }L',\text{ caused by the insertion of }i\text{ in }L',\text{ is proportional to the number of elements removed from }L'\text{ in that step, so the total time needed (exclusively) to process }L\text{ is }O(n).\text{ On top of that, for each position }i,\text{ we run one }\textbf{find}\text{ operation and a number of }\textbf{union}\text{ operations proportional to the number of elements removed from }L'\text{ in that step. Overall we do }O(n)\text{ }\textbf{union}\text{ and }\textbf{find}\text{ operations on the }\textbf{union-find}\text{ data structure. This takes in total, for all }i, O(n)\text{ time (including the initialisation). Thus, the time complexity of phase 1b is linear.}\n\nThirdly, phase 2. Assume that }w_0\text{ is the input word of this phase. Clearly, }|w_0| = m \leq n,\text{ and we have computed the coordinates for all its positions (and maybe eliminated some positions of the initial input word }w).\text{ We partition in linear time }O(n)\text{ the interval }[1, m]\text{ into }2t + 1\text{ (possibly empty) lists of positions }L_1, \ldots, L_{2t+1}\text{ such that the following conditions hold. Firstly, all elements of }L_i\text{ are smaller than those of }L_{i+1}\text{ for }1 \leq i \leq 2t.\text{ Secondly, for }i\text{ odd, the elements }j\text{ in }L_i\text{ have }x_j + y_j < k + 1;\text{ for each }i\text{ even, there exist }a_i, b_i\text{ such that }a_i + b_i = k + 1\text{ and for all }j\text{ in }L_i\text{ we have }x_j = a_i, y_j = b_i.\text{ Thirdly, we want }t\text{ to be minimal with these properties. We now produce, also in linear time, a new list }U:\text{ for each }i \leq t\text{ and }j \in L_{2i}\text{ we add the triplet }i, w[j], j\text{ in }U.\text{ We sort the list of triples }U\text{ (cf. [11] Prop. 10)}\text{ with radix sort in linear time [3]. After sorting it, }U\text{ can be decomposed in }t\text{ consecutive blocks }U_1, U_2, \ldots, U_t,\text{ where }U_i\text{ contains the positions of }L_{2i}\text{ sorted w.r.t. the order on }\Sigma\text{ (i.e., determined by the second component of the pair). As such, }U_i\text{ induces a new order on the positions of }w_0\text{ stored in }L_{2i}.\text{ We can now construct a word }w_1\text{ by just writing in order the letters of }w_0\text{ corresponding to the positions stored in }L_i,\text{ for }i\text{ from }1\text{ to }2t + 1,\text{ such that the letters of }L_i\text{ are written in the original order, for }i\text{ odd, and in
Remark 9. Theorem 8 allows us to decide in linear time whether a word $w$ over $\Sigma$ 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 $\ell$ of archs in the arch factorisation. For a lighter notation, we use $u_i = \text{ar}_w(i)$ 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 $\Sigma$ 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 if $u_i$ contained all letters of $\Sigma$, so $|u_i| \geq |\Sigma|$. The conclusion follows.

The following combinatorial result characterise universality by repetitions.

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 $(1 \cdot 2 \cdots \sigma)^k$.

This improves the complexity of the algorithm reported in [11], where the problem was solved in $O(|\Sigma| \log |\Sigma| + n)$ 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 $\Sigma$ are equal, and this requires $\Theta(|\Sigma| \log |\Sigma|)$ time [5].
Theorem 11. A word $w \in \Sigma^{\geq k}$ with $\text{alp}(w) = \Sigma$ is $k$-universal for $k \in \mathbb{N}_0$ iff $\text{ScatFact}_k(w^n) = \text{ScatFact}_k(w^{n+1})$ for an $n \in \mathbb{N}$. Moreover we have $\iota(w^n) \geq kn$ if $\iota(w) = k$.

Proof. Assume firstly $w$ to be $k$-universal, i.e. we have $\text{ScatFact}_k(w) = \Sigma^k$. This implies $\Sigma^k \subseteq \text{ScatFact}_k(w^n)$ for all $n \in \mathbb{N}$. On the other hand we have $\text{ScatFact}_k(w^n) \subseteq \Sigma^k$ and thus $\text{ScatFact}_k(w^n) = \Sigma^k = \text{ScatFact}_k(w^{n+1})$ for all $n \in \mathbb{N}$. For the second direction assume for a fixed $n \in \mathbb{N}$, $\text{ScatFact}_k(w^n) = \text{ScatFact}_k(w^{n+1})$. We prove firstly that $\text{ScatFact}_k(w^n) = \Sigma^k$ holds. Let $v \in \Sigma^*$. If $v = \varepsilon$, we have $v \in \text{ScatFact}_k(w^n)$. Let $|v| = \ell \in \mathbb{N}$ and assume that $\Sigma^\ell - 1 \subseteq \text{ScatFact}_k(w^n)$. Thus $v[1 \ldots \ell - 1] \in \text{ScatFact}_k(w^n)$. By $\text{alp}(w) = \Sigma$ we have $v \in \text{ScatFact}_k(w^{n+1}) = \text{ScatFact}_k(w^n)$. If $n = 1$ we have immediately $\text{ScatFact}_k(w) = \Sigma^k$ and thus $w$ is $k$-universal. Consider $n \in \mathbb{N}_{\geq 2}$. Suppose now $\text{ScatFact}_k(w^{n-1}) \subseteq \Sigma^k$. Let $w^{n-1} = \text{ar}_{\omega^{n-1}}(1) \ldots \text{ar}_{\omega^{n-1}}(\ell) r(w^{n-1})$ be the arch factorisation of $w^{n-1}$ for an appropriate $\ell \in [k - 1]$. Choose $p \in \ell$ such that $w = \text{ar}_{\omega}(1) \ldots \text{ar}_{\omega}(p)r'$ and $r'$ is a proper prefix of $\text{ar}_{\omega}(p+1)$. Then $r'$ is a suffix of $w^{n-1}$ and $w^n$. Especially $r(w)$ is a suffix of $r'$. Choose $a \in \Sigma \setminus \text{alp}(r')$, i.e. $a \notin \text{alp}(r(w))$. By [17] Proposition 2, $m[w^{n-1}]a \notin \text{ScatFact}_k(w^n)$. By $\text{alp}(w) = \Sigma$ we have on the other hand $m[w^{n-1}]a \in \text{ScatFact}_k(w^{n+1})$ - a contradiction. Thus we have $\text{ScatFact}_k(w^{n-1}) = \Sigma^k$. Inductively we get $\text{ScatFact}_k(w) = \Sigma^k$ and thus $w$ is $k$-universal.

For the second claim, we get immediately that $w^n$ is at least $kn$-universal if $\iota(w) = k$, since the arch factorisation of $w$ occurs in each $w$ of the repetition. □

As witnessed by $w = \text{aab} \in \{a, b\}^*$, $\iota(w^n)$ can be greater than $n \cdot \iota(w)$: $w$ is universal, not 2-universal but $w^2 = \text{aab} \cdot \text{ba} \cdot \text{aab} \cdot \text{b}$ 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 $w$ and $ww$ of minimal length.

Proof. Just like in the proof of Theorem [10] we compute the decomposition (arch factorisation) $w = u_1 \ldots u_k$ such that, for $i \in [k - 1]$, the factor $w[1..m_i] = u_1 \ldots u_i$ is the shortest prefix of $w$ such that $\Sigma^i \subseteq \text{ScatFact}_i(w[1..m_i])$, and $u_k$ (called in the arch factorisation the rest) either does not contain all letters of $\Sigma$ or it does, but if we remove its last letter then it does not contain anymore all letters of $\Sigma$, i.e., $\Sigma^k \subseteq \text{ScatFact}_k(w)$ but $\Sigma^k \notin \text{ScatFact}_k(w[1..n - 1])$.

If $u_k$ does not contain all letters of $\Sigma$, then $k > 1$ (as $w$ contains all letters of $\Sigma$). The procedure described in the proof of Theorem [10] identifies a letter $a$ that does not occur in $u_k$. We construct the word $x = w|m_1|w|m_2| \cdots |w|m_{k-1}|a = m(w)a$ (where $m(w)$ is defined w.r.t. the arch factorisation). Then $x$ is not a scattered factor of $w$ (and all shorter words are scattered factors of $w$), but $x$ is scattered factor of $ww$ (as $a$ occurs in $w$, because $k > 1$). Indeed, if $x$ were a scattered factor of $w$, then its $i$th letter should correspond to the letter occurring
on position \( j_i \geq m_i \) of \( w \). This is clear for \( m_1 \): if \( w[m_1] \) occurred also to the left of \( m_1 \) in \( w \), then \( u_1 \) would not be the shortest prefix of \( w[1..n] \) that contains all letters of \( \Sigma \). Then, for \( i \geq 1 \), assume the property holds for the first \( i \) letters of \( x \). We show it for \( i+1 \). So, \( x[i+1] \) should correspond to a letter of \( w \) occurring to the right of \( x[i] \). So on a position strictly greater than \( m_i \). But \( x[i+1] = w[m_{i+1}] \) occurs of the first time to the left of \( m_i \) on position \( m_i+1 \). So, our statement is correct. Now, if the \( (k-1) \)th letter of \( x \) occurs on a position greater or equal to \( m_{k-1} \), then the last letter of \( x \), namely \( a \), should occur in \( u_k = w[m_{k-1}+1..m_k] \), a contradiction.

If \( u_k \) contains all letters of \( \Sigma \), then let \( x = w[m_1]w[m_2] \cdots w[m_k]a = m(w)a \), for some \( a \in \Sigma \). Just like before, we can show that \( x \) is not a scattered factor of \( w \), but all shorter words are scattered factors of \( w \); also \( x \) is clearly a scattered factor of \( ww \).

Running the procedure described in Theorem 10 takes linear time, and constructing \( x \) also takes linear time. The conclusion follows.

\[ \square \]

**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., \[10\], 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 integer 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 \( \text{ScatFact}_k(w) = \text{ScatFact}_k(w^R) \).

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 \( u = \text{Pref}_1[w] \) 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 \( w[\lceil n/2 \rceil] \subseteq \text{alph}(r(u)) = \Sigma \).

**Proof.** Consider firstly \( |w| \equiv 0 \), i.e. \( w = uu^R \). By \( \iota(u) = k \), \( u \) has an arch factorisation with \( k \) factors which also occur in \( u^R \). This implies \( \iota(uu^R) \geq 2k \). Suppose \( \iota(uu^R) = 2k+1 \). Let \( uu^R = \alpha_{uu^R}(1) \cdots \alpha_{uu^R}(2k+1)r(ww^R) \) be the arch factorisation. Since \( k \) is maximal, \( \alpha_{uu^R}(1) \cdots \alpha_{uu^R}(k+1) \) is not a prefix of \( u \), i.e. \( \alpha_{uu^R}(k+2) \) is a factor of \( u^R \) and thus \( \alpha_{uu^R}(k+2) \alpha_{uu^R}(2k+1) \alpha_{uu^R}(k+1) \) is a suffix of \( u^R \). Hence we get \( \alpha_{uu^R}(k+1) = r(u) \) for a prefix \( y \) of \( u^R \). If \( |r(u)| = |y| \) we have \( r(u) = y^R \) and thus \( \Sigma = \text{alph}(\alpha_{uu^R}(k+1)) \subseteq \text{alph}(r(u)) \subset \Sigma \). If \( |r(u)| < |y| \), then \( r(u) \) is a prefix of \( y \). This implies \( \Sigma = \text{alph}(\alpha_{uu^R}(k+1)) = \text{alph}(y) \) and consequently we found an arch factorisation of \( w \) (the second one) with \( k+1 \) factors. Consider now \( |r(u)| > |y| \). Then \( y^R \) is a suffix of \( r(u) \) but by
the definition of the arch factorisation \( y||y\| \) does not occur in \( r(u)[1\ldots|r(u)|-1] \). Since we get a contradiction in all three cases, the claim is proven for even-length palindromes.

By a similar argument odd-length palindromes have to have exactly the letter in the middle which is missing in \( r(u) \) to be 1-universal.

\[ \square \]

Remark 16. If we consider the universality of a word \( w = w_1\ldots w_m \) for \( m \in \mathbb{N} \) with \( w_i \in \{u, uR\} \) for a given word \( u \in \Sigma^\ast \), then a combination of the previous results can be applied. Each time either \( u^2 \) or \( (uR)^2 \) occurs Theorem 11 can be applied (and the results about circular universality that finish this section). Whenever \( uuR \) or \( uRu \) 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 \( \pi \) of \( \Sigma \) can we compute \( \iota(w\pi(w)) \)?

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

**Proof.** Let \( w \in \Sigma^\ast \) and \( w = ar_w(1)\ldots ar_w(k)r(w) \) be the arch factorisation of \( w \) for an appropriate \( k \in \mathbb{N}_0 \). By the definition of the arch factorisation \( ar_w(i)[|ar_w(i)|] \) does not occur in \( ar_w(i)[1\ldots|ar_w(i)|-1] \) for all \( i \in [k] \). Set \( k_i = \sum_{j=1}^{i} |ar_w(j)| \) for \( i \in [k] \). Thus \( \pi(ar_w(i))[|ar_w(i)|] \) occurs only once in \( \pi(w)[k_i + 1\ldots k_{i+1}] \) and exactly as the last letter. Hence \( \pi(ar_w(1))\ldots \pi(ar_w(k))\pi(r(w)) \) is the arch factorisation of \( \pi(w) \). The other direction follows by applying \( \pi^{-1} \) as a permutation to \( \pi(w) \).

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

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

**Proof.** Consider firstly that \( r(w)r(\pi(w)^R) \) is 1-universal. Then we get

\[ w\pi(w) = ar_w(1)\ldots ar_w(k)r(w).r(\pi(w)^R)^R(ar_{\pi(w)^R}(k))^R \ldots (ar_{\pi(w)^R}(1))^R. \]

Since all archs are 1-universal by definition, the assumption implies that \( w\pi(w) \) is \( (2\iota(w) + 1) \)-universal and thus \( \iota(w\pi(w)) \geq 2\iota(w) + 1 \). The equality follows by the definition of \( \iota \). For the other direction assume \( \iota(w\pi(w)) = 2\iota(w) + 1 \). Here, we get the arch factorisation

\[ w\pi(w) = ar_{w\pi(w)}(1)\ldots ar_{w\pi(w)}(2\iota(w) + 1)r(w\pi(w)). \]
This implies
\[
\begin{align*}
\ar_{w\pi(w)}(1)\ldots\ar_{w\pi(w)}(2\iota(w) + 1)r(w\pi(w)) &= \ar_{w}(1)\ldots\ar_{w}(k)r(w)r(\pi(w)^R)(\ar_{\pi(w)}\pi(k))^{R}\ldots(\ar_{\pi(w)}\pi(1))^{R}.
\end{align*}
\]
By \(\iota(w) = k\) only the first \(k\) archs can be contained in \(w\). This implies that \(r(w)\) is a prefix of \(\ar_{w\pi(w)}(k + 1)\). Choose \(y \in \Sigma^+\) with \(\ar_{w\pi(w)}(k + 1) = r(w)y\).

By Lemma \[\text{17}\] we have \(\ar_{w}(i) = \pi(\ar_{\pi(w)}(i))\) and thus \(y = r(\pi(w)^R)y\). By \(\ar_{w\pi(w)}(k + 1) = \Sigma\) the claim is proven.

\(\square\)

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

\textbf{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 \(\pi\) 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: \(a_1^n\) \(a_2^n\ldots a_{|\Sigma|}^n\) for \(\Sigma = \{a_1, \ldots, a_{|\Sigma|}\}\) is \(1\)-universal but \((a_1\ldots a_{|\Sigma|})^m\) 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 \(\Sigma = \{a, b, c, d\}\) and \(w = abccddabacddbc\). Note that \(w\) is not \(3\)-universal (\(\text{dda} \notin \text{ScatFact}_3(w)\)) but \(2\)-universal. Moreover, the conjugate \(bbccddabacddc\) of \(w\) is \(3\)-universal; accordingly, \(w\) is \(3\)-circular universal.

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

\textbf{Proof.} Since \(w\) is a conjugate of itself, \(w\) is at least \(k\)-circular universal. Suppose \(\zeta(w) = k + 2\). Choose \(x, y \in \Sigma^+\) with \(w = xy\) and \(yx = \ar_{yx}(1)\ldots\ar_{yx}(k + 2)r(yx)\). Since \(\iota(w) = k\) there is no \(i\) such that \(y = w_1\ldots w_i\) (otherwise \(w = xy\) would be \((k + 1)\)-universal). Thus there exists a \(j \in [k + 2]\) and a proper prefix \(y_j\) of \(y\) such that \(y = w_1\ldots w_{j-1}y_j\); let \(x_1\) be such that \(w_j = y_jx_1\). This implies \(w = xy = x_1w_{j+1}\ldots w_{k+2}w_1\ldots w_1y_1\) and we get that \(k + 1\) archs are contained in \(w\). This contradicts the maximality of \(k\).

For the second claim let \(w = xy\) and \(yx = \ar_{yx}(1)\ldots\ar_{yx}(k + 1)r(yx)\). If \(y\) contains all archs then \(\iota(w) = k + 1\). If \(y\) does not contain all archs, there exists an \(i \in [k + 1]\) such that a prefix of \(\ar_{yx}(i)\) is a suffix of \(y\) and the corresponding suffix of \(\ar_{yx}(i)\) is a prefix of \(x\). Thus \(\ar_{yx}(1)\ldots\ar_{yx}(i-1)\ar_{yx}(i+1)\ar_{yx}(k+1)\) is a scattered factor of \(w\). \(\square\)

\textbf{Lemma 21.} \(w \in \Sigma^+\). If \(\iota(w) = k\) and \(\zeta(w) = k + 1\) then there exists \(v, z, u \in \Sigma^+\) such that \(w = vz\), with \(u, v \neq \varepsilon\) and \(\iota(z) = k\).
Proof. By \( \zeta(w) = k + 1 \) there exist \( x, y \in \Sigma^* \) with \( w = xy \) and \( yx = \ar_yw(1) \ldots \ar_yw(k + 1)r(yx) \). Since \( \iota(w) = k \) there is no \( i \) such that \( y = w_1 \cdots w_i \) (otherwise \( w = xy \) would be \((k + 1)\)-universal). Thus, there exists \( i \in [k + 1]_0 \) with \( w_{i + 1} = w \) and \( u \) is a proper and non-empty suffix of \( y \) and \( v \) is a proper and non-empty prefix of \( x \) with \( \text{alph}(u), \text{alph}(v) \subset \Sigma \). This implies
\[
yw = xy = v\ar_w(i + 2) \ldots \ar_w(k + 1)\ar_w(1) \ldots \ar_w(i)u.
\]
Let \( z = \ar_w(i + 2) \ldots \ar_w(k + 1)\ar_w(1) \ldots \ar_w(i) \). Clearly, \( z \) contains 1-universal words, so \( \iota(z) \geq k \). By \( \iota(w) = k \) follows immediately \( \iota(z) \leq k \).

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 \( s \in \mathbb{N} \).

**Proof.** By Lemma 21 there exist \( v, z, u \in \Sigma^* \) with \( w = vzu \), \( \iota(z) = k \), and \( \text{alph}(v), \text{alph}(u) \subset \Sigma \). Consequently we have that
\[
w^s = (vzu)^s = (zuv)^{s-1}zu
\]
is \( ((s - 1)(k + 1) + k) \)-universal, thus \( \iota(w^s) \geq (sk + s - 1) \). Since \( \iota(w) = k \) and \( w^s \) only contains \( s - 1 \) transitions from one \( w \) to another, \( w^s \) cannot have a higher universality.

The other direction of Theorem 22 does not hold for arbitrary alphabets: Consider the 2-universal word \( w = \text{babccaabc} \). 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 \( w \in \{a, b\}^* \) with \( \iota(w) = k \) and \( s \in \mathbb{N} \). Then \( \iota(w^s) = sk + s - 1 \) if \( \zeta(w) = k + 1 \) and \( sk \) otherwise.

**Proof.** By Theorem 22 it suffices to prove \( \zeta(w) = k + 1 \) if \( w^s \) is \((sk + 1)\)-universal. Assume \( \iota(w) \geq sk + 1 \). If for all conjugates \( v \) of \( w \) we have \( v[1] \neq v[|w|] \) then \( w \) is of even length and we have \( w = (ab)^k \) or \( w = (ba)^k \); this implies immediately \( \zeta(w) = k \) by the arch factorisation. Thus we know that there exists a conjugate \( v \) of \( w \) with \( v[1] = v[|w|] \). Since \( w^s \) is a conjugate of \( w^s \) and \( w^s \) is \((sk + 1)\)-universal, \( v^s \) is \((sk + 1)\)-circular universal. By Lemma 21 we get that \( v^s \) is \((sk)\)-universal and by Theorem 22 follows that \( v^{2s} \) is \((2sk + 1)\)-universal. By [3] Theorem 4 \( v^{2s} \) contains \( 2sk + 1 \) disjoint occurrences of \( ab \) or \( ba \). By \( v[1] = v[n] \) none of these occurrences can start in one \( v \) and end in the following. This implies that one \( v \) contains \( k + 1 \) of these occurrences and therefore \( \iota(v) \geq k + 1 \). Hence we get \( \zeta(w) = k + 1 \).
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 \( w^\ell \) 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 \( \text{alph}(x) = \Sigma \), 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 \( \iota(x[j..n]) : t_x[j] = \max \{ t \mid \text{ScatFact}_t(x[j..n]) = \Sigma^\ell \} \), and
- the minimal \( \ell \in [n] \) with \( \iota(x[j..\ell]) = \iota(x[j..|x|]) : m_x[j] = \min \{ i \mid \text{ScatFact}_i[j] \} \)
\((x[j..i]) = \Sigma^{t_x[j]} \).\)

**Proof.** Let for each \( j \in [n] \) and letter \( a \in \Sigma \), denote \( g_a[j] = \min \{ i \mid i \geq j, w[i] = a \} \) (by convention, \( g_a[j] = +\infty \) if \( a \) does not occur in \( w[j..n] \)). Clearly, \( u_x[j] = \max \{ g_a[j] \mid a \in \Sigma \} \) holds, i.e., \( u_x[j] \) is the end position of the shortest word starting on position \( j \) in \( x \) which contains all letters of \( \Sigma \) (the value \( g_a[j] \) is strongly related to the value \( X_a(w[j..n]) \) - read "next \( a \) in \( w[j..n] \)" used in \([11]\) to denote the first position of \( a \) in \( w[j..n] \)). It is essential to note that we will not compute all the values \( g_a[j] \), but only the values \( u_x[j] \), for all \( j \). As such, \( x[j..u_x[j]] \) is the shortest universal prefix of \( x[j..n] \).

Computing the elements of \( u_x[\cdot] \) is done as follows: let \( C \) be an array with \( |\Sigma| \) elements, all initialised to 0. As \( \Sigma \) is considered to be the set of numbers \( \{1, \ldots, |\Sigma|\} \), we will consider that \( C \) is indexed by the letters of \( \Sigma \). Also, initialise the variable \( h \) with \( |\Sigma| \).

While \( h > 0 \), we consider the positions \( j \) of \( x \) from the right to the left, i.e., from \( n \) downwards. When reading \( x[j] \), we set \( C[x[j]] = j \), and if \( C[x[j]] \) was 0 before setting it to \( j \), then we decrement \( h \) by 1. As soon as we have \( h = 0 \) we stop. At this point we have \( C[a] = g_a[j] \) for all \( a \in \Sigma \), so \( C[a] \) is the leftmost occurrence of \( a \) to the right of \( j \), and \( x[j..n] \) is the shortest suffix of \( x \) that contains all letters of \( \Sigma \). We can set \( u_x[j'] = +\infty \), for all \( j' > j \), and \( u_x[j] = \max \{ C[a] \mid a \in \Sigma \} \).

Now let \( m = u_x[j] \), and \( d = j + 1 \) (\( x[d..n] \) is the longest suffix of \( x \) which is not universal).

For \( i \) from \( j - 1 \) downto 1 we do the following. If \( m \neq C[x[i]] \) (i.e., \( x[i] \) is not the same as the letter whose leftmost occurrence in \( x[i+1..n] \) was the rightmost among all letters of \( \Sigma \)), we just set \( C[x[i]] = i \). If \( m = C[x[i]] \) (i.e., \( x[i] \) is the same as the letter whose leftmost occurrence in \( x[i+1..n] \) was the rightmost among all letters of \( \Sigma \)), we first set \( C[x[i]] = i \) and then we need to recompute \( m \), the maximum of \( C \) (the position of the letter whose first occurrence in \( x[i..n] \) is the rightmost among all letters). To do this, we decrement \( m \) by 1 repeatedly, until it reaches a value \( p \) such that \( C[x[p]] = p \). At that point, \( m = p \) is the
leftmost position on which the letter \( x[m] \) occurs in \( x[i..n] \), and all letters of \( \Sigma \) occur in \( x[i..m] \). In this way, we ensure that \( C[a] = g_a[i] \) for all \( a \in \Sigma \) and \( m \) points to the maximum element of \( C \). In both cases, we set \( u_x[i] = m \), and repeat the process for \( i - 1 \).

At the end of the computation described above, we computed \( u_x[j] \) for every position \( j \) of \( x \), i.e., we know for each position \( j \) of \( x \) the shortest universal prefix of \( x[j..n] \). The computation described above runs in time \( O(n) \). For each value \( j \) we set \( C[x[j]] \) in constant time and then, if needed, recompute the value of \( m \); this last part is not carried in constant time for each \( j \), but in total \( m \) traverses only once the entire word \( x \) from right to left, so, summing the time spent to update \( m \) for all values of \( j \), we still get \( O(n) \) time in total.

We now move on to the main phase of our algorithm. For \( j \in [n] \), we want to compute \( t_x[j] = \max \{ t \mid \text{ScatFact}_x(x[j..n]) = \Sigma^t \} \) and \( m_x[j] = \min \{ i \mid \text{ScatFact}_{t_x[j]}(x[j..i]) = \Sigma^{t_x[j]} \} \).

We show how to compute \( m_x[j] \) and \( t_x[j] \) for all positions \( j \) of \( x \), in \( O(n) \) total time, by a simple dynamic programming algorithm. For \( j \geq d \), we have \( t_x[j] = 0 \) and \( m_x[j] = u_x[j] \). For smaller values of \( j \), we have \( m_x[j] = u_x[j] + m_x[u_x[j] + 1] \) and \( t_x[j] = 1 + t_x[u_x[j] + 1] \). Indeed, the maximum exponent \( t_x[j] \) such that \( \Sigma^{t_x[j]} = \text{ScatFact}_{t_x[j]}(x[j..n]) \) is obtained by taking the shortest prefix \( x[j..u_x[j]] \) of \( x[j..n] \) that contains all letters of \( \Sigma \) and returning \( 1 \) plus the maximum exponent \( t_x[u_x[j] + 1] \) such that \( \Sigma^{t_x[u_x[j] + 1]} \) is included in the set of scattered factors of the suffix \( x[u_x[j] + 1..n] \) that follows \( x[j..u_x[j]] \). The value \( m_x[j] \) is computed according to a similar idea. Clearly, computing \( m_x[j] \) and \( t_x[j] \) takes constant time for each \( j \), so linear time overall.

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. \( \text{alph}(w) = \Sigma \): \( w[i..j] \) is \( \Sigma \)-universal iff \( j \geq u_w[i] \). The combinatorial results of Section \[4\] 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 \( \ell \)-universal, for a given binary word \( w \) and a number \( \ell \), can be reduced to computing the circular universality of \( w \). Unfortunately, this is not the case for all alphabets, as also shown in Section \[4\]. 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 \( \ell \) such that \( w^\ell \) is \( k \)-universal.

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

**Proof.** Consider the word \( x = ww \). In a preprocessing phase, using Lemma \[24\] we compute in \( O(|x|) = O(n) \) time the values \( t_x[j] \) and \( m_x[j] \) for \( j \in [2n] \).

We want to compute the minimum \( \ell \) such that \( w^\ell \) is \( k \)-universal. The general idea is the following: for \( p \geq 1 \), we compute the largest value \( i_p \) such that
\( \Sigma^i_p = \text{ScatFact}_i(w^p) \) as well as the shortest prefix \( w^{p-1}w[1..s_p] \) of \( w^p \) which is \( i_p \)-universal (as each \( w \) contains all letters of \( \Sigma \), it is clear that the shortest prefix of \( w^p \) which is \( i_p \)-universal must extend inside the \( p^{th} \) \( w \)). These values can be computed for a certain \( p \) using the corresponding values for \( p - 1 \) and the arrays we constructed in the preprocessing phase: \( i_p = i_{p-1} + t[x][s_{p-1}] \) and \( s_p = s_{p-1} + m[x][s_{p-1}] - n \). Essentially, for each \( p \), we just extend to the right in \( w^p \), as much as we can, the shortest prefix with the desired property constructed for \( w^{p-1} \). In a simple version of our algorithm we could do that until \( i_p \geq k \) (which happens after at most \( k \) iterations), and return \( p \) as the value we are searching for. However, this would lead to an algorithm with running time \( O(n + \ell \log k/\log n) \) (where the \( \log k/\log n \) factor comes from the fact that the operands in each addition \( i_p = i_{p-1} + t[x][s_{p-1}] \) may have up to \( \log k \) digits). As \( \ell \leq k \) and it is natural to assume that \( k \) is given in its binary representation, this algorithm could be exponential in the worst case.

We can optimise the idea above to work faster by exploiting the periodicity that occurs in the sequence \( (s_p)_{p \in \mathbb{N}} \), defined for the repetitions of word \( w \). By the pigeonhole principle, there always exist \( p_1, p_2 \leq n + 1 \) such that \( s_{p_1} = s_{p_2} \). So, while \( p \leq n + 1 \) we compute \( i_p \) and \( s_p \), as above, but keep track of the values taken by \( s_p \) and stop this loop as soon as the current \( s_p \) has the same value as some previously computed \( s_{p'} \) or \( i_p \geq k \) (in the latter case, we proceed as above, and return \( p \) as the value \( \ell \) we look for). More precisely, we use an array \( S \) with \( n \) elements, all set initially to 0. After computing \( s_p \), if \( S[s_p] = 0 \) then we set \( S[s_p] = p \); if \( S[s_p] \neq 0 \) we proceed as follows. We stop the loop and compute two values \( p_1 = S[s_p] \) and \( p_2 = p \). It is immediate that \( p_2 \) is the smallest \( p \) such \( s_{p_1} = s_p \) and there are no other \( p, p' < p_2 \) such that \( s_p = s_{p'} \). Computing \( p_1 \) and \( p_2 \) takes \( O(n) \) time. Note that all arithmetic operations we did so far are done on numbers that fit in constant memory.

Assume now that we have computed \( p_2 = p_1 + \delta \) and \( i_{p_2} = i_{p_1} + d \). It is clear that, for all \( j \geq 0 \), we have \( s_{p_1+jd} = s_{p_1} \) and \( i_{p_1+jd} = i_{p_1} + jd \). Now, let \( m = k - i_{p_1} \) and \( g = \lfloor \frac{m}{\delta} \rfloor \). Computing these numbers takes \( O(\log k/\log n) \) time. Let \( p_3 = p_1 + g\delta \) (again, we need \( O(\log k/\log n) \) time to compute \( p_3 \)). We have \( s_{p_3} = s_{p_1} \) and \( i_{p_3} = i_{p_1} + gd \leq k \) (these operations take \( O(\log k/\log n) \) time). Also, \( i_{p_3+d} > k \). Let \( z = k - i_{p_3} \) and \( s_{p_3} \leq d \). So, for \( p \) from \( p_3 \) to \( p_3 + \delta \) we proceed as follows. If \( i_p - i_{p_3} \geq z \) (i.e., \( i_p \geq k \)), return \( p \) as the value \( \ell \) we search for. Otherwise, compute \( i_{p+1} - i_{p_3} = (i_p - i_{p_3}) + t[s_p] \) (in time \( O(1) \) as it can be done with only adding numbers which are smaller than \( d \)) and \( s_{p+1} = s_p + m[s_p] - n \), and iterate. Because we certainly reach, in this loop, at least one \( p \) such that \( i_p \geq k \), and \( \delta \leq n \), the execution of the loop takes \( O(n) \) time.

Hence, we get the smallest \( \ell \) such that \( w^\ell \) is \( k \)-universal (i.e., \( i_{\ell} \geq k \)), in \( O(n + \log k/\log n) \) 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.
For $\ell, n \in \mathbb{N}$ and $w_1, \ldots, w_n \in \Sigma^*$, define $\langle w_1, \ldots, w_n \rangle$ as the set of all words $w = x_1 \ldots x_\ell$ with $x_i \in \{w_1, \ldots, w_n\}$ and $\langle w_1, \ldots, w_n \rangle = \bigcup_{\ell \in \mathbb{N}} \langle w_1, \ldots, w_n \rangle_\ell$.

**Definition 26.** Let $n \in \mathbb{N}$. The set $S = \{w_1, \ldots, w_n | w_i \in \Sigma^*, i \in [n]\}$ is $k$-universal if there exists $u \in \langle w_1, \ldots, w_n \rangle$ such that $u$ is $k$-universal.

Firstly we need introduce some notation for convenience and to prove an auxiliary lemma. To each $S \subseteq \Sigma$ we associate a word $u_S$ with $|u_S| = |S|$ and $\alpha(u_S) = S$ (i.e., $u_S$ is a linear ordering of the letters from $S$). Following the notations from Lemma 24, for a word $x$, let $t_x = \max\{t \in \mathbb{N} \mid \text{ScatFact}_t(x) = \Sigma^t\}$ and $m_x = \min\{i \in \mathbb{N} \mid \text{ScatFact}_i(x[1..i]) = \Sigma^i\}$; clearly, if $t_x = 0$, then $m_x = 0$, too. Note now that, for a word $u$ with $\alpha(u) = S$ and $|u| = |S|$, we have $t_{u_S} = t_u$ and $m_{u_S} = m_u$, for all $w \in \Sigma^*$. Consider $w_1, \ldots, w_p \in \Sigma^*$, and take $j \in [p]$. For $\ell \in \mathbb{N}$ and $S' \subseteq \Sigma$, we define $\max_\ell(S, j, S') = \max\{t_w \mid w = u_sw_jw_j', w' \in \langle w_1, \ldots, w_p \rangle_\ell-1 \text{ and } \alpha(w|m_w + 1..|w|) = S'\}$. By the remarks regarding the choice of the word $u_S$, $\max_\ell(S, j, S')$ is clearly well defined.

**Lemma 27.** For $w_1, \ldots, w_p \in \Sigma^*$, $S \subseteq \Sigma$, $\ell \in \mathbb{N}_{\geq 2}$, and all $\ell' \in [\ell-1]$, we have $\max_\ell(S, j, S') = \max\{\max_\ell(S, k, S'') + \max_{\ell - \ell'}(S'', j, S') \mid k \in [p], S'' \subseteq \Sigma\}$.

**Proof.** Let $\ell'$ be a natural number such that $1 \leq \ell' < \ell$. Let $i_1, \ldots, i_\ell \in [p]$ such that $i_\ell = j$, $\max_\ell(S, j, S') = t_w$, and $\alpha(w|m_w + 1..|w|) = S'$, for $w = w_{i_1}w_{i_2} \cdots w_{i_\ell}$. Take $x' = w_{i_1}w_{i_2} \cdots w_{i_\ell'}$, $x'' = \alpha(x'|m_{x'}|x'|)$, and $x''' = w_{i_{\ell'-1}}w_{i_\ell'} \cdots w_{i_1}$. It is not hard to see that $\max_\ell(S, j, S') = t_{x'''}$.

Assume that $\max_\ell(S, i_\ell, S''') > t_{x'''}$. Let $h_{i_1}, \ldots, h_{i_\ell} \in \Sigma^*$ be such that $h_{i_\ell} = i_\ell$, $\max_\ell(S, h_{i_\ell}, S''') = t_{x'''}$, and $\alpha(w'|m_{w'} + 1..|w'|) = S'$, for $w' = w_{i_1}w_{i_2} \cdots w_{i_\ell}$. Then, for $w'' = w_{i_1}w_{i_2} \cdots w_{i_\ell}w_{i_{\ell'-1}} \cdots w_{i_1}$, we have $t_{w''} > t_w = \max_\ell(S, j, S')$, a contradiction. Thus, $\max_\ell(S, i_\ell, S''') = t_{x'''}$. We can similarly show that $\max_{\ell - \ell'}(S'', j, S') = t_{x'''}$.

Assume now that there exists $r \in [p]$ and $T \subseteq \Sigma$ such that $\max_{\ell - \ell'}(T, j, S') > t_{x'''}$. Let $j_1, \ldots, j_\ell \in [p]$ and $w_{j_1}, \ldots, w_{j_\ell} \in \Sigma^*$ be such that $j_{\ell'} = r$, $j_{\ell'} = j$, $\max_{\ell - \ell'}(T, j_{\ell'}, S') = t_{x'''}$, and $\alpha(w|x'|m_x + 1..|x'|) = T$, for $x = w_{j_{\ell'+1}} \cdots w_{j_{\ell'}}$, and $\max_{\ell - \ell'}(T, j_{\ell'}, S') = t_{y''}$ and $\alpha(y|m_y + 1..|y|) = S'$, for $y = w_{j_{\ell'+1}} \cdots x_{j_{\ell'}}$. Then, clearly, for $v = w_{j_1} \cdots x_{j_{\ell'}}$ we have $t_v > t_w = \max_\ell(S, j, S')$, a contradiction, considering the form of $v$.

**Theorem 28.** Given $w_1, \ldots, w_p \in \Sigma^*$ with $|w_1 \cdots w_p| = n$ and $\alpha(w_1 \cdots w_p) = \Sigma$, and $k \in \mathbb{N}$, we can compute the minimal $\ell$ 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, some being arithmetic operations on numbers with log $k$ bits. The overall time complexity of our algorithm is $O(\log_k n) (2^{3|\Sigma|}p^2 \log \ell + n)$.

**Proof.** Note first that, because $\Sigma = \alpha(w_1 \cdots w_p)$, we have $\ell \leq pk$. We can now sketch the algorithm computing $\ell$. The general idea is first to construct, by
dynamic programming, concatenations of $2^e$ factors of the set \( \{ w_1, \ldots, w_p \} \), for larger and larger \( e \), until we find one such concatenation with \( 2^f \) elements that is \( k' \)-universal, for some \( k' \geq k \). That is, we compute the values \( N_e[S, S', j] = \max_{2^e}(S, i, S') \), for \( e \) from 0 until we reach an array \( N_f \) which contains a value \( N_f[\emptyset, S', j] \geq k \). Note that \( 2^f \) is the smallest power of 2 such that the concatenation of \( 2^f \) numbers is \( k \)-universal, so, consequently, \( f \leq 2\ell \). The values in each of the array \( N_e \) are computed by dynamic programming based on the values in the array \( N_{e-1} \), using Lemma 27 for \( \ell = 2^e \) and \( \ell' = 2^{e-1} \). Once this computation is completed, we use binary search to obtain the exact value of \( \ell \), as required. However, we now have the benefit that we can perform this binary search in an interval upper bounded by \( 2^f \in O(\ell) \).

In the following we describe the algorithm in details. We will evaluate its complexity first as the number of steps (including arithmetic operations on numbers with up to \( \log k \) bits) it performs. Then we compute its actual time complexity.

We start with a preprocessing phase. We construct the \( p \times |\Sigma| \) matrix \( F[,1] \), indexed by the numbers between 1 and \( p \) and the letters of \( \Sigma \) (which are numbers between 1 and \( |\Sigma| \)). We have \( F[i,a] \) is the position of the first (i.e., leftmost) occurrence of each letter \( x \in \Sigma \) in \( w_i \). This matrix can be computed as follows. Initialise all elements of \( F \) with 0. For each \( i \), we traverse \( w_i \), letter by letter, from left to right. When the letter \( x \in \Sigma \) is read on position \( j \) of \( w_i \), if \( F[i, z] = 0 \) then we set \( F[i,x] = j \). The total number of steps needed to do this is \( O(|\Sigma|p + n) \) (as it includes the initialisation of \( F \)). Similarly, we construct the \( p \times |\Sigma| \) matrix \( L[i,1] \), indexed by the numbers between 1 and \( p \) and the letters of \( \Sigma \), where \( L[i,a] \) is the position of the rightmost occurrence of each letter \( x \in \Sigma \) in \( w_i \).

We compute also in the preprocessing phase the data structures from Lemma 24, for each word \( w_i \), with \( i \in [p] \). So, we have for each word \( w_i \) the arrays \( t_{w_i}[j] = \max\{ t \mid \text{ScatFact}_{\Sigma}(w_i[j..n]) = \Sigma^t \} \) and \( m_{w_i}[j] = \min\{ g \mid \text{ScatFact}_{\Sigma}(w_i[j..g]) = \Sigma^t_{w_i}[j] \} \). This is done in \( O(n) \) steps.

Then, for each set \( S \subseteq \Sigma \) and \( i \in [p] \), we compute in \( O(|\Sigma|) \), the value \( j = \max\{ F[x,i] \mid x \in \Sigma \setminus S \} \). Basically, \( w_i[1..j] \) is the shortest prefix of \( w_i \) such that \( u_kw_i \) contains all letters of \( S \). Let \( g = m_{w_i}[j+1] \), and let \( S' \subseteq \Sigma \) be the set of letters contained by \( w_i[y+1..|w_i|] \). The set \( S' \) can be computed in \( O(|\Sigma|) \) time, by selecting in \( S' \) the letters \( x \in \Sigma \) with \( L[i,x] > g \). We set \( M_1[S,i] = (1 + t_{w_i}[j], S') \), where \( M_1 \) is an additional matrix we use. The computation of \( M_1[S,i] \), performed for a set \( S \) and a number \( i \in [p] \), takes \( O(|\Sigma|) \). So, in total we compute the matrix \( M_1 \) in \( O(2^{|\Sigma|}p) \) time. It is worth noting that if \( M_1[S,i] = (h, S') \), then \( \max_{2^e}(S, i, S') = h \).

The main phase of the algorithm follows. If there is an element \( M_1[\emptyset,i] = (h, S') \) such that \( h \geq k \), then we return \( \ell = 1 \). If not we proceed as described next.

For \( e \) natural number such that \( \log_{pk} + 1 \geq e \geq 1 \), we define the 3-dimensional array \( N_e[,\cdot,\cdot] \), whose first two indices are subsets of \( \Sigma \), and the third is a number from \( [p] \), and \( N_e[S, S', i] = \max_{2^e}(S, i, S') \). That is, \( N_e[S, S', i] \) stores the maximum \( k \) such that there exists \( k \)-universal word \( w \) which is the concatenation of \( u_S \) followed by \( 2^e \) words from \( \{ w_1, \ldots, w_p \} \), ending with \( w_i \), and, moreover,
if \( w' \) is the suffix of \( w \) that follows the shortest \( k \)-universal prefix of \( w \), then \( \text{alph}(w') = S' \). The elements \( N_e[S, S', i] \) will be computed by dynamic programming, using Lemma 27 for \( \ell = 2^e \) and \( \ell' = \frac{2^e}{2} \).

For \( e = 1 \), the elements of the array \( N_e \) are computed as follows. By Lemma 27, \( N_1[S, S', i] = \max\{ g \mid g = g_1 + g_2 \text{ where } M_1[S, j] = (g_1, S''), \text{ and } M_1[S'', i] = (g_2, S') \} \), with \( j \in [p], S'' \subseteq \Sigma \). For \( e > 1 \), we have \( N_e[S, S', i] = \max\{ g \mid g = g_1 + g_2 \text{ where } N_{e-1}[S, S'', j] = g_1 \text{ and } N_{e-1}[S'', S', i] = g_2, \text{ with } j \in [p], S'' \subseteq \Sigma \} \).

Clearly, computing each of the arrays \( N_e \) takes \( O(2^{3|\Sigma|}p^2) \). It is not hard to see that the maximum element of \( N_e \) is strictly greater than the maximum element of \( N_{e-1} \).

We stop the computation of the arrays \( N_e \) as soon as we reach such an array \( N_f \) such that there exists \( i \) and \( S' \) with \( N[\emptyset, S', i] \geq k \). We get that \( 2^{f-1} < \ell \leq 2^f \) (where \( \ell \) is the value we want to compute), so the total time needed to compute all these arrays is \( O(2^{3|\Sigma|}p^2 \log \ell) \).

Now we need to search \( \ell \) between \( b = 2^{f-1} \) and \( s = 2^f \). We will do this by an adapted binary search. Denote \( N' = N_{f-1} \) and \( N'' = N_f \). Let \( h \) be maximum such that \( b + 2^h < s \). We compute the 3-dimensional array \( N_{mid}[\cdot, \cdot, \cdot] \), indexed just as the arrays \( N_e \). We have \( N_{mid}[S, S', i] = \max\{ g \mid g = g_1 + g_2 \text{ where } N'[S, S'', j] = g_1 \text{ and } N_{k}[S'', S', i] = g_2, \text{ with } j \in [p], S'' \subseteq \Sigma \} \). If \( N_{mid} \) contains an element greater or equal to \( \ell \), we repeat this search for the same \( b \) and \( N' \), and \( s = b + 2^h \) and \( N'' = N_{mid} \). Otherwise, we repeat the search for the same \( s \) and \( N'' \), and using \( b + 2^h \) instead of \( b \) and \( N_{mid} \) instead of \( N' \). We stop the process if \( b = s - 1 \), and return \( s \). This procedure is iterated \( O(f) = O(\log \ell) \) times. Thus, computing \( \ell \) is done in \( O(2^{3|\Sigma|}p^2 \log \ell) \) steps, some of which are arithmetic operations on numbers with up to \( \log k \) bits.

The overall complexity of the algorithm is, thus, \( O(\frac{\log k}{\log n} (2^{3|\Sigma|}p^2 \log \ell) + n) \).

Note that, in the case stated in the previous theorem, computing the minimal number of words (from a given set) that should be concatenated in order to obtain a \( k \)-universal word is fixed parameter tractable w.r.t. the parameter \( \log |\Sigma| \), the size of the alphabet of the input words. If both \( p \), the number of input words, and \( \log |\Sigma| \) are constant, the algorithm runs in \( O(n + \log \ell) \) steps, which is linear w.r.t. the size of the input because \( \log \ell \leq \log(pk) = \log p + \log k \) (but the overall time is still affected by the operations on large numbers). In fact, we can give a solution with a linear number of steps for this problem in the case of words over binary alphabets (and \( p \) is not necessarily constant). The main idea is, in this case, we can show that, from an input set of words, only a constant number are useful when trying to construct a \( k \)-universal word by a minimal number of concatenations. The following result is based on the arch factorisation and Proposition 10.

**Theorem 29.** Given \( k \in \mathbb{N} \) and \( w_1, \ldots, w_p \in \{a, b\}^* \) with \( \text{alph}(w_1 \cdots w_p) = \{a, b\} \) and \( |w_1 \cdots w_p| = n \), we can compute in \( O(n + \log \ell) \) steps the minimal \( \ell \) for which there exist \( \{i_1, \ldots, i_t\} \subseteq [k] \) such that \( w_{i_1} \cdots w_{i_t} \) is \( k \)-universal. The overall complexity of the algorithm is, thus, \( O(\frac{\log k}{\log n} \log \ell + n) \).
Proof. Let \( u_0 \in \{w_1, \ldots, w_p\} \) be such that \( t_{u_0} \geq t_{w_i} \) for all \( i \in [p] \). For each \( x \in \{a, b\} \), let \( u_x \in \{w_1, \ldots, w_p\} \) be such that \( u_x \) starts with \( x \) and \( t_{u_x, [2..|w_x|]} \geq t_{w_i, [2..|w_i|]} \) for all \( i \in [p] \). For each \( x \in \{a, b\} \), let \( v_x \in \{w_1, \ldots, w_p\} \) be such that \( v_x \) ends with \( x \) and \( t_{v_x, [1..|v_x|]} \geq t_{w_i, [1..|w_i|]} \) for all \( i \in [p] \). For each pair \( x, y \in \{a, b\} \), let \( u_{x,y} \in \{w_1, \ldots, w_p\} \) be such that \( u_{x,y} \) starts with \( x \) and ends with \( y \) and \( t_{u_{x,y}, [2..|u_{x,y}|]} \geq t_{w_i, [2..|w_i|]} \) for all \( i \in [p] \). In case of equalities, we just any word that fulfills the desired property.

Let \( S = \{u_0\} \cup \{u_x \mid x \in \{a, b\}\} \cup \{v_x \mid x \in \{a, b\}\} \cup \{u_{x,y} \mid x, y \in \{a, b\}\} \). Clearly, \( S \) contains at most 9 words. Note that all words of \( S \) can be computed in \( O(n) \) time, using the same strategy as in Proposition 10.

One can show that for every concatenation of \( m \) words from \( \{w_1, \ldots, w_p\} \) which is \( k \)-universal, there exist a concatenation of \( m \) words from \( S \) which is \( k' \)-universal, for some \( k' \geq k \). Thus, it is enough to solve the problem for the input set \( S \), of constant size, instead of the whole \( \{w_1, \ldots, w_p\} \). Hence, by Theorem 28 the conclusion follows.

Indeed, let \( w = w_{i_1} \cdots w_{i_{t-1}} w_{i_t} w_{i_{t+1}} \cdots w_{i_m} \), such that \( w_{i_t} \notin S \). To compute \( t_w = t \) we can proceed as in Proposition 10 and identify \( t \) factors \( d_1, \ldots, d_t \in \{ab, ba\} \) of \( w \) such that \( w = s_0 d_1 s_1 \cdots d_t s_t \), where \( s_i \in \{a, b\}^*_t \) for \( i \in [t]_0 \). Clearly, \( |\text{alph}(s_i)| \leq 1 \), for all \( i \in [t] \). Now, we do a case analysis.

Let \( x = w_{i_1}^1 \) and \( y = w_{i_t}^1 \). If the first letter of \( w_{i_t} \) is the last letter of a factor \( d_i \) and the last letter of \( w_{i_t} \) is the first letter of a factor \( d_j \) (with \( i < j \)), let \( w' = w_{i_1} \cdots w_{i_{t-1}} u_{x,y} w_{i_{t+1}} \cdots w_{i_m} \); it is immediate that \( t_{w'} \geq t_w \). If the first letter of \( w_{i_t} \) is the first letter of some \( d_i \) but the last letter of \( w_{i_t} \) is not the first letter of any factor \( d_j \) (where \( j > i \)), let \( w' = w_{i_1} \cdots w_{i_{t-1}} u_x w_{i_{t+1}} \cdots w_{i_m} \); it is immediate that \( t_{w'} \geq t_w \). If the first letter of \( w_{i_t} \) is not the last letter of any factor \( d_i \) but the last letter of \( w_{i_t} \) is the first letter of a factor \( d_j \), let \( w' = w_{i_1} \cdots w_{i_{t-1}} u_y w_{i_{t+1}} \cdots w_{i_m} \); it is immediate that \( t_{w'} \geq t_w \). Finally, if the first letter of \( w_{i_t} \) is not the last letter of any factor \( d_i \) and the last letter of \( w_{i_t} \) is not the first letter of any factor \( d_j \), let \( w' = w_{i_1} \cdots w_{i_{t-1}} u_0 w_{i_{t+1}} \cdots w_{i_m} \); it is immediate that \( t_{w'} \geq t_w \).

So, if a concatenation of \( m \) words \( w_{i_1} \cdots w_{i_m} \) is \( t \)-universal, we could iteratively replace all the words which are not part of \( S \) by words of \( S \) and obtain a concatenation with \( m \) input words, which is \( t' \)-universal, with \( t' \geq t \). Therefore, to solve the problem from the statement of the theorem is enough to produce the set \( S \) and then solve the problem for the input set \( S \) instead of \( \{w_1, \ldots, w_p\} \). For that we can use the algorithm from Theorem 28, which will run in \( O(n + \log k \log \ell) \) steps, because both \( S \) and \( \Sigma \) are constant.

In a particular case of Theorem 28 each of the input words contain all letters of \( \Sigma \). Once again, we obtain a polynomial algorithm.

**Theorem 30.** Given \( w_1, \ldots, w_p \in \Sigma^* \), with \( \text{alph}(w_i) = \Sigma \) for all \( i \in [p] \) and \( |w_1 \cdots w_p| = n \), and \( k \in \mathbb{N} \), we can compute in polynomial time \( O(n + p^d |\Sigma| \log \log k) \) the minimal \( \ell \) for which there exist \( \{i_1, \ldots, i_k\} \subseteq [k] \) with \( w_{i_1} \cdots w_{i_k} \) is \( k \)-universal.
The proofs of Theorems 28 and 30 are based on a common dynamic pro-
gramming algorithm: for all subsets \( S \subset \Sigma \) and increasing values of an integer \( \ell > 1 \), we compute the maximal universality index \( m \) that we can obtain by concatenating \( 2^\ell \) words from the input set such that the respective concatenation consists in a prefix which is \( m \)-universal, followed by a suffix over \( S \). We stop as soon we reach an \( m \geq k \) as universality index. We then optimise the number of concatenated words needed to obtain universality index \( k \) by binary search. Now, for Theorem 28 we really have to consider all the sets \( S \), in each step, while in the case of Theorem 30 it is enough to consider only the sets that occur as alphabets of the suffixes of the input words. This is why this strategy can be implemented more efficiently in the case when all input words are universal to begin with.

Proof. (of Theorem 30) We follow the idea of the algorithm of Theorem 28: construct, by dynamic programming, longer and longer concatenations of factors of the set \( \{w_1, \ldots, w_p\} \), until one such concatenation which is \( k \)-universal is obtained. The main difference is that in each concatenation \( w = w_{i_1} \cdots w_{i_m} \), the shortest prefix of \( w \) which is \( k \)-universal ends inside \( w_{i_m} \), because \( \text{alph}(w_i) = \Sigma \) for all \( i \in [p] \). As such, the \( \ell \) we search for is at most \( k \), but also this allows us to get rid of the exponential dependency on \( \Sigma \) from Theorem 28, as we can now work with certain suffixes of the words \( w_i \), instead of subsets of \( \Sigma \), when defining our dynamic programming structures. Informally, our algorithm works as follows: we find the highest universality index of a concatenation of \( 2^\ell \) words of \( \{w_1, \ldots, w_p\} \), which starts inside \( w_i \) and ends inside \( w_j \), for all \( i \) and \( j \), and suitable starting and, respectively, ending positions. This can be efficiently computed for several reasons. Firstly, such a concatenation is obtained by putting together two concatenations of roughly \( 2^{\ell-1} \) words of \( \{w_1, \ldots, w_p\} \) which have the highest universality index, the first starting in the same place within \( w_i \), followed by \( 2^{\ell-1} - 2 \) words of the input set, and ending with a prefix of length \( c \) of some \( w_q \), and the second one starting with \( w_q[c+1..|w_q|] \) followed by \( 2^{\ell-1} \) words from the input set, ending in the same place within \( w_j \). Secondly, a concatenation of \( 2^\ell \) words of \( \{w_1, \ldots, w_p\} \) with the highest universality index, ending inside \( w_j \), can actually only end on some very specific positions of \( w_j \): the positions where each letter of \( \Sigma \) occurs for the first time in the shortest prefix of \( w_j \) that contains all letters of \( \Sigma \). Consequently, the starting positions of such concatenations (useful in our algorithm either directly as solutions, or as building blocks for larger concatenations) are also restricted. Putting these two ideas together, and using an adapted binary search to search for \( \ell \), we obtain an algorithm with the stated complexity.

Once again, we start with some preliminaries and a preprocessing phase. Let \( \sigma = |\Sigma| \).

To begin with, let us consider a concatenation \( w = w_{i_1} \cdots w_{i_m} \), and let \( t \) be the maximum number such that \( w \) is \( t \)-universal. By Lemma 10 we can greedily decompose \( w = d_1 \cdots d_{d'} \), such that \( \text{alph}(d_j) = \Sigma \), \( \text{alph}(d') \) is a strict subset of \( \Sigma \), and \( d_1 \cdots d_{d_j} \) is the shortest prefix of \( w \) which is \( j \)-universal, for all \( j \in [t] \). Because \( \text{alph}(w_{i_i}) = \Sigma \) for all \( i \), we have that each factor \( d_j \) is either fully
contained in one of the words $w_{i_g}$, for $j \in [t]$ and $g \in [m]$, or it starts inside $w_{i_g}$ and ends inside $w_{i_{g+1}}$, for some $g \in [m]$. In the following, we call a factor $d_j$ crossing if it starts inside $w_{i_g}$ and ends inside $w_{i_{g+1}}$, for some $g \in [m]$. If $d_j$ is such a factor, then $d_j$ can only start on some very specific positions inside $w_{i_g}$. Firstly, the suffix of $w_{i_g}$ that comes after $d_{j-1}$ cannot contain all letters of $\Sigma$. So $d_{j-1}$ must end inside the shortest suffix of $w_{i_g}$ that contains all letters of $\Sigma$. Assume this suffix starts on position $r$ and note that it starts with the last occurrence of some letter of $\Sigma$ in $w_{i_g}$. So, $d_{j-1}$ ends on a position $r' \geq r$. Due to the greedy construction of $d_{j-1}$, it follows that the last letter of $d_{j-1}$ occurs in $w[r..r']$ exactly once. So, $d_{j-1}$ ends on the first occurrence of a letter of $\Sigma$ to the right of $r$. There are at most $\sigma$ such positions. Consequently, $d_j$ starts exactly on the next position after $d_{j-1}$ ends, and we also have at most $\sigma$ positions where $d_j$ may start.

In conclusion, in every word $w_i$, part of a concatenation $w = w_{i_1} \ldots w_{i_n}$, there are at most $\Sigma$ positions where a crossing factor can start. Each crossing factor $d_i$ is constructed by appending to $d_j$ (in a left to right traversal, from the starting position of the factor) the letters of the considered concatenation until $\Sigma = \text{alph}(d_j)$. Therefore, $d_i$ is uniquely determined by the two factors it crosses ($w_{i_g}$ and $w_{i_{g+1}}$) and its starting position inside $w_{i_g}$. Hence, there can be at most $O(p^2|\Sigma|)$ crossing factors overall, and we will determine all of them in our preprocessing.

In the preprocessing phase, we first construct the $p \times |\Sigma|$ matrix $F[\cdot, \cdot]$, indexed by the numbers between 1 and $p$ and the letters of $\Sigma$ (which are numbers between 1 and $|\Sigma|$). We have $F[i, a]$ is the position of the first (i.e., leftmost) occurrence of each letter $x \in \Sigma$ in $w_i$. This matrix can be computed as follows. Initialise all elements of $F$ with 0. For each $i$, we traverse $w_i$, letter by letter, from left to right. When the letter $x \in \Sigma$ is read on position $j$ of $w_i$, if $F[i, z] = 0$ the we set $F[i, x] = j$. The total number of steps needed to do this is $O(|\Sigma|p + n)$ (as it includes the initialisation of $F$). Similarly, we construct the $p \times |\Sigma|$ matrix $L[\cdot, \cdot]$, indexed by the numbers between 1 and $p$ and the letters of $\Sigma$, where $L[i, a]$ is the position of the rightmost occurrence of each letter $x \in \Sigma$ in $w_i$. Using $L[i, \cdot]$ we also determine the position $r_i$ of $w_i$ such that $w_i[r_i..|w_i|]$ is the shortest suffix of $w_i$ that contains all letters of $\Sigma$. Also, in another traversal of $w_i$ we can compute the increasingly sorted list $L_i$ of positions where each letter of $\Sigma$ occurs for the first time in $w_i[r_i..|w_i|]$. More precisely, we construct the lists $L_i = (i_1, x_1), \ldots, (i_\sigma, x_\sigma)$, where $i_g < i_{g+1}$ for $g \in [\sigma]$, and $\{x_1, \ldots, x_\sigma\} = \Sigma$. The needed to compute all these structures is $O(n)$.

Now, we compute the factors crossing from $w_i$ to $w_j$. They should start on one of the positions $i_1 + 1, i_2 + 1, \ldots, i_\sigma + 1$, obtained using $L_i$. Let $c_{i,j} [t_g + 1]$ be the crossing factor that starts on position $i_g + 1$ for some $g \in [\sigma]$. The prefix of $c_{i,j} [t_g + 1]$ contained in $w_i$ contains only the letters $x_{g+1}, \ldots, x_\sigma$ and none of the letters $x_1, \ldots, x_g$. Thus, $c_{i,j} [t_g + 1]$ extends in $w_j$ until it contains all the missing letters, i.e., till the maximum position among $F[j, x_1], F[j, x_2], \ldots, F[j, x_g]$. This observation allows us to compute the respective crossing factors efficiently. Let
\(C[i, j, g]\) be the last position (inside \(w_j\)) of \(c_{i,j}[i_g + 1]\). Then \(C[i, j, 1] = F[j, x_1]\). For \(g > 1\), \(C[i, j, g] = \max\{F[j, x_p], C[i, j, g-1]\}\).

The time needed to compute the values \(C[i, j, \cdot]\) is \(O(\sigma)\). We do this computation for all \(i\) and \(j\), and, as such, we identify the starting and ending positions for all possible crossing factors in \(O(p^2\sigma)\).

Still in the preprocessing phase, we compute the data structures from Lemma 24 for each word \(w_i\), with \(i \in [p]\). So, we have for each word \(w_i\) the arrays \(t_{w_i}[j] = \max\{t | \text{ScatFact}(w_i[j..n]) = \Sigma^t\}\) and \(m_{w_i}[j] = \min\{g | \text{ScatFact}_{w_i}[j](w_i[j..g]) = \Sigma^{t_{w_i}[j]}\}\). Let \(t_{w_i} = t_{w_i}[1]\) and \(m_{w_i} = m_{w_i}[1]\). This takes \(O(n)\) time.

Further, we present the main phase of our algorithm, that computes the value \(\ell\) for which there exist \(\{i_1, \ldots, i_k\} \subseteq [k]\) such that \(w_{i_1} \cdots w_{i_k}\) is \(k\)-universal.

Firstly, if there exists \(i\) such that \(t_{w_i} \geq k\), we have \(\ell = 1\). Otherwise, we continue as follows.

For \(e \in [k], e \geq 1\), we define the 3-dimensional arrays \(R_e[\cdot, \cdot, \cdot]\), whose first and third indices are numbers \(i, j \in [p]\), and second index is a number from \([0] \cup L_i\) (so each \(R_e\) has size \(O(p^2\sigma)\)). We define \(R_e[i, j, c] = (t, d)\) where \(t\) is the maximum number for which there exist \(i_2, \ldots, i_{2^e - 1} \in [p]\) such that \(t_w = t\), where \(w = w_i[c+1..w_i]\|w_{i_2} \cdots w_{i_{2^e-1}} w_j\) and \(d\) is the minimum number for which there exist \(i_2, \ldots, i_{2^e-1}\) such that \(t_w = t\), where \(w = w_i[c+1..w_i]\|w_{i_2} \cdots w_{i_{2^e-1}} w_j[1..d]\) if \(t_w = t\) (i.e., \(t_w\) is as large as possible). We define also the 3-dimensional arrays \(P_e[\cdot, \cdot, \cdot]\), exactly as the above with the single difference that in the definition of the elements of \(P_e\) we consider the concatenation of \(2^e + 1\) elements, not just \(2^e\) as it was the case for \(R_e\).

The elements \(R_e[i, j, c]\) and \(P_e[i, j, c]\) can be computed by dynamic programming, somehow similarly to the approach of Theorem 28. To simplify the exposure we also define the array \(R_0[i, \cdot, \cdot]\), in which only the elements \(R_0[i, i, c-1] = (t_{w_i}[c], m_{w_i}[c])\) are defined (the others are set to \(-\infty\)). Clearly, \(R_0\) can be computed in \(O(p^2\sigma)\).

To describe the general computation, we need to compare pairs of numbers. We say that \((a, b)\) is more useful than \((c, d)\) if \(a > b\) or \(a = b\) and \(c \leq d\). Also, if \(p = (a, b)\) is a pair of natural numbers, then its first projection is \(\pi_1(p) = a\) and its second projection is \(\pi_2(p) = b\).

To compute \(R_1\) we can use the formula:
\[
R_1[i, j, c-1] = (t_{w_i}[c] + 1 + \pi_1(R_0[j, j, C_{i,j}[m_{w_i}[c] + 1]]), m_{w_i}[1 + C_{i,j}[m_{w_i}[c] + 1]],
\]
for \(i, j \in [p]\) and \(c \in \{0\} \cup L_i\).

Indeed, when computing \(R[i, j, c-1]\) we start on position \(c\) of \(w_i\) and essentially try to identify as many consecutive strings whose alphabet is \(\Sigma\) in the concatenation of \(w_i\) and \(w_j\) as possible. Firstly, using \(t_{w_i}[c]\) and \(m_{w_i}[c]\) we find the shortest factor \(w_i[c..m_{w_i}[c]]\) which has the highest universality index among
all factors of $w_i$ starting on position $c$. Then we use the crossing factor that corresponds to $m_{w_i}[c]$ to move into $w_j$, on position $c' = C_{i,j}[m_{w_i}[c] + 1]$, and then find the shortest factor $w_j[c',m_{w_j}[c']]$ which has the highest universality index among all factors of $w_j$ starting on position $c'$. Following similar arguments to those in the proof of Lemma 10 we get that $R_0$ is correctly computed in this way: our strategy here corresponds exactly to the greedy strategy employed in the respective lemma.

After we compute $R_e$, for some $e \geq 1$, we first compute $P_e$. The formula for the elements of $P_e$ is given in the following. Let $q \in [p]$ be such that $R_e[q,j,1+m_{w_q}[1+C_{i,q}[m_{w_i}[c] + 1]])$ is more useful than any other pair $R_e[q',j,1+m_{w_q}[1+C_{i,q'}[m_{w_i}[c] + 1]])$ for $q' \in [p]$. We then can compute

$$P_e[i,j,c-1] = (t_{w_i}[c] + 1 + t_{w_q}[1+C_{i,q}[m_{w_i}[c] + 1]]) + \pi_1(R_e[q,j,m_{w_q}[1+C_{i,q}[m_{w_i}[c] + 1]]),$$

for $i,j \in [p]$ and $c \in \{0\} \cup L_i$. Similarly to the computation of $R_1$, when computing $P_e[i,j,c-1]$ we start on position $c$ of $w_i$ and try to add to $w_i[c..|w_i|]$ a concatenation of $2^e$ words of $\{w_1,\ldots,w_p\}$ (ending with $w_j$), which contains as many consecutive strings, whose alphabet is $\Sigma$, as possible. This is done using

---

**Fig. 1.** The computation of $R_1[i,j,c-1]$

**Fig. 2.** The computation of $P_e[i,j,c-1]$
the greedy approach of Lemma [10]. As such, we use \( t_{w_i}[c] \) and \( m_{w_i}[c] \) we find the shortest factor \( w_i[c..m_{w_i}[c]] \) which has the highest universality index among all factors of \( w_i \) starting on position \( c \). Then we identify the word \( w_q \) such that after using the crossing factor that corresponds to \( m_{w_i}[c] \) to move from \( w_i \) into \( w_q \) we can reach \( w_j \) by concatenating another \( 2^e - 2 \) factors in between, to obtain a word with the highest universality index among all such possible concatenations. Once again, it is not hard to see that this formula is correct (see also the figure below). Firstly, the choice of the factor \( w_i[c..m_{w_i}[c]] \) as the first group of consecutive strings, each with the alphabet \( \Sigma \), is correct due to the greedy approach in Lemma [10]. Then, we need to cross into the rest of the factors in the concatenation of words from \( \{w_1, \ldots, w_p\} \). For each choice \( w_q' \) of the factor following \( w_i \) in this concatenation, we cross into this word from \( w_i \) in an optimal way: we use the crossing string ending on \( C_{i,q'}[m_{w_i}[c] + 1] \). Any shorter word would not work, any longer word does not make sense due to the greedy strategy of Lemma [10]. Then, using the already computed \( m_{w_i} \) and \( R_e \) we start from \( 1 + C_{i,q'}[m_{w_i}[c] + 1] \) and follow the optimal selection of the concatenated strings given by these arrays. We then select from all these possibilities (computed for each \( q' \)) the one that produces a string with higher universality index. So, the computation of \( P_e[i, j, c - 1] \) is correct.

After computing \( P_e \) for some \( e \geq 1 \), we compute \( R_{e+1} \). For some \( i, j \in [p] \) and \( c \in \{0\} \cup L_i \), let \( q \in [p] \) be such that \( \pi_1(R_e[i, q, c]) = \pi_1(P_e[q, j, \pi_2(R_e[i, q, c])]) \)
\( \pi_1(R_e[i, q', c]) = \pi_1(P_e[q', j, \pi_2(R_e[i, q', c])]) \) for all \( q' \in [p] \). To break equalities, we select \( q \) such that \( \pi_2(P_e[q, j, \pi_2(R_e[i, q, c])]) \) is minimal. Then, we can compute \( R_{e+1}[i, j, c - 1] \) by the following formula:

\[
R_{e+1}[i, j, c - 1] = (\pi_1(R_e[i, q, c - 1]) + \pi_1(P_e[q, j, \pi_2(R_e[i, q, c - 1])]),
\pi_2(P_e[q, j, \pi_2(R_e[i, q, c - 1])]))
\]

for \( i, j \in [p] \) and \( c \in \{0\} \cup L_i \).

The idea is pretty similar to how we computed the other arrays. We start on position \( c + 1 \) of \( w_i \) and try to add to \( w_i[c+1..w_i] \) a concatenation of \( 2^{e+1} - 1 \) words of \( \{w_1, \ldots, w_p\} \) (ending with \( w_j \)), which contains as many consecutive strings, whose alphabet is \( \Sigma \), as possible. We iterate over all possible choices for the \( 2^e \)-th word in this concatenation, namely \( w_q \). We use the value computed found in \( R_e[i, q, c] \) to find the concatenation of \( 2^e \) words with highest

![Fig. 3. The computation of \( R_{e+1}[i, j, c - 1] \)](image-url)
universality index that starts with \(w_i[c..w_i]\) and ends with \(w_q\). Then we continue this concatenation again in the best way (i.e., by the concatenation of \(2^{e+1}\) words with the highest universality index), as given by \(P_e[q', j, \pi_2(R_e[i, q', c])].\) Then we just take the value \(q\) for which we obtained the most useful pair \((\pi_1(R_e[i, q, c]) + \pi_1(P_e[q, j, \pi_2(R_e[i, q, c]))), \pi_2(P_e[q, j, \pi_2(R_e[i, q, c]))).\) Once more, the greedy approach shown to be correct in Lemma \(10\) proves that the formula used for the elements of \(R_{e+1}\) is also correct.

Clearly, the complexity of computing each element of \(P_e\) and \(R_e\) is \(O(p)\). So, computing each of these matrices takes \(O(p^3\sigma)\).

As in the proof of Theorem \(28\) we stop as soon as we computed an array \(R_f\) that contains an element \(R_f[i, j, 0]\) with \(\pi_1(R_f[i, j, 0]) \geq k\). We have \(f \in O(\log \ell)\).

Now we need to search \(\ell\) between \(b = 2^{\ell-1}\) and \(s = 2^\ell\). And we can proceed exactly as in the proof of the aforementioned theorem, by an adapted binary search. Denote \(R' = R_{f-1}\) and \(R'' = R_f\). Let \(h\) be maximum such that \(b+2^h < s\).

We compute the 3-dimensional array \(R_{mid}[i, j, c - 1]\), indexed just as the arrays \(R_e\).

We have
\[
R_{mid}[i, j, c - 1] = (\pi_1(R'[i, q, c - 1]) + \pi_1(P_h[q, j, \pi_2(R'[i, q, c - 1]))), \\
\pi_2(P_h[q, j, \pi_2(R'[i, q, c - 1]))),
\]
for \(i, j \in [p]\) and \(c \in \{0\} \cup L_i\).

If \(R_{mid}\) contains an element whose first component is greater or equal to \(\ell\), we repeat this search for the same \(b\) and \(R'\), and \(s = b + 2^h\) and \(R'' = R_{mid}\).

Otherwise, we repeat the search for the same \(s\) and \(R''\), and using \(b + 2^h\) instead of \(b\) and \(R_{mid}\) instead of \(R'\). We stop the process if \(b = s - 1\), and return \(s\). This procedure is iterated \(O(f) = O(\log \ell)\) times.

The overall number of steps of the algorithm we described is, thus, \(O(p^3\sigma \log \ell + n)\). Of course, in the part where we compute concatenations with large universality index we need to manage arithmetic operations with \(\log k\)-bit numbers. So, our algorithm requires \(O(p^3\sigma \log \ell \log k + n)\) time.

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 31.** 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 \(\ell\).

**Proof.** To compute the longest prefix \(w[1..i-1]\) of \(w\) which has universality index \(\ell\), we use data structures from Lemma \(24\). We start with \(j = 1\) and \(k = 0\). While \(k \neq \ell + 1\) do \(t = u_w[j]\), increase \(k\), set \(j = j + 1\). If \(k = \ell + 1\) then \(w[1..t]\) is the shortest prefix of \(w\) which is \(\ell + 1\) universal. Therefore the longest prefix \(w[1..i-1]\) of \(w\) which has universality index \(\ell\) is \(w[1..t-1]\). A similar approach can be used for suffixes.

**Theorem 31** allows us to compute which is the shortest prefix (suffix) we should delete so that we get a string of universality index \(\ell\). 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 \(\ell\), 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.
6 Conclusion

In this paper we investigated the scattered factor universality of words. We have proven how this universality behaves if a word is repeated and how this characterisation can be exploited to obtain linear-time algorithms for obtaining an uncommon scattered factor. Moreover we set the universality of a palindrome into relation with its first half (minus one letter if the length is odd) as well as the generalised repetition $w\pi(w)$ for a morphic permutation $\pi$. The last part of Section 4 dealt with circular universality. Here we have proven the relation between universality and circular universality and we have proven that the characterisation in Theorem 23 does not hold for arbitrary alphabets. We conjecture that for an alphabet of cardinality $\sigma$ the notion of circularity has to be generalised such that, assuming the word as a circle, not once but $\sigma-1$ times the word has to read before the universality is increased. Finally in the last section we developed data structures that allow us to determine the universality of factors of a given word.

References

1. Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In Proc. SODA 2018, pages 1216–1235. SIAM, 2018.
2. Herman Z. Q. Chen, Sergey Kitaev, Torsten Mütze, and Brian Y. Sun. On universal partial words. Electronic Notes in Discrete Mathematics, 61:231–237, 2017.
3. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, 3rd Edition. MIT Press, 2009.
4. Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. Algorithms on strings. Cambridge University Press, 2007.
5. Joel D. Day, Pamela Fleischmann, Florin Manea, and Dirk Nowotka. $k$-spectra of weakly-$c$-balanced words. In Proc. DLT 2019, volume 11647 of Lecture Notes in Computer Science, pages 265–277. Springer, 2019.
6. Nicolaas G. de Bruijn. A combinatorial problem. Koninklijke Nederlandse Akademie v. Wetenschappen, 49:758–764, 1946.
7. Aldo de Luca, Amy Glen, and Luca Q. Zamboni. Rich, sturmian, and trapezoidal words. Theor. Comput. Sci., 407(1-3):569–573, 2008.
8. David P. Dobkin and Richard J. Lipton. On the complexity of computations under varying sets of primitives. J. Comput. Syst. Sci., 18(1):86–91, 1979.
9. Xavier Droubay, Jacques Justin, and Giuseppe Pirillo. Episturmian words and some constructions of de Luca and Rauzy. Theor. Comput. Sci., 255(1-2):539–553, 2001.
10. Cees H. Elzinga, Sven Rahmann, and Hui Wang. Algorithms for subsequence combinatorics. Theor. Comput. Sci., 409(3):394–404, 2008.
11. Lukas Fleischer and Manfred Kufleitner. Testing Simon’s congruence. In Proc. MFCS 2018, volume 117 of LIPIcs, pages 62:1–62:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
12. Dominik D. Freydenberger, Pawel Gawrychowski, Juhani Karhumäki, Florin Manea, and Wojciech Rytter. Testing $k$-binomial equivalence. CoRR, abs/1509.00622, 2015.
13. Harold N. Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of disjoint set union. In Proc. 15th STOC, pages 246–251, 1983.

14. Pawel Gawrychowski, Narad Rampersad, Jeffrey Shallit, and Marek Szykula. Existential length universality. to appear at STACS, abs/1702.03961, 2020.

15. Bennet Goeckner, Corbin Groothuis, Cyrus Hettle, Brian Kell, Pamela Kirkpatrick, Rachel Kirsch, and Ryan W. Solava. Universal partial words over non-binary alphabets. Theor. Comput. Sci, 713:56–65, 2018.

16. Simon Halfon, Philippe Schnoebelen, and Georg Zetzsche. Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In Proc. LICS 2017, pages 1–12, 2017.

17. Jean-Jacques Hebrard. An algorithm for distinguishing efficiently bit-strings by their subsequences. Theoretical Computer Science, 82(1):35–49, 22 May 1991.

18. Markus Holzer and Martin Kutrib. Descriptive and computational complexity of finite automata - A survey. Inf. Comput., 209(3):456–470, 2011.

19. Hiroshi Imai and Takao Asano. Dynamic segment intersection search with applications. In Proc. 25th Annual Symposium on Foundations of Computer Science, FOCS, pages 393–402. IEEE Computer Society, 1984.

20. Prateek Karandikar, Manfred Kufleitner, and Philippe Schnoebelen. On the index of Simon’s congruence for piecewise testability. Inf. Process. Lett., 115(4):515–519, 2015.

21. Prateek Karandikar and Philippe Schnoebelen. The height of piecewise-testable languages with applications in logical complexity. In Proc. CSL 2016, volume 62 of LIPIcs, pages 37:1–37:22, 2016.

22. Prateek Karandikar and Philippe Schnoebelen. The height of piecewise-testable languages and the complexity of the logic of subwords. Logical Methods in Computer Science, 15(2), 2019.

23. Markus Krötzsch, Tomáš Masopust, and Michaël Thomazo. Complexity of universality and related problems for partially ordered NFAs. Inf. Comput., 255:177–192, 2017.

24. Dietrich Kuske and Georg Zetzsche. Languages ordered by the subword order. In Proc. FOSSACS 2019, volume 11425 of Lecture Notes in Computer Science, pages 348–364. Springer, 2019.

25. Marie Lejeune, Julien Leroy, and Michel Rigo. Computing the k-binomial complexity of the Thue-Morse word. CoRR, abs/1812.07330, 2018.

26. Julien Leroy, Michel Rigo, and Manon Stipulanti. Generalized Pascal triangle for binomial coefficients of words. CoRR, abs/1705.08270, 2017.

27. M. Lothaire. Combinatorics on Words. Cambridge University Press, 1997.

28. David Maïer. The complexity of some problems on subsequences and supersequences. J. ACM, 25(2):322–336, April 1978.

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

30. Alexandru Mateescu, Arto Salomaa, and Sheng Yu. Subword histories and Parikh matrices. Journal of Computer and System Sciences, 68(1):1–21, 2004.

31. Narad Rampersad, Jeffrey Shallit, and Zhi Xu. The computational complexity of universality problems for prefixes, suffixes, factors, and subwords of regular languages. Fundam. Inf., 116(1-4):223–236, January 2012.

32. Michel Rigo and Pavel Salimov. Another generalization of abelian equivalence: Binomial complexity of infinite words. Theor. Comput. Sci., 601:47–57, 2015.

33. Arto Salomaa. Connections between subwords and certain matrix mappings. Theoretical Computer Science, 340(2):188–203, 2005.
34. Shinnosuke Seki. Absoluteness of subword inequality is undecidable. *Theor. Comput. Sci.*, 418:116–120, 2012.
35. Imre Simon. Piecewise testable events. In *Autom. Theor. Form. Lang., 2nd GI Conf.*, volume 33 of *LNCS*, pages 214–222. Springer, 1975.
36. Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, January 1974.
37. Georg Zetzsche. The complexity of downward closure comparisons. In *Proc. ICALP 2016*, volume 55 of *LIPIcs*, pages 123:1–123:14, 2016.