On Arch Factorization and Subword Universality for Words and Compressed Words

Philippe Schnoebelen, Julien Veron

On arch factorization and subword universality for words and compressed words

Ph. Schnoebelen and J. Veron

LMF, CNRS & ENS Paris-Saclay, France *

Abstract. Using arch-jumping functions and properties of the arch factorization of words, we propose a new algorithm for computing the subword circular universality index of words. We also introduce the subword universality signature for words, that leads to simple algorithms for the universality indexes of SLP-compressed words.

1 Introduction

A *subword* of a given word is obtained by removing some letters at arbitrary places. For example, `abba` is a subword of abracadabra, as witnessed by the underlined letters. Subwords are a fundamental notion in formal language theory and in algorithmics but they are not as well-behaved as *factors*, a special case of subwords where the kept letters correspond to an interval inside the original word.[1]

Words and languages can be characterised or compared via their subwords. For example, we can distinguish $u_1 =$ `nationalists` from $u_2 =$ `antinationalists` by the subword $x =$ `ino`. Indeed, only u_2 has x as a subword. We say that x is a *distinguisher* (also, a *separator*) between u_1 and u_2. Observe that `ino` is a *shortest* distinguisher between the two words.[2] In applications one may want to distinguish between two similar DNA strings, or two traces of some program execution: in these situations where inputs can be huge, finding a short distinguishing subword requires efficient algorithms [Sim03]. When considering the usual first-order logic of words (i.e., labelled linear orders), a distinguisher x can be seen as a Σ_1 formula separating the two words.

Definability by subwords. These considerations led Imre Simon to the introduction of *piecewise-testable* languages in his 1972 Phd thesis [Sim72,Sim75]: these languages can be defined entirely in terms of forbidden and required subwords.

* Work partially supported by Labex DigiCosme (project ANR-11-LABEX-0045-DIGICOSME) operated by ANR as part of the program « Investissement d'Avenir » Idex Paris-Saclay (ANR-11-IDEX-0003-02).

[1] Some papers use the terminology "subwords" for factors, and "scattered subwords" or "scattered factors" for subwords. We follow [SS83].

[2] This is a very rare situation with the English lexicon, where different words almost always admit a length-2 distinguisher. To begin with, two words can already admit a length-1 distinguisher unless they use exactly the same set of letters.

In logical terms, this corresponds to $\mathcal{B}\Sigma_1$-definability, see [DGK08]. Piecewise testability is an important and fundamental concept, and it has been extended to, among others, trees [BSS12,GS16], picture languages [Mat98], or words over arbitrary scattered linear orderings [CP18].

From a descriptive complexity point of view, a relevant measure is the *length of subwords* used in defining piecewise-testable languages, or in distinguishing between two individual words. Equivalently, the required length for these subwords is the required number of variables for the $\mathcal{B}\Sigma_1$ formula. This measure was investigated in [KS19] where it is an important new tool for bounding the complexity of decidable logic fragments.

Subword universality. Barker, Day *et al.* introduced the notion of subword universality: a word u is *k-universal* if all words of length at most k are subwords of u [BFH$^+$20,DFK$^+$21]. They further define the *subword universality index* $\iota(u)$ as the largest k such that u is k-universal. Their motivations come, among others, from works in reconstructing words from subwords [DPFD19] or computing edit distance [DFK$^+$21], see also the survey in [KKMS22]. In [BFH$^+$20], the authors prove several properties of $\iota(u)$, e.g., when u is a palindrome, and further introduce the *circular* subword universality index $\zeta(u)$, which is defined as the largest $\iota(u')$ for u' a conjugate of u. Alternatively, $\zeta(u)$ can be seen as the subword universality index $\iota([u]_\sim)$ for a *circular word* (also called necklace, or cyclic word), i.e., an equivalence class of words modulo conjugacy.

While it is easy to compute $\iota(u)$, computing $\zeta(u)$ is trickier but [BFH$^+$20] proves several bounds relating $\zeta(u)$ to the values of $\iota(u^n)$ for $n \in \mathbb{N}$. This is leveraged in [FGN21] where an $O(|u| \cdot |A|)$ algorithm computing $\zeta(u)$ is given. That algorithm is quite indirect, with a delicate and nontrivial correctness proof. Further related works are [KKMS21] where, given that $\iota(u) = k$, one is interested in all the words of length $k+1$ that do not occur as subwords of u, [FHH$^+$22] where one considers words that are just a few subwords away from k-universality, and [KKMP22] where the question whether u has a k-universal factor of given length is shown to be NP-complete.

Our contribution. In this paper we introduce new tools for studying subword (circular) universality. First we focus on the arch factorizations (introduced by Hébrard [Héb91]) and show how *arch jumping* functions lead to simple proofs of combinatorial results on subword universality indexes, allowing a new and elegant algorithm for computing $\zeta(u)$. These arch-jumping functions are implicit in some published constructions and proofs (e.g., in [FK18,FGN21,KKMS21]) but studying them explicitly brings simplifications and improved clarity.

In a second part we give bilinear-time algorithms that compute the universality indexes ι and ζ for compressed words. This is done by introducing a compact *subword universality signature* that can be computed compositionally. These algorithms and the underlying ideas can be useful in the situations we mentioned earlier since long DNA strings or program execution traces are usually very repetitive, so that handling them in compressed form can entail huge savings in both memory and communication time.

More generally this is part of a research program on algorithms and logics for computing and reasoning about subwords [KS15,HSZ17,KS19,GLHK+20]. In that area, handling words in compressed form raises additional difficulties. For example it is not known whether one can compute efficiently the length of a shortest distinguisher between two compressed words. Let us recall here that reasoning on subwords is usually harder than reasoning on factors, and this is indeed true for compressed words: While deciding whether a compressed X is a *factor* of a compressed Y is polynomial-time, deciding whether X is a *subword* of Y is intractable (in PSPACE and PP-hard, see [Loh12, Sect. 8]). However, in the special case where one among X or Y is a *power word*, i.e., a compressed word with restricted nesting of concatenation and exponentiation, the subword relation is polynomial-time, a result crucial for the algorithms in [Sch21] where one handles exponentially long program executions in compressed forms.

Outline of the paper. Section 2 recalls all the necessary definitions for subwords and universality indexes. Section 3 introduces the arch-jumping functions, relates them to universality indexes and proves some basic combinatorial results. Then Section 4 provides a simple algorithm for the circular universality index. In Section 5 we introduce the subword universality signature of words and show how they can be computed compositionally. Finally Section 6 considers SLP-compressed words and their subword universality indexes.

2 Basic notions

Words and subwords. Let $A = \{\mathtt{a}, \mathtt{b}, \ldots\}$ be a finite alphabet. We write u, v, w, s, t, x, $y \ldots$ for words in A^*. Concatenation is denoted multiplicatively while ε denotes the empty word. When $u = u_1u_2u_3$ we say that u_1 is a *prefix*, u_2 is a *factor*, and u_3 is a *suffix*, of u. When $u = vw$ we may write $v^{-1}u$ to denote w, the suffix of u one obtains after removing its v prefix. When $u = v_0w_1v_1w_2 \cdots w_nv_n$, the concatenation $w_1w_2 \cdots w_n$ is a *subword* of u, i.e., a subsequence obtained from u by removing some of its letters (possibly none, possibly all). We write $u \preccurlyeq v$ when u is a subword of v.

A word $u = a_1 \cdots a_\ell$ has length ℓ, written $|u| = \ell$, and we let $A(u) \stackrel{\text{def}}{=} \{a_1, \ldots, a_\ell\}$ denote its alphabet, a subset of A. We let $\mathit{Cuts}(u) = \{0, 1, \ldots, \ell\} \subseteq \mathbb{N}$ denote the set of *cutting positions inside* u, i.e., positions between u's letters, where u can be split: for $0 \leq i \leq j \leq \ell$, we let $u(i, j)$ denote the factor $a_{i+1}a_{i+2} \cdots a_j$. With this notation, $u(0, j)$ is u's prefix of length j, and $u(i, \ell)$ is the suffix $(u(0, i))^{-1}u$. Note also that $u(i, i) = \varepsilon$ and $u(i, j) = u(i, k)\, u(k, j)$ whenever the factors are defined. If $u = u_1u_2$, we say that u_2u_1 is a *conjugate* of u. For $i \in \mathit{Cuts}(u)$, the i-th conjugate of u is $u(i, \ell)\, u(0, i)$ and is denoted by $u^{\sim i}$. Finally $u^{\mathrm{R}} \stackrel{\text{def}}{=} a_\ell \cdots a_1$ denotes the *mirror* of u.

Rich words and arch factorizations. A word $u \in A^*$ is *rich* if it contains at least one occurrence of each letter $a \in A$, otherwise we say that it is *incomplete*. A rich word having no rich strict prefix is an *arch*. The mirror of an arch is

called a *co-arch* (it is generally not an arch). Observe that an arch (or a co-arch) necessarily ends (respectively, starts) with a letter that occurs only once in it.

The *arch factorization* of u, introduced by Hebrard [Héb91], is a decomposition $u = s_1 \cdots s_m \cdot r$ of u into $m+1$ factors given by the following:
— if u is not rich then $m = 0$ and $r = u$,
— otherwise let s_1 be the shortest prefix of u that is rich (it is an arch) and let $s_2, \ldots, s_m, r$ be the arch factorization of the suffix $(s_1)^{-1}u$.

We write $r(u)$ for the last factor in u's factorization, called the *rest* of u. For example, with $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, the arch factorization of $u_{\text{ex}} = \mathtt{baccabbcbaabacba}$ is $\mathtt{bac} \cdot \mathtt{cab} \cdot \mathtt{bcba} \cdot \mathtt{abac} \cdot \mathtt{ba}$, with $m = 4$ and $r(u_{\text{ex}}) = \mathtt{ba}$. Thus the arch factorization is a leftmost decomposition of u into arches, with a final rest $r(u)$.

There is a symmetric notion of co-arch factorization where one factors u as $u = r' \cdot s'_1 \cdots s'_m$ such that r' is incomplete and every s'_i is a co-arch, i.e., a rich factor whose first letter occurs only once.

All the above notions assume a given underlying alphabet A, and we should speak more precisely of "A-rich" words, "A-arches", or "rest $r_A(u)$". When A is understood, we retain the simpler terminology and notation.

Subword universality. In [BFH+20], Barker et al. define the *subword universality index* of a word u, denoted $\iota_A(u)$, or just $\iota(u)$, as the largest $m \in \mathbb{N}$ such that any word of length m in A^* is a subword of u.

It is clear that $\iota(u) = m$ iff the arch factorization of u has m arches. Hence one can compute $\iota(u)$ in linear time simply by scanning u from left to right, keeping track of letter appearances in consecutive arches, and counting the arches [BFH+20, Prop. 10]. Using that scanning algorithm for ι, one sees that the following equalities hold for all words u, v:

$$\iota(u\,v) = \iota(u) + \iota\big(r(u)v\big)\,, \qquad\qquad r(u\,v) = r\big(r(u)v\big)\,. \tag{1}$$

Barker et al. further define the *circular subword universality index* of u, denoted $\zeta(u)$, as the largest $\iota(u')$ for u' a conjugate of u. Obviously, one always has $\zeta(u) \geq \iota(u)$. Note that $\zeta(u)$ can be strictly larger that $\iota(u)$, e.g., with $A = \{\mathtt{a}, \mathtt{b}\}$ and $u = \mathtt{aabb}$ one has $\iota(u) = 1$ and $\zeta(u) = 2$. These descriptive complexity measures are invariant under mirroring of words, i.e., $\iota(u^{\mathrm{R}}) = \iota(u)$ and $\zeta(u^{\mathrm{R}}) = \zeta(u)$, and monotonic w.r.t. the subword ordering:

$$u \preccurlyeq v \implies \iota(u) \leq \iota(v) \wedge \zeta(u) \leq \zeta(v)\,. \tag{2}$$

The behaviour of ζ can be deceptive. For example, while ι is superadditive, i.e., $\iota(uv) \geq \iota(u) + \iota(v)$ —just combine eqs. (1) and (2)— we observe that $\zeta(uv) < \zeta(u) + \zeta(v)$ can happen, e.g., with $u = \mathtt{ab}$ and $v = \mathtt{bbaa}$.

3 Arch-jumping functions and universality indexes

Let us fix a word $w = a_1 a_2 \cdots a_L$ of length L. We now introduce the α and β *arch-jumping functions* that describe the reading of an arch starting from some

position inside w. For $i \in \mathit{Cuts}(w)$, we let

$$\alpha(i) = \min\{j \mid A\big(w(i,j)\big) = A\}, \qquad \beta(j) = \max\{i \mid A\big(w(i,j)\big) = A\}.$$

These are partial functions: $\alpha(i)$ and $\beta(j)$ are undefined when $w(i, L)$ or, respectively, $w(0, j)$, does not contain all the letters from A. See Figure 1 for an illustration.

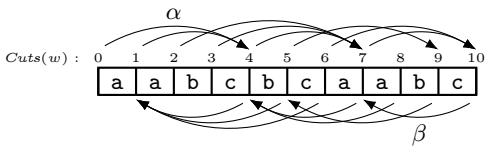


Fig. 1. Arch-jumping functions α, β for $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and $w = \mathtt{aabcbcaabc}$.

The following properties are easily seen to hold for all $i, j \in \mathit{dom}(\alpha)$:

$$\alpha(i) \geq i + |A|\,, \qquad i \leq j \implies \alpha(i) \leq \alpha(j)\,, \tag{3}$$

$$\beta(\alpha(i)) \geq i\,, \qquad \alpha(\beta(\alpha(i))) = \alpha(i)\,. \tag{4}$$

Since β is a mirror version of α, it enjoys similar properties that we won't spell out here.

Remark 3.1. As will be seen in the rest of this section, the arch jumping functions are a natural and convenient tool for reasoning about arch factorizations. Similar concepts can certainly be found in the literature. Already in [Héb91], Hébrard writes $p(n)$ for what we write $\alpha^n(0)$, i.e., the n-times iteration $\alpha(\alpha(\cdots(\alpha(0))\cdots))$ of α on 0: the starting point for the $p(n)$'s is fixed, not variable. In [FK18], Fleischer and Kufleitner use rankers like X_a and Y_b to jump from a current position in a word to the next (or previous) occurrence of a given letter, here a and b: this can specialise to our α and β if one knows what is the last letter of the upcoming arch. In [KKMS21] `minArch` corresponds exactly to our α, but there `minArch` is a data structure used to store information, not a notational tool for reasoning algebraically about arches.

3.1 Subword universality index via jumping functions

The connection between the jumping function α and the subword universality index $\iota(w)$ is clear:

$$\iota(w) = \max\big\{n \;\big|\; \alpha^n(0) \text{ is defined}\big\}\,. \tag{5}$$

For example, w in Figure 1 has $\alpha^3(0) = 10 = |w|$ so $\iota(w) = 3$.

We can generalise Equation (5): $\iota(w) = n$ implies $\alpha^p(0) \leq \beta^{n-p}(|w|)$ for all $p = 0, \ldots, n$, and the reciprocal holds. We can use this to prove the following:

Proposition 3.2. $\iota(u\,v) \leq \iota(u) + \iota(v) + 1$.

Proof. Write n and n' for $\iota(u)$ and $\iota(v)$. Thus, on $w = u\,v$ with $L = |u| + |v|$, one has $\alpha^{n+1}(0) > |u|$ and $\beta^{n'+1}(L) < |u|$. See Fig. 2. Hence $\iota(w) < n + n' + 2$.

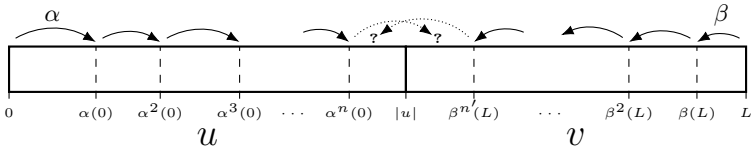


Fig. 2. Comparing $\iota(u\,v)$ with $\iota(u) + \iota(v)$.

We can also reprove a result from [BFH+20]:

Proposition 3.3. $\iota(u\,u^{\mathrm{R}}) = 2\iota(u)$.

Proof. Write n for $\iota(u)$. When $w = u\,u^{\mathrm{R}}$ and $L = |w|$, the factor $w\big(\alpha^n(0), \beta^n(L)\big)$ is $r(u) \cdot r(u)^{\mathrm{R}}$ hence is not rich. Thus $\alpha^{n+1}(0) > |u| + |r(u)^{\mathrm{R}}| = \beta^n(L)$, entailing $\iota(u\,u^{\mathrm{R}}) < 2n + 1$.

3.2 Subword circular universality index via jumping functions

The jumping functions can be used to study the circular universality index $\zeta(u)$. For this we consider the word $w = u\,u$ obtained by concatenating two copies of u, so that $L = 2\ell$. Now, instead of considering the conjugates of u, we can consider the factors $w(i, i+\ell)$ of w: see Figure 3.

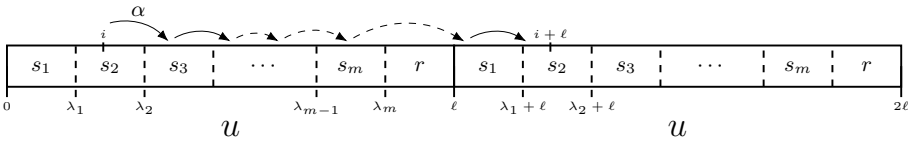


Fig. 3. Computing $\iota(u^{\sim i})$ on $w = u^2$.

This leads to a characterisation of $\zeta(u)$ in terms of α on $w = u\,u$:

$$\zeta(u) = \max_{0 \leq i < \ell} \max\big\{n \;\big|\; \alpha^n(i) \leq i + \ell\big\} \tag{6}$$

or, using $u^{\sim \ell} = u^{\sim 0}$,

$$= \max_{0 < i \leq \ell} \max\big\{n \;\big|\; \alpha^n(i) \leq i + \ell\big\}\,. \tag{7}$$

Bounding $\zeta(u)$. For $k = 0, \dots, m$, we write λ_k for the cumulative length $|s_1 \cdots s_k|$ of the k first arches of u, i.e., we let $\lambda_k \stackrel{\text{def}}{=} \alpha^k(0)$.

The following Lemma and its corollary are a version of Lemma 20 from [BFH+20] but we give a different proof.

Lemma 3.4. *Let u and u' be two conjugate words.*
(a) $\iota(u) - 1 \le \iota(u') \le \iota(u) + 1$.
(b) If furthermore $r(u) = \varepsilon$ then $\iota(u') \le \iota(u)$.

Proof. Let $s_1 \cdots s_m \cdot r$ be the arch factorization of u and assume that $u' = u^{\sim i}$ as depicted in Figure 3.
(a) If the position i falls inside some arch s_p of u (or inside the rest r) we see that $s_{p+1} \cdots s_m \cdot s_1 \cdots s_{p-1}$ is a subword of u' hence $\iota(u') \ge m - 1$. This gives $\iota(u) - 1 \le \iota(u')$, and the other inequality is obtained by exchanging the roles of u and u'.
(b) If furthermore $r = \varepsilon$, then $\lambda_{p-1} \le i < \lambda_p$ for some p. Looking at u' as a factor of $w = u^2$ (and assuming that $\alpha^{m+1}(i)$ is defined) we deduce $\alpha^{m+1}(i) \ge \alpha^{m+1}(\lambda_{p-1}) = \lambda_p + \ell > i + \ell$. This proves $\iota(u^{\sim i}) < m + 1$.

Corollary 3.5. *(a) $\iota(u) \le \zeta(u) \le \iota(u) + 1$.*
(b) Furthermore, if $r(u) = \varepsilon$, then $\zeta(u) = \iota(u)$.

4 An $O(|u| \cdot |A|)$ algorithm for $\zeta(u)$

The following crucial lemma shows that computing $\zeta(u)$ does not require checking all the conjugates $u^{\sim i}$ for $0 \le i < \ell$.

Lemma 4.1. *Let $u = a_1 \cdots a_\ell$ be a rich word with arch factorization $s_1 \cdots s_m \cdot r$.*
(a) There exists some $0 < d \le \lambda_1 \stackrel{def}{=} |s_1|$ such that $\zeta(u) = \iota(u^{\sim d})$.
(b) Furthermore, there exists $a \in A$ such that $d = \min\{i \mid a_i = a\}$, i.e., d can be chosen as a position right after a first occurrence of a letter in u.

Proof. Let $n = \zeta(u)$. For (a) it is enough to show that $\iota(u^{\sim d}) \ge n$ for some $d \in (0, \lambda_1]$.

By Equation (7) there exists some $0 < i_0 \le \ell$ such that $\alpha^n(i_0) \le i_0 + \ell$. We consider the sequence $i_0 < i_1 < \cdots < i_n$ given by $i_{k+1} = \alpha(i_k)$. If $i_n \le \ell$ then taking $d = 1$ works: monotonicity of α entails $\alpha^n(d) \le \alpha^n(i_0) \le \ell$ and we deduce $\iota(u^{\sim d}) \ge n$. Clearly $d = 1$ fulfils (b).

So assume $i_n > \ell$ and let k be the largest index such that $i_k \le \ell$ (hence $k < n$). Since $\alpha(\ell) = \ell + \lambda_1$ (recall $\lambda_1 \stackrel{\text{def}}{=} |s_1|$), monotonicity of α entails $i_{k+1} = \alpha(i_k) \le \ell + \lambda_1$, i.e., i_{k+1} lands inside the first arch of the second copy of u in w.

Let now $d \stackrel{\text{def}}{=} i_{k+1} - \ell$ so that $u^{\sim d} = w(d, d + \ell) = w(d, i_{k+1})$. Since $\alpha^{n-k-1}(i_{k+1}) = i_n \le i_0 + \ell$, one has $\alpha^{n-k-1}(d) \le i_0$ hence $\iota\big(w(d, i_0)\big) \ge n - k - 1$. We also have $\iota\big(w(i_0, i_{k+1})\big) = k + 1$ since $i_{k+1} = \alpha^{k+1}(i_0)$. This yields

$$\iota(u^{\sim d}) = \iota\big(w(d, d + \ell)\big) \ge (n - k - 1) + (k + 1) = n\,,$$

entailing (a). For (b) observe that $w(i_{k+1}-1, i_{k+1})$ is the last letter of an arch across the end of the first u in w to the beginning of the second u in w. Since it is the first occurrence of this letter in this arch, it is also in u. Since d is i_{k+1} shifted to the first copy of u, (b) is fulfilled.

Algorithm 4.2 (Computing $\zeta(u)$). For each position d such that $u(d-1, d)$ is the first occurrence of a letter in u, one computes $\iota(u^{\sim d})$ (in time $O(|u|)$ for each d), and returns the maximum value found. □

The correctness of this algorithm is given by Lemma 4.1 (if u is not rich, $\zeta(u) = 0$ and this will be found out during the computation of $\iota(u^{\sim 1})$). It runs in time $O(|A| \cdot |u|)$ since there are at most $|A|$ values for d, starting with $d = 1$.

There are two heuristic improvements that can speed up the algorithm[3]:

- As soon as we have encountered two different values $\iota(u^{\sim d}) \neq \iota(u^{\sim d'})$, we can stop the search for a maximum in view of corollary 3.5.(a).
  For example, for $u = \texttt{aabaccb}$, the first occurrences of `a`, `b`, and `c`, are with $d = 1, 3$ and 5. So one starts with computing $\iota(u^{\sim 1}) = \iota(\texttt{abaccb}\,\texttt{a}) = 2$. Then one computes $\iota(u^{\sim 3}) = \iota(\texttt{accb}\,\texttt{aab}) = 1$. Now, and since we have encountered two different values, we may conclude immediately that $\zeta(u) = 2$ without the need to compute $\iota(u^{\sim 5})$.
- When computing some $\iota(u^{\sim d})$ leads us to notice $r(u^{\sim d}) = \varepsilon$, we can stop the search in view of corollary 3.5.(b).
  For example, and again with $u = \texttt{aabaccb}$, the computation of $\iota(u^{\sim 1})$ led us to the arch-factorization $u^{\sim 1} = \texttt{abac} \cdot \texttt{cba} \cdot \varepsilon$, with 2 arches and with $r(u^{\sim 1}) = \varepsilon$. We may conclude immediately that $\zeta(u) = \iota(u^{\sim 1}) = 2$ without trying the remaining conjugates.

Observe that the above algorithm does not have to explicitly build $u^{\sim d}$. It is easy to adapt any naive algorithm for $\iota(u)$ so that it starts at some position d and wraps around when reaching the end of u.

5 Subword universality signatures

In this section, we write $\iota_*(u)$, $r_*(u)$, etc., to denote the values of $\iota(u)$, $r(u)$, etc., *when one assumes that $A(u)$ is the underlying alphabet*. This notation is less heavy than writing, e.g., $\iota_{A(u)}(u)$, but it is needed since we shall consider simultaneously $\iota_*(u)$ and $\iota_*(v)$ when $A(u) \neq A(v)$, i.e., when the two universality indexes have been obtained in different contexts.

When u is a word, we define a function S_u on words via:

$$S_u(x) = \big\langle \iota_*(x\,u), A\big(r_*(x\,u)\big)\big\rangle \quad \text{for all } x \text{ such that } A(u) \not\subseteq A(x). \tag{8}$$

In other words, $S_u(x)$ is a summary of the arch factorization of $x\,u$: it records the number of arches in $x\,u$ and the letters of the rest $r_*(x\,u)$, assuming that the alphabet is $A(x\,u)$.

[3] They do not improve the *worst-case* complexity.

Note that $S_u(x)$ is only defined when $A(u) \not\subseteq A(x)$, i.e., when at least one letter from u does not appear in x. With this restriction, $S_u(x)$ and $S_u(x')$ coincide (or are both undefined) whenever $A(x) = A(x')$. For this reason, we sometimes write $S_u(B)$, where B is a set of letters, to denote any $S_u(x)$ with $A(x) = B$.

We are now almost ready to introduce the main new object: a compact data structure with enough information for computing S_u on arbitrary arguments.

With a word u we associate $e(u)$, a word listing the letters of u in the order of their first appearance in u. For example, by underlining the first occurrence of each letter in $u = \underline{\mathtt{c}}\mathtt{ca}\underline{\mathtt{a}}\mathtt{cab}\underline{\mathtt{b}}\mathtt{cbba}$ we show $e(u) = \mathtt{cab}$. We also write $f(u)$ for the word listing the letters of u in order of their last occurrence: in the previous example $f(u) = \mathtt{cba}$.

Definition 5.1. *The* subword universality signature *of a word u is the pair $\Sigma(u) = \langle e(u), \mathtt{s}_u \rangle$ where $\mathtt{s}_u$ is S_u restricted to the strict suffixes of $e(u)$.*

Example 5.2. With $u = \underline{\mathtt{a}}\mathtt{ab}\underline{\mathtt{a}}\mathtt{c}\underline{\mathtt{c}}$ we have:

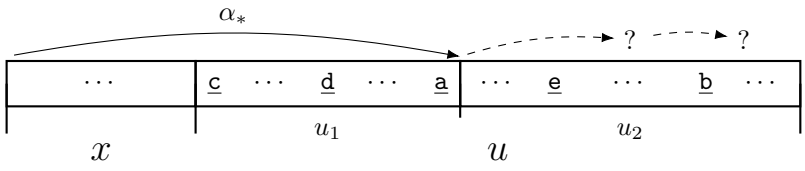
$$\Sigma(u) = \begin{cases} e(u) = \mathtt{abc} \\ \mathtt{s}_u = \begin{cases} \varepsilon \mapsto \langle 1, \varnothing \rangle \\ \mathtt{c} \mapsto \langle 1, \{\mathtt{a}, \mathtt{c}\} \rangle \\ \mathtt{bc} \mapsto \langle 2, \varnothing \rangle \end{cases} \end{cases} \quad \text{in view of:} \quad \begin{array}{c} \varepsilon \cdot u = \mathtt{aabac} \cdot \varepsilon \\ \mathtt{c} \cdot u = \mathtt{caab} \cdot \mathtt{ac} \\ \mathtt{bc} \cdot u = \mathtt{bca} \cdot \mathtt{abac} \cdot \varepsilon \end{array}$$

NB: the strict suffixes of $e(u)$ are ε, $\mathtt{c}$ and $\mathtt{bc}$.

While finite (and quite small) $\Sigma(u)$ contains enough information for computing S_u on any argument x on any alphabet. One can use the following algorithm:

Algorithm 5.3 (Computing $S_u(x)$ from $\Sigma(u)$).
Given inputs x and $\Sigma(u) = \langle e(u), \mathtt{s}_u \rangle$ we proceed as follows:
(a) Retrieve $A(u)$ from $e(u)$. Check that $A(u) \not\subseteq A(x)$, since otherwise $S_u(x)$ is undefined.
(b) Now with $x \in \mathit{dom}(S_u)$, let y be the longest suffix of $e(u)$ with $A(y) \subseteq A(x)$ —necessarily y is a *strict* suffix of $e(u)$— and extract $\langle n_y, B_y \rangle$ from $\mathtt{s}_u(y)$.
(c.1) If $A(x) \subseteq A(u)$, return $S_u(x) = \langle n_y, B_y \rangle$.
(c.2) Similarly, if $n_y = 1$ return $S_u(x) = \langle n_y, B_y \rangle$.
(c.3) Otherwise return $S_u(x) = \langle 1, A(u) \rangle$.

Proof (of correctness). Assume $x \in \mathit{dom}(S_u)$. Since u contains a letter not appearing in x, the first arch of $x\,u$ ends inside u, so let us consider the factorization $u = u_1 u_2$ such that $x\,u_1$ is the first arch of $x\,u$ (see picture below, where $e(u)$ is underlined).



Now u_1 has a last letter, say a, that appears only once in u_1 and not at all in x. Observe that a letter b appears after a in $e(u)$ iff it does not appear in u_1, and thus must appear in x. Hence the y computed in step (b) is the suffix of $e(u)$ after a (in the above picture y would be $\mathtt{eb}$).

If $A(x) \subseteq A(u)$ then $y\,u_1$ is rich, and is in fact an arch since its last letter, a, appears only once. So $S_u(x)$ and $S_u(y)$ coincide and step (c.1) is correct.

In case $A(x) \not\subseteq A(u)$, both x and u contain some letters that are absent from the other word, so necessarily $\iota_*(x\,u) = 1$ and $r_*(x\,u) = u_2$. There only remains to compute $A(u_2)$ from $\Sigma(u)$. We know that $\mathtt{s}_u(y) = \langle n_y, B_y\rangle$. If $n_y > 1$ this means that u_2 contains at least another $A(u)$-arch, so $A(u_2) = A(u)$ and step (c.3) is correct. If $n_y = 1$ this means that $y\,u$ only has one arch, namely $y\,u_1$, and B_y provides $A(u_2)$: step (c.2) is correct in this case.

Remark 5.4 (Space and time complexity for Algorithm 5.3). For simplifying our complexity evaluation, we assume that there is a fixed maximum size for alphabets so that storing a letter $a \in A$ uses space $O(1)$, e.g., 64 bits. When storing $\Sigma(u)$, the $e(u)$ part uses space $O(|A|)$. Now $\mathtt{s}_u$ can be represented in space $O(|A| \log |u|)$ when $e(u)$ and $f(u)$ are known: it contains at most $|A|$ pairs $\langle n_x, B_x\rangle$ where x is a suffix of $e(u)$ and B_x is always the alphabet of a strict suffix of $f(u)$: x and B_x can thus be represented by a position (or a letter) in $e(u)$ and $f(u)$. The n_x values each need at most $\log |u|$ bits.

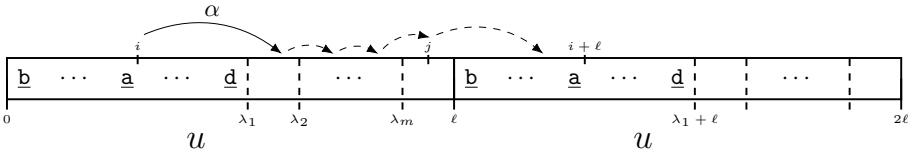
Regarding time, the algorithm runs in time $O\big(|x| + |\Sigma(u)| + |A(u)|\big)$. □

5.1 Universality indexes from signatures

Obviously the signature $\Sigma(u)$ contains enough information for retrieving $\iota_*(u)$: this is found in $\mathtt{s}_u(\varepsilon)$. More interestingly, one can also retrieve $\zeta_*(u)$:

Proposition 5.5. *Let u be a word with $\iota_*(u) = m$. Then $\zeta_*(u) = m+1$ iff there exists a strict suffix x of $e(u)$ with $\mathtt{s}_u(x) = \langle n_x, B_x\rangle$ such that $n_x = m+1$ and $A(x) \subseteq B_x$. Otherwise $\zeta_*(u) = m$.*

Proof. ($\Leftarrow$): assume $\mathtt{s}_u(x) = \langle m+1, B_x\rangle$ with $A(x) \subseteq B_x$. Thus $\iota_*(x\,u) = m+1$. Factor u as $u = u_1u_2r$ such that $x\,u_1$ is the first arch of $x\,u$ and such that $r = r_*(x\,u)$ is its rest. Then u_2 contains m arches and $B_x = A(r)$. Let now $u' \stackrel{\text{def}}{=} r\,u_1u_2$. We claim that $\iota_*(u') = m+1$. Indeed $r\,u_1$ is rich since $x\,u_1$ is rich and $A(x) \subseteq A(r)$, so $\iota_*(r\,u_1u_2) \geq m+1$. Since u' and u are conjugates, we deduce $\zeta_*(u) = \iota_*(u') = m+1$ from Corollary 3.5.(a).
($\Rightarrow$): assume $\zeta_*(u) = m+1$. By Lemma 4.1 we know that $\iota_*\big(u^{\sim i}\big) = m+1$ for some position $0 < i \leq \lambda_1$ falling just after a first occurrence of a letter in u. Looking at factors of $w = u\,u$ as we did before, we have $\alpha^{m+1}(i) \leq i+\ell$, leading to $j \stackrel{\text{def}}{=} \alpha^m(i) \leq \ell$ (see picture below).



Define now x as the suffix of $e(u)$ that contains all letters in $u(i, \lambda_1)$, that is, all underlined letters to the right of i. This is a strict suffix since $i > 0$. Now $x\,u(0,i)$ is rich, and $u(i,j)$ is made of exactly m arches, so $\iota_*(x\,u) = n_x = m+1$ and $r_*(x\,u) = u(j,\ell)$.
Then $B_x = A\big(u(j,\ell)\big)$ and $w(j, i+\ell)$ is rich, so $w(j,\ell)$ contains all letters missing from $w(i, i+\ell) = u(0,i)$. In other words $B_x \supseteq A(x)$, concluding the proof.

Corollary 5.6 (Computing universality indexes from signatures). *One can compute $\iota_*(u)$ and $\zeta_*(u)$ from $\Sigma(u)$ in time $(|A| + \log|u|)^{O(1)}$.*

Actual implementations can use heuristics based on Lemma 3.4.(b): if $\mathtt{s}_u(\varepsilon) = \langle m, \varnothing\rangle$ then $\zeta_*(u) = m$.

5.2 Combining signatures

Subword universality signatures can be computed compositionally.

Algorithm 5.7 (Combining signatures). The following algorithm takes as input the signatures $\Sigma(u)$ and $\Sigma(v)$ of any two words and computes $\Sigma(u\,v)$:

(a) Retrieve $A(u)$ and $A(v)$ from $e(u)$ and $e(v)$, then compute $e(u\,v)$ as $e(u)\,e'$ where e' is the subword of $e(v)$ that only retains the letters from $A(v) \smallsetminus A(u)$.
(b) Consider now any strict suffix x of $e(u\,v)$ and compute $\mathtt{s}_{u\,v}(x)$ as follows:
(b.1) If $A(v) \not\subseteq A(x) \cup A(u)$ then let $\mathtt{s}_{u\,v}(x) \stackrel{\text{def}}{=} S_v\big(x\,e(u)\big)$, using Algorithm 5.3.
(b.2) If $A(v) \subseteq A(x) \cup A(u)$, then $A(u) \not\subseteq A(x)$. Write $\big\langle n, B\big\rangle$ for $\mathtt{s}_u(x)$:
(b.2.1) If now $A(v) \cup B \neq A(x) \cup A(u)$ then let $\mathtt{s}_{u\,v}(x) \stackrel{\text{def}}{=} \big\langle n, A(v) \cup B\big\rangle$.
(b.2.2) Otherwise retrieve $\mathtt{s}_v(B) = \big\langle n', B'\big\rangle$ and let $\mathtt{s}_{u\,v}(x) \stackrel{\text{def}}{=} \big\langle n + n', B'\big\rangle$.

Proof (of correctness). Step (a) for $e(u\,v)$ is correct.
In step (b) we want to compute $S_{u\,v}(x)$. Now $x\,(u\,v) = (x\,u)\,v$ so $S_{u\,v}(x)$ coincides with $S_v(x\,u)$ *when the latter is defined* . This is the case in step (b.1) where one computes $S_v(x\,u)$ by replacing $x\,u$ with $x\,e(u)$, an argument with same alphabet (recall that the algorithm does not have access to u itself).
In step (b.2) where $S_v(x\,u)$ is not defined, computing $S_u(x)$ provides n and $B = A(r)$ for the arch factorization $x\,u = s_1 \cdots s_n \cdot r$ of $x\,u$.
We can continue with the arch factorization of $r\,v$ and combine the two sets of arches if these factorizations rely on the same alphabet: this is step (b.2.2).
Otherwise, $r\,v$ only uses a subset of the letters of $x\,u$. There won't be a new arch, only a longer rest: $r_*(x\,u\,v) = r\,v$. Step (b.2.1) is correct.

Note that Algorithm 5.7 runs in time $O\big(|A(u\,v)| + |\Sigma(u)| + |\Sigma(v)|\big)$ and that the result has linear size $|\Sigma(u\,v)| = O(|\Sigma(u)| + |\Sigma(v)|)$.

6 Universality indexes for SLP-compressed words

We are now ready to compute the universality indexes of SLP-compressed words. Recall that an SLP X is an acyclic context-free grammar in Chomsky normal

form where furthermore each non-terminal has only one production rule, i.e., the grammar is deterministic (see survey [Loh12]). SLPs are the standard mathematical model for compression of texts and files and, modulo polynomial-time encodings, it encompasses most compression schemes used in practice.

Formally, an SLP X with m rules is a list $\langle N_1 \to \rho_1; \cdots; N_m \to \rho_m \rangle$ of production rules where each right-hand side ρ_i is either a letter a from A or a concatenation $N_j\, N_{j'}$ of two nonterminals with $j, j' < i$. It has size $|X| = O(m \log m)$ when A is fixed.

Each nonterminal N_i encodes a word, its *expansion*, given inductively via:

$$exp(N_i) \stackrel{\text{def}}{=} \begin{cases} a & \text{if } \rho_i = a, \\ exp(N_j)\, exp(N_{j'}) & \text{if } \rho_i = N_j\, N_{j'}. \end{cases}$$

Finally, the expansion $exp(X)$ of the SLP itself is the expansion $exp(N_m)$ of its last nonterminal. This is a word (or file) of length $2^{O(|X|)}$ and one of the main goals in the area of compressed data science is to develop efficient methods for computing relevant information about $exp(X)$ directly from X, i.e., without actually decompressing the word or file.

In this spirit we can state:

Theorem 6.1. *The universality indexes $\iota\big(exp(X)\big)$ and $\zeta\big(exp(X)\big)$ can be computed from an SLP X in bilinear time $O\big(|A| \cdot |X|\big)$.*

Proof. One just computes $\Sigma\big(exp(N_1)\big)$, …, $\Sigma\big(exp(N_k)\big)$ for the non-terminals $N_1, \ldots, N_k$ of X. If N_i is associated with a production rule $N_i \to N_{i_1} N_{i_2}$, we compute $\Sigma\big(exp(N_i)\big)$ by combining $\Sigma\big(exp(N_{i_1})\big)$ and $\Sigma\big(exp(N_{i_2})\big)$ via Algorithm 5.7 (recall that $i_1, i_2 < i$ since the grammar is acyclic). If N_i is associated with a production $N_i \to a$ for some $a \in A$, then $\Sigma\big(exp(N_i)\big) = \Sigma(a)$ is trivial. In the end we can extract the universality indexes of $exp(X)$, defined as $exp(N_k)$, from $\Sigma\big(exp(N_k)\big)$ using Corollary 5.6. Note that all signatures have size $O(|A| \cdot |X|)$ since for any $u = exp(N_i)$, $\log |u|$ is in $O(|X|)$. With the analysis of Algorithm 5.7 and Corollary 5.6, this justifies the claim about complexity.

7 Conclusion

We introduced arch-jumping functions and used them to describe and analyse the subword universality and circular universality indexes $\iota(u)$ and $\zeta(u)$. In particular, this leads to a simple and elegant algorithm for computing $\zeta(u)$.

In a second part we defined the subword universality signatures of words, a compact data structure with enough information for extracting $\iota(u)$ and $\zeta(u)$. Since one can efficiently compute the signature of $u\,v$ by composing the signatures of u and v, we obtain a polynomial-time algorithm for computing $\iota(X)$ and $\zeta(X)$ when X is a SLP-compressed word. This raises our hopes that one can compute some subword-based descriptive complexity measures on compressed words, despite the known difficulties encountered when reasoning about subwords.

References

[BFH+20] L. Barker, P. Fleischmann, K. Harwardt, F. Manea, and D. Nowotka. Scattered factor-universality of words. In *Proc. 24th Int. Conf. Developments in Language Theory (DLT 2020)*, volume 12086 of *LNCS*, pages 14–28. Springer, 2020.

[BSS12] M. Bojańczyk, L. Segoufin, and H. Straubing. Piecewise testable tree languages. *Logical Methods in Comp. Science*, 8(3), 2012.

[CP18] O. Carton and M. Pouzet. Simon's theorem for scattered words. In *Proc. 22nd Int. Conf. Developments in Language Theory (DLT 2018)*, volume 11088 of *LNCS*, pages 182–193. Springer, 2018.

[DFK+21] J. D. Day, P. Fleischmann, M. Kosche, T. Koß, F. Manea, and S. Siemer. The edit distance to k-subsequence universality. In *Proc. 38th Int. Symp. Theoretical Aspects of Computer Science (STACS 2021)*, volume 187 of *LIPiCS*, pages 25:1–25:19. Leibniz-Zentrum für Informatik, 2021.

[DGK08] V. Diekert, P. Gastin, and M. Kufleitner. A survey on small fragments of first-order logic over finite words. *Int. J. Foundations of Computer Science*, 19(3):513–548, 2008.

[DPFD19] Day J. D., Fleischmann P., Manea F., and Nowotka D. k-spectra of weakly c-balanced words. In *Proc. 23rd Int. Conf. Developments in Language Theory (DLT 2019)*, volume 11647 of *LNCS*, pages 265–277. Springer, 2019.

[FGN21] P. Fleischmann, S. B. Germann, and D. Nowotka. Scattered factor universality – the power of the remainder. arXiv:2104.09063 [cs.CL], April 2021.

[FHH+22] P. Fleischmann, L. Haschke, A. Huch, A. Mayrock, and D. Nowotka. Nearly k-universal words - investigating a part of Simon's congruence. In *Proc. 24th Int. Conf. Descriptional Complexity of Formal Systems (DCFS 2022)*, volume 13439 of *LNCS*, pages 57–71. Springer, 2022.

[FK18] L. Fleischer and M. Kufleitner. Testing Simon's congruence. In *Proc. 43rd Int. Symp. Math. Found. Comp. Sci. (MFCS 2018)*, volume 117 of *LIPiCS*, pages 62:1–62:13. Leibniz-Zentrum für Informatik, 2018.

[GLHK+20] J. Goubault-Larrecq, S. Halfon, P. Karandikar, K. Narayan Kumar, and Ph. Schnoebelen. The ideal approach to computing closed subsets in well-quasi-orderings. In *Well Quasi-Orders in Computation, Logic, Language and Reasoning*, volume 53 of *Trends in Logic*, chapter 3, pages 55–105. Springer, 2020.

[GS16] J. Goubault-Larrecq and S. Schmitz. Deciding piecewise testable separability for regular tree languages. In *Proc. 43rd Int. Coll. Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *LIPiCS*, pages 97:1–97:15. Leibniz-Zentrum für Informatik, 2016.

[Héb91] J.-J. Hébrard. An algorithm for distinguishing efficiently bit-strings by their subsequences. *Theoretical Computer Science*, 82(1):35–49, 1991.

[HSZ17] S. Halfon, Ph. Schnoebelen, and G. Zetzsche. Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In *Proc. 32nd ACM/IEEE Symp. Logic in Computer Science (LICS 2017)*, pages 1–12. IEEE Comp. Soc. Press, 2017.

[KKMP22] M. Kosche, T. Koß, F. Manea, and V. Pak. Subsequences in bounded ranges: Matching and analysis problems. In *Proc. 16th Int. Conf. Reachability Problems (RP 2022)*, volume 13608 of *LNCS*, pages 140–159. Springer, 2022.

[KKMS21] M. Kosche, T. Koß, F. Manea, and S. Siemer. Absent subsequences in words. In *Proc. 15th Int. Conf. Reachability Problems (RP 2021)*, volume 13035 of *LNCS*, pages 115–131. Springer, 2021.

[KKMS22] M. Kosche, T. Koß, F. Manea, and S. Siemer. Combinatorial algorithms for subsequence matching: A survey. In *Proc. 12th Int. Workshop Non-Classical Models of Automata and Applications (NCMA 2022)*, volume 367 of *EPTCS*, pages 11–27, 2022.

[KS15] P. Karandikar and Ph. Schnoebelen. Generalized Post embedding problems. *Theory of Computing Systems*, 56(4):697–716, 2015.

[KS19] P. Karandikar and Ph. Schnoebelen. The height of piecewise-testable languages and the complexity of the logic of subwords. *Logical Methods in Comp. Science*, 15(2), 2019.

[Loh12] M. Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.

[Mat98] O. Matz. On piecewise testable, starfree, and recognizable picture languages. In *Proc. Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS '98)*, volume 1378 of *LNCS*, pages 203–210. Springer, 1998.

[Sch21] Ph. Schnoebelen. On flat lossy channel machines. In *Proc. 29th EACSL Conf. Computer Science Logic (CSL 2021)*, volume 183 of *LIPiCS*, pages 37:1–37:22. Leibniz-Zentrum für Informatik, 2021.

[Sim72] I. Simon. *Hierarchies of Event with Dot-Depth One*. PhD thesis, University of Waterloo, Dept. Applied Analysis and Computer Science, Waterloo, ON, Canada, 1972.

[Sim75] I. Simon. Piecewise testable events. In *Proc. 2nd GI Conf. on Automata Theory and Formal Languages*, volume 33 of *LNCS*, pages 214–222. Springer, 1975.

[Sim03] I. Simon. Words distinguished by their subwords. In *Proc. 4th Int. Conf. on Words (WORDS 2003)*, 2003.

[SS83] J. Sakarovitch and I. Simon. Subwords. In M. Lothaire, editor, *Combinatorics on Words*, volume 17 of *Encyclopedia of Mathematics and Its Applications*, chapter 6, pages 105–142. Cambridge Univ. Press, 1983.