Testing Simon's congruence

Lukas Fleischer[1]
FMI, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
fleischer@fmi.uni-stuttgart.de

Manfred Kufleitner
Department of Computer Science, Loughborough University
Epinal Way, Loughborough LE11 3TU, United Kingdom
m.kufleitner@lboro.ac.uk

Abstract

Piecewise testable languages are a subclass of the regular languages. There are many equivalent ways of defining them; Simon's congruence $\sim_k$ is one of the most classical approaches. Two words are $\sim_k$-equivalent if they have the same set of (scattered) subwords of length at most k. A language L is piecewise testable if there exists some k such that L is a union of $\sim_k$-classes.

For each equivalence class of $\sim_k$, one can define a canonical representative in shortlex normal form, that is, the minimal word with respect to the lexicographic order among the shortest words in $\sim_k$. We present an algorithm for computing the canonical representative of the $\sim_k$-class of a given word $w \in A^*$ of length n. The running time of our algorithm is in $\mathcal{O}(|A|\, n)$ even if $k \leq n$ is part of the input. This is surprising since the number of possible subwords grows exponentially in k. The case $k > n$ is not interesting since then, the equivalence class of w is a singleton. If the alphabet is fixed, the running time of our algorithm is linear in the size of the input word. Moreover, for fixed alphabet, we show that the computation of shortlex normal forms for $\sim_k$ is possible in deterministic logarithmic space.

One of the consequences of our algorithm is that one can check with the same complexity whether two words are $\sim_k$-equivalent (with k being part of the input).

2012 ACM Subject Classification Mathematics of computing → Combinatorics on words, Theory of computation → Formal languages and automata theory

Keywords and phrases regular language, scattered subword, piecewise testability, string algorithm

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.62

Acknowledgements We would like to thank the anonymous reviewers for their helpful comments. Their suggestions significantly improved the presentation of this paper.

1 Introduction

We write $u \preccurlyeq v$ if the word u is a (scattered) subword of v, that is, if there exist factorizations $u = u_1 \cdots u_n$ and $v = v_0 u_1 v_1 \cdots u_n v_n$. In the literature, subwords are sometimes called *piecewise* subwords to distinguish them from factors. Higman showed that, over finite alphabets, the relation $\preccurlyeq$ is a well-quasi-ordering [3]. This means that every language contains only finitely many minimal words with respect to the subword ordering. This led to the consideration of piecewise testable languages. A language L is *piecewise testable* if there exists a finite set of words T such that $v \in L$ only depends on $\{u \in T \mid u \preccurlyeq v\}$; in

[1] Supported by the German Research Foundation (DFG) under grant DI 435/5–2.

other words, the occurrence and non-occurrence of subwords in T determines membership in L. Equivalently, a language L is piecewise testable if it is a finite Boolean combination of languages of the form $A^*a_1A^* \cdots a_nA^*$ with $a_i \in A$ (using the above notation, the sequences $a_1 \cdots a_n$ in this combination give the words in T). The piecewise testable languages are a subclass of the regular languages and they play a prominent role in many different areas. For instance, they correspond to the languages definable in alternation-free first-order logic [20] which plays an important role in database queries. They also occur in learning theory [10, 15] and computational linguistics [2, 14].

In the early 1970s, Simon proved his famous theorem on piecewise testable languages: A language is piecewise testable if and only if its syntactic monoid is finite and $\mathcal{J}$-trivial [18]. An immediate consequence of Simon's Theorem is that it is decidable whether or not a given regular language L is piecewise testable. Already in his PhD thesis [17], Simon considered the complexity of this problem when L is given as a deterministic finite automaton (DFA). His algorithm can be implemented to have a running time of $\mathcal{O}(2^{|A|}n^2)$ for an n-state DFA over the alphabet A. This result was successively improved over the years [1, 9, 19, 21] with the latest algorithm having a running time of $\mathcal{O}(|A|^2\, n)$; see [8]. If the input is a DFA, then the problem is NL-complete [1]; and if the input is a nondeterministic finite automaton, the problem is PSPACE-complete [4]. Restricting the length of the relevant subwords T to some constant k leads to the notion of k-piecewise testable languages. At first sight, it is surprising that, for every fixed $k \geq 4$, deciding whether a given DFA accepts a k-piecewise testable language is coNP-complete [8]; see also [11].

One of the main tools in the original proof of Simon's Theorem is the congruence $\sim_k$ for $k \in \mathbb{N}$. By definition, two words u and v satisfy $u \sim_k v$ if u and v have the same subwords of length at most k. Naturally, the relation $\sim_k$ is nowadays known as *Simon's congruence*. It is easy to see that a language L is piecewise testable if and only if there exists k such that L is a union of $\sim_k$-classes. Understanding the combinatorial properties of $\sim_k$ is one of the main tools in the study of piecewise testable languages. For example, in the proof of his theorem, Simon already used that $(uv)^k \sim_k (uv)^k u$ for all words u, v. Upper and lower bounds on the index of $\sim_k$ were given by Kátai-Urbán et al. [7] and Karandikar et al. [6].

There are two natural approaches for testing whether or not $u \sim_k v$ holds. The first approach constructs a DFA $\mathcal{A}_{k,u}$ for the language $\{w \preccurlyeq u \mid k \geq |w|\}$ of the subwords of u of length at most k and a similar DFA $\mathcal{A}_{k,v}$ for v. Then $u \sim_k v$ if and only if $\mathcal{A}_{k,u}$ and $\mathcal{A}_{k,v}$ accept the same language. This can be tested with Hopcroft's algorithm in time almost linear in the size of the automata [5]. Here, *almost linear* in n means $\mathcal{O}(n \cdot a(n))$ where $a(n)$ is the inverse Ackermann function. It is possible to construct the automata such that $\mathcal{A}_{k,u}$ has at most $k\,|u| + 2$ states, see the remark at the end of Section 2 below. Hence, the resulting test is almost linear in $|A|\; k\; |uv|$ if the alphabet is A.

The second approach to testing $u \sim_k v$ is the computation of normal forms. A normal form is a unique representative of a $\sim_k$-class. In particular, we have $u \sim_k v$ if and only if u and v have the same normal form. By computing the normal forms for both words and then checking whether they are identical, the complexity of this test of $u \sim_k v$ is the same as the computation of the normal forms. We should mention that the computation of normal forms is also interesting in its own right since it can provide some insight into the combinatorial properties of $\sim_k$. Normal forms for $k = 2$ and $k = 3$ were considered by Kátai-Urbán et al. [7] and normal forms for $k = 4$ were given by Pach [12]. An algorithm for computing normal forms for arbitrary k was found only recently by Pach [13]. Its running time is $\mathcal{O}(|A|^k\,(n + |A|))$ for inputs of length n over the alphabet A, that is, polynomial for fixed k and exponential otherwise.

We significantly improve this result by providing an algorithm with a running time in $\mathcal{O}(|A|\,n)$ even if k is part of the input. For every fixed alphabet, the running time is linear, which is optimal. Moreover, the algorithm can easily be adapted to run in deterministic logarithmic space, thereby addressing an open problem from [7]. As a consequence we can check with the same running time (or the same complexity) whether two given words are $\sim_k$-equivalent even if k is part of the input, thereby considerably improving on the above automaton approach.

Our algorithm actually does not compute just some normal form but the *shortlex normal form* of the input word u, *i.e.*, a shortest, and among all shortest the lexicographically smallest, word v such that $u \sim_k v$. Our main tools are so-called *rankers* [16, 22]. For each position i in the input word, the algorithm computes the lengths of the shortest X-rankers and Y-rankers reaching i. One can then derive the shortlex normal form by deleting and sorting certain letters based on these attributes. A more detailed outline of the paper is given in Section 3.

2 Preliminaries

Let A be a finite alphabet. The elements in A are called *letters* and a sequence of letters $u = a_1 \cdots a_\ell$ is a *word* of *length* $|u| = \ell$. The set of all words over the alphabet A is A^*. Throughout this paper, a, b and c are used to denote letters. For a word $a_1 \cdots a_\ell$, the numbers $\{1, \ldots, \ell\}$ are called *positions* of the word, and i is a *c-position* if $a_i = c$. The letter a_i is the *label* of position i. Two positions i and j with $i < j$ are *consecutive* c-positions if $a_i = a_j = c$ and $a_\ell \neq c$ for all $\ell \in \{i+1, \ldots, j-1\}$. A word $a_1 \cdots a_\ell$ is a *subword* of a word $v \in A^*$ if v can be written as $v = v_0 a_1 \cdots v_{\ell-1} a_\ell v_\ell$ for words $v_i \in A^*$. We write $u \preccurlyeq v$ if u is a subword of v. A *congruence* on A^* is an equivalence relation $\sim$ such that $u \sim v$ implies $puq \sim pvq$ for all $u, v, p, q \in A^*$. For $k \in \mathbb{N}$, *Simon's congruence* $\sim_k$ on A^* is defined by $u \sim_k v$ if and only if u and v contain the same subwords of length at most k.

We assume that the letters of the alphabet are totally ordered. A word u is *lexicographically smaller than* v if, for some common $p \in A^*$, there exists a prefix pa of u and a prefix pb of v such that $a < b$. (We apply the lexicographic order only for words of the same length; in particular, we do not care about the case when u is a proper prefix of v.) Given a congruence $\sim$ on A^*, we define the *shortlex normal form* of a word u to be the shortest word v such that $u \sim v$ and such that no other word $w \in A^*$ with $w \sim v$ and $|w| = |v|$ is lexicographically smaller than v. In other words, we first pick the shortest words in the $\sim$-class of u and among those, we choose the lexicographically smallest one.

Our main tools are so-called *rankers* [16, 22]. An X*-ranker* is a nonempty word over the alphabet $\{\mathsf{X}_a \mid a \in A\}$ and a Y*-ranker* is a nonempty word over $\{\mathsf{Y}_a \mid a \in A\}$. The length of a ranker is its length as a word. The modality X_a means neXt-a and is interpreted as an instruction of the form "go to the next a-position"; similarly, Y_a is a shorthand for Yesterday-a and means "go to the previous a-position". More formally, we let $\mathsf{X}_a(u) = i$ if i is the smallest a-position of u, and we let $r\mathsf{X}_a(u) = i$ for a ranker r if i is the smallest a-position greater than $r(u)$. Symmetrically, we let $\mathsf{Y}_a(u) = i$ if i is the greatest a-position of u and we let $r\mathsf{Y}_a(u) = i$ if i is the greatest a-position smaller than $r(u)$. In particular, rankers are processed from left to right. Note that the position $r(u)$ for a ranker r and a word u can be undefined. If $r(u)$ is defined, then we say that r *reaches* the position $r(u)$. Similarly, r *visits* a position i if $s(u) = i$ for some prefix s of r. A word $b_1 \cdots b_\ell$ *defines* an X-ranker $\mathsf{X}_{b_1} \cdots \mathsf{X}_{b_\ell}$ and a Y-ranker $\mathsf{Y}_{b_\ell} \cdots \mathsf{Y}_{b_1}$. We have $u \preccurlyeq v$ if and only if $r(v)$ is defined for the

X-ranker (resp. Y-ranker) r defined by u. Similarly, if r is the X-ranker defined by u and s is the Y-ranker defined by v, then $uv \preccurlyeq w$ if and only if $r(w) < s(w)$. The correspondence between X-rankers and subwords leads to the following automaton construction.

▶ Remark. Let u be a word of length n. We construct a DFA $\mathcal{A}_{k,u}$ for the language $\{w \preccurlyeq u \mid |w| \leq k\}$. The set of states is $\{(0,0)\} \cup \{1,\dots,k\} \times \{1,\dots,n\}$ plus some sink state which collects all missing transitions. The initial state is $(0,0)$ and all states except for the sink state are final. We have a transition $(\ell, i) \xrightarrow{a} (\ell+1, j)$ if $\ell < k$ and j is the smallest a-position greater than i. The idea is that the first component counts the number of instructions and the second component gives the current position.

3 Attributes and outline of the paper

To every position $i \in \{1,\dots,n\}$ of a word $a_1 \cdots a_n \in A^n$, we assign an *attribute* (x_i, y_i) where x_i is the length of a shortest X-ranker reaching i and y_i is the length of a shortest Y-ranker reaching i. We call x_i the *x-coordinate* and y_i the *y-coordinate* of position i.

▶ **Example 1.** We will use the word $u = bacbaabada$ as a running example throughout this paper. The attributes of the positions in u are as follows:

$$\overset{12}{b}\;\overset{12}{a}\;\overset{11}{c}\;\overset{22}{b}\;\overset{23}{a}\;\overset{32}{a}\;\overset{31}{b}\;\overset{42}{a}\;\overset{11}{d}\;\overset{21}{a}$$

The letter a at position 5 can be reached by the Y-ranker $\mathsf{Y}_b\mathsf{Y}_a\mathsf{Y}_a$ and the a at position 6 can be reached by the X-ranker $\mathsf{X}_c\mathsf{X}_a\mathsf{X}_a$. Both rankers visit both positions 5 and 6. No X-ranker visiting position 6 can avoid position 5 and no Y-ranker visiting position 5 can avoid position 6. Deleting either position 5 or 6 reduces the attributes of the other position to $(2,2)$.

We propose a two-phase algorithm for computing the shortlex normal form of a word u within its $\sim_k$-class. The first phase deletes letters and results in a word of minimal length within the $\sim_k$-class of u. The second phase sorts blocks of letters to get the minimal word with respect to the lexicographic ordering. Both phases depend on the attributes. The computation of the attributes and the first phase are combined as follows.

Phase 1a: Compute all x-coordinates from left to right.
Phase 1b: Compute all y-coordinates from right to left while dynamically deleting a position whenever the sum of its coordinates would be bigger than $k+1$.
Phase 2: Swap consecutive letters b and a (with $b > a$) whenever they have the same attributes and the sum of the x- and the y-coordinate equals $k+1$.

As we will show, a crucial property of Phase 1b is that the dynamic process does not mess up the x-coordinates of the remaining positions that were previously computed in Phase 1a.

The **outline** of the paper is as follows. In Section 4, we prove that successively deleting all letters where the sum of the attributes is bigger than $k+1$ eventually yields a length-minimal word within the $\sim_k$-class of the input. This statement has two parts. The easier part is to show that we can delete such a position without changing the $\sim_k$-class. The more difficult part is to show that if no such deletions are possible, the word is length-minimal within its $\sim_k$-class. In particular, no other types of deletions are required. Also note that deleting letters can change the attributes of the remaining letters.

Section 5 has two components. First, we show that commuting consecutive letters does not change the $\sim_k$-class if (a) the two letters have the same attribute and (b) the sum of the x- and the y-coordinate equals $k+1$. Moreover, such a commutation does not change any attributes. Then, we prove that no other types of commutation are possible within the $\sim_k$-class. This is quite technical to formalize since, a priori, we could temporarily leave the $\sim_k$-class only to re-enter it again with an even smaller word.

Finally, in Section 6, we present an easy and efficient algorithm for computing shortlex normal forms for $\sim_k$. First, we show how to efficiently compute the attributes. Then we combine this computation with a single-pass deletion procedure; in particular, we do not have to successively re-compute the attributes after every single deletion. Finally, an easy observation shows that we only have to sort disjoint factors where the length of each factor is bounded by the size of the alphabet. Altogether, this yields an $\mathcal{O}(|A|\,n)$ algorithm for computing the shortlex normal form of an input word of length n over the alphabet A. Surprisingly, this bound also holds if k is part of the input.

4 Length reduction

In order to reduce words to shortlex normal form, we want to identify positions in the word which can be deleted without changing its $\sim_k$-class. The following proposition gives a sufficient condition for such deletions.

▶ **Proposition 2.** *Consider a word uav with $a \in A$ and $|ua| = i$. If the attribute (x_i, y_i) at position i satisfies $x_i + y_i > k+1$, then $uav \sim_k uv$.*

Proof. Let $w \preccurlyeq uav$ with $|w| \leq k$. Assume that $w \not\preccurlyeq uv$. Let $w = paq$ such that $p \preccurlyeq u$ and $q \preccurlyeq v$. Note that $pa \not\preccurlyeq u$ and $aq \not\preccurlyeq v$. If $|p| \geq x_i - 1$ and $|q| \geq y_i - 1$, then

$$k \geq |w| = |p| + 1 + |q| \geq (x_i - 1) + 1 + (y_i - 1) = x_i + y_i - 1 > k,$$

a contradiction. Therefore, we have either $|p| < x_i - 1$ or $|q| < y_i - 1$. By left-right symmetry, it suffices to consider $|p| < x_i - 1$. The word pa defines an X-ranker of length less than x_i which reaches position i. This is not possible by definition of x_i. Hence, $w \preccurlyeq uv$. Conversely, if $w \preccurlyeq uv$ for a word w, then obviously we have $w \preccurlyeq uav$. This shows $uav \sim_k uv$. ◀

▶ **Example 3.** Let $u = bacbaabada$ as in Example 1 and let $k = 3$. Note that the attributes (x_i, y_i) at positions $i \in \{5, 6\}$ satisfy the condition $x_i + y_i > k + 1$. By Proposition 2, deleting any of these positions yields a $\sim_k$-equivalent word. However, deleting both positions yields the word $bacbbada \not\sim_k u$ since $cab \preccurlyeq u$ and $cab \not\preccurlyeq bacbbada$.

Consider a position i with attribute (x_i, y_i) in a word u. Let

$$R_i^u = \{r \mid r \text{ is an X-ranker with } r(u) = i \text{ and } |r| = x_i\}\,.$$

We have $R_i^u \neq \emptyset$ by definition of x_i. We define a *canonical* X*-ranker* $r_i^u \in R_i^u$ by minimizing the reached positions, and the minimization procedure goes from right to left: Let $S_{x_i} = R_i^u$ and, inductively, we define S_j as a nonempty subset of S_{j+1} as follows. Let p_j be the minimal position in u visited by the prefixes s of length j of the rankers in S_{j+1}; then S_j contains all rankers in S_{j+1} such that their prefixes of length j visit the position p_j. Since the minimal positions (and their labels) in this process are unique, we end up with $|S_1| = 1$. Now, the ranker r_i^u is given by $S_1 = \{r_i^u\}$. By abuse of notation, we will continue to use the symbol r for arbitrary rankers while r_i^u denotes canonical rankers. The following example shows that minimizing from right to left (and not the other way round) is crucial.

▶ **Example 4.** Let $u = abcabcdaefccabc$. The attributes of the letters are as follows:

1 3	1 3	1 3	2 2	2 2	2 2	1 1	2 2	1 1	1 1	2 3	3 2	2 1	2 1	3 1
a	b	c	a	b	c	d	a	e	f	c	c	a	b	c
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The last c is at position 15 and its attribute is $(3, 1)$. It is easy to verify that $\mathsf{X}_e\mathsf{X}_a\mathsf{X}_c$ is an X-ranker of length 3 visiting position 15 and that there is no X-ranker of length 2 reaching this position. The unique Y-ranker of length 1 reaching position 15 is Y_c. We have

$$R^u_{15} = \{\mathsf{X}_d\mathsf{X}_b\mathsf{X}_c, \mathsf{X}_e\mathsf{X}_a\mathsf{X}_c, \mathsf{X}_e\mathsf{X}_b\mathsf{X}_c, \mathsf{X}_f\mathsf{X}_a\mathsf{X}_c, \mathsf{X}_f\mathsf{X}_b\mathsf{X}_c\}\,.$$

Using the above notation, it is easy to see that $S_3 = R^u_{15}$, $S_2 = \{\mathsf{X}_e\mathsf{X}_a\mathsf{X}_c, \mathsf{X}_f\mathsf{X}_a\mathsf{X}_c\}$, and $S_1 = \{\mathsf{X}_e\mathsf{X}_a\mathsf{X}_c\}$. All prefixes of length 2 of rankers in S_2 reach position $p_2 = 13$; the prefix of length 1 of the ranker in S_1 reaches position $p_1 = 9$. The ranker visiting positions 9, 13 and 15 (and no other positions) is $r^u_{15} = \mathsf{X}_e\mathsf{X}_a\mathsf{X}_c$, the unique ranker in S_1.

Also note that the minimal positions m_j visited by prefixes of length j of the rankers in R^u_{15} are $m_1 = 7$, $m_2 = 13$, and $m_3 = 15$; but there is no single ranker of length 3 visiting positions 7, 13, and 15.

While r^u_i is defined in some right-to-left manner, it still has an important left-to-right property when positions of the same label are considered.

▶ **Lemma 5.** *Let $i < j$ be two consecutive c-positions in a word u with attributes (x_i, y_i) and (x_j, y_j), respectively. If $x_j > x_i$, then $r^u_j = r^u_i\mathsf{X}_c$.*

Proof. Since no position ℓ with $i < \ell < j$ is labelled by c, we have $r^u_i\mathsf{X}_c(u) = j$. In particular, $x_j \leq x_i + 1$ and, hence, $x_j = x_i + 1$ by assumption. This shows $R^u_i\mathsf{X}_c \subseteq R^u_j$. Let $r^u_j = r\mathsf{X}_c$. We have $r(u) \geq r^u_i(u)$, since otherwise $r\mathsf{X}_c(u) \leq i < j$, a contradiction. The minimization in the definition of r^u_j now yields $r(u) = r^u_i(u)$. The remaining minimization steps in the definition of r^u_i and r^u_j consider the same rankers and thus the same positions. Hence, $r = r^u_i$. ◀

We now want to prove that the condition introduced in Proposition 2 always results in a shortest word within the corresponding $\sim_k$-class. To this end, we first need the following technical lemma and then prove the main theorem of this section.

▶ **Lemma 6.** *Let $u = a_1 \cdots a_n$ be a word and let $i < j$ be consecutive c-positions. Moreover, let the parameters (x_i, y_i) and (x_j, y_j) satisfy $x_i + y_i \leq k + 1$ and $x_j \leq k$, respectively. For every word v with $u \sim_k v$, we have $r^u_i(v) < r^u_j(v)$.*

Proof. We have $x_i \leq k$ and $x_j \leq k$. Therefore, both $r^u_i(v)$ and $r^u_j(v)$ are defined because this only depends on subwords of length at most k which are identical for u and v. Since $j = r^u_i\mathsf{X}_c(u)$, we have $x_j \leq x_i + 1$. If $x_j = x_i + 1$, then $r^u_j = r^u_i\mathsf{X}_c$ by Lemma 5 and hence $r^u_i(v) < r^u_j(v)$. Therefore, we can assume $x_j \leq x_i$. Suppose that $r^u_i(v) \geq r^u_j(v)$. Let $q\mathsf{Y}_c$ be a Y-ranker with $q\mathsf{Y}_c(u) = i$ and $|q\mathsf{Y}_c| = y_i$. Let w_i be the word which defines r^u_i, let w_j be the word which defines r^u_j, and let z be the word which defines q. We have:

$$\begin{aligned}
& w_i z \preccurlyeq u && \text{since } r^u_i(u) = i < q(u)\\
\Rightarrow\ & w_i z \preccurlyeq v && \text{since } |w_i z| = x_i + y_i - 1 \leq k \text{ and } u \sim_k v\\
\Rightarrow\ & w_j z \preccurlyeq v && \text{since } r^u_i(v) \geq r^u_j(v)\\
\Rightarrow\ & w_j z \preccurlyeq u && \text{since } |w_j z| = x_j + y_i - 1 \leq x_i + y_i - 1 \leq k\\
\Rightarrow\ & q(u) > j &&\\
\Rightarrow\ & q\mathsf{Y}_c(u) \geq j > i. &&
\end{aligned}$$

This contradicts $q\mathsf{Y}_c(u) = i$. Therefore, we have $r^u_i(v) < r^u_j(v)$. ◀

▶ **Theorem 7.** *If u is a word such that the attribute (x_i, y_i) of every position i satisfies $x_i + y_i \leq k + 1$, then u has minimal length within its $\sim_k$-class.*

Proof. Let v be any word satisfying $u \sim_k v$. Let ρ map the position j of u to the position $r_j^u(v)$ of v. Consider some letter c occurring in u. Then, by Lemma 6, the function ρ is order-preserving on the set of c-positions in u. In particular, the word v has at least as many occurrences of c as u. This holds for all letters c in u. Hence, $|u| \leq |v|$. ◀

▶ **Example 8.** Consider $u = bacbaabada$ from Example 1 and let $k = 3$. As explained in Example 3, we must not delete both position 5 and position 6. However, we can delete positions 5 and 8 to obtain a $\sim_k$-equivalent word with the following attributes:

$$\overset{1\,2}{b}\ \ \overset{1\,2}{a}\ \ \overset{1\,1}{c}\ \ \overset{2\,2}{b}\ \ \overset{2\,2}{a}\ \ \overset{3\,1}{b}\ \ \overset{1\,1}{d}\ \ \overset{2\,1}{a}$$

By Theorem 7, there is no shorter word in the same $\sim_k$-class.

5 Commutation

In the previous section, we described how to successively delete letters of a word in order to obtain a length-minimal $\sim_k$-equivalent word. It remains to show how to further transform a word of minimal length into shortlex normal form. In the first two statements, we give a sufficient condition which allows us to swap letters b and a while preserving the $\sim_k$-class.

▶ **Lemma 9.** *Consider two words $ubav$ and $uabv$ with $a, b \in A$. Let (x_ℓ, y_ℓ) denote the attribute of position ℓ in $ubav$, and let (x'_ℓ, y'_ℓ) denote the attribute of position ℓ in $uabv$. Suppose that $|ub| = i$ and that the attributes (x_i, y_i) and (x_{i+1}, y_{i+1}) satisfy $x_i = x_{i+1}$. Then all positions ℓ satisfy $x'_\ell = x_\ell$.*

Proof. Throughout this proof, we frequently rely on the following simple observation: If rs is a ranker of minimal length reaching position ℓ in a word w, then r is also of minimal length reaching position $r(w)$.

We can assume that $a \neq b$. It suffices to show that no ranker in R_{i+1}^{ubav} visits position i in $ubav$ and no ranker in R_{i+1}^{uabv} visits position i in $uabv$. This implies that for all $\ell \in \{1, \dots, n\}$, no ranker in R_ℓ^{ubav} or in R_ℓ^{uabv} visits both i and $i+1$ in the corresponding words and thus, we have $R_i^{ubav} = R_{i+1}^{uabv}$ and $R_{i+1}^{ubav} = R_i^{uabv}$ as well as $R_\ell^{ubav} = R_\ell^{uabv}$ for $\ell \notin \{i, i+1\}$. Note that all rankers in R_i^{ubav} and in R_{i+1}^{ubav} have length $x_i = x_{i+1}$.

Suppose, for the sake of contradiction, that a ranker $r \in R_{i+1}^{ubav}$ visits position i in $ubav$. Then, we can write $r = s\mathsf{X}_b\mathsf{X}_a$ with $s\mathsf{X}_b(ubav) = i$. Note that $|s\mathsf{X}_b| \geq x_i$ by the definition of x_i. Since $x_{i+1} = x_i \leq |s\mathsf{X}_b|$, there exists a ranker of length at most $|s\mathsf{X}_b| < |r|$ reaching position $i+1$ in $ubav$, contradicting the choice of r.

Suppose that a ranker $r \in R_{i+1}^{uabv}$ visits position i in $uabv$. Let $r = s\mathsf{X}_a\mathsf{X}_b$ with $s\mathsf{X}_a(uabv) = i$. Note that $s\mathsf{X}_a(ubav) = i+1$ and, since $x_{i+1} = x_i$, there exists a ranker $\hat{s}$ of length at most $|s\mathsf{X}_a|$ such that $\hat{s}(ubav) = i$. Now, $\hat{s}$ is a ranker of length $|\hat{s}| \leq |s\mathsf{X}_a| < |r|$ with $\hat{s}(uabv) = i+1$, a contradiction to $r \in R_{i+1}^{uabv}$. ◀

▶ **Proposition 10.** *Let $ubav$ be a word with $|ub| = i$ and attributes $(x_i, y_i) = (x_{i+1}, y_{i+1})$ satisfying $x_i + y_i = k + 1$. Then $ubav \sim_k uabv$.*

Proof. Suppose that there exists a word w with $|w| \leq k$ such that $w \preccurlyeq ubav$ but $w \not\preccurlyeq uav$ and $w \not\preccurlyeq ubv$. Then we can write $w = w_1 baw_2$ such that $w_1 b \preccurlyeq ub$, $w_1 b \not\preccurlyeq u$, $aw_2 \preccurlyeq av$, and $aw_2 \not\preccurlyeq v$. Thus, the word $w_1 b$ defines an X-ranker r with $r(ubav) = i$ and, similarly, aw_2

defines a Y-ranker s with $s(ubav) = i+1$. We see that $|r|+|s| = |w| \leq k$, but this contradicts $|r| + |s| \geq x_i + y_{i+1} = k + 1$. Therefore, every subword of $ubav$ of length at most k is also a subword of $uabv$.

By Lemma 9 and its left-right dual, the attributes of the positions i and $i + 1$ in $uabv$ are both identical to (x_i, y_i). Therefore, the same reasoning as above shows that every subword of $uabv$ of length at most k is also a subword of $ubav$. This shows $ubav \sim_k uabv$. ◀

▶ **Example 11.** Let us reconsider the length-minimal word $u = bacbabda$ from Example 8 and let again $k = 3$. The attributes are as follows:

$$\begin{array}{cccccccc} 12 & 12 & 11 & 22 & 22 & 31 & 11 & 21 \\ b & a & c & b & a & b & d & a \end{array}$$

The attributes (x_4, y_4) and (x_5, y_5) at positions 4 and 5 satisfy $x_4 = x_5$, $y_4 = y_5$ and $x_4 + y_4 = k + 1$. By Proposition 10, we obtain $bacabbda \sim_k u$. Note that the attributes (x_1, y_1) and (x_2, y_2) at the first two positions satisfy $x_1 = x_2$, $y_1 = y_2$ but $x_1 + x_2 < k + 1$. And, in fact, $abcbabda \not\sim_k u$ since $abc \preccurlyeq abcbabda$ but $abc \not\preccurlyeq u$.

It remains to show that repeated application of the commutation rule described in Proposition 10 actually suffices to obtain the lexicographically smallest representative of a $\sim_k$-class. The next lemma shows, using canonical rankers, that indeed all length-minimal representatives of a $\sim_k$-class can be transformed into one another using this commutation rule.

▶ **Lemma 12.** *Let $u \sim_k v$ such that both words u and v have minimal length in their $\sim_k$-class. Let (x_ℓ, y_ℓ) denote the attribute of position ℓ of u. Consider two positions $i < j$ of u. If either $(x_i, y_i) \neq (x_j, y_j)$ or $x_i + y_i < k + 1$, then $r_i^u(v) < r_j^u(v)$.*

Proof. If i and j have the same label, then the claim follows from Lemma 6. In the remainder of this proof, let their labels be different. In particular, we cannot have $r_i^u(v) = r_j^u(v)$. Suppose $(x_i, y_i) \neq (x_j, y_j)$ or $x_i + y_i < k + 1$. If $x_i + y_j \geq k + 1$ and $x_j + y_i \geq k + 1$, then, by minimality and Proposition 2, we have $x_i + y_i = k + 1$ and $x_j + y_j = k + 1$. This yields $x_i + y_j = k + 1$ and $x_j + y_i = k + 1$. Thus, $x_i = x_i + x_j + y_j - k - 1 = x_j$ and, similarly, $y_i = y_i + x_j + y_j - k - 1 = y_j$; this shows $(x_i, y_i) = (x_j, y_j)$, a contradiction. Therefore, we have either $x_i + y_j \leq k$ or $x_j + y_i \leq k$.

Let p_i and p_j be the words defining the rankers r_i^u and r_j^u, respectively. Symmetrically to the definition of the canonical X-ranker, we could also define canonical Y-rankers s_i^u and s_j^u such that $s_i^u(u) = i$, $|s_i^u| = y_i$, $s_j^u(u) = j$, and $\left|s_j^u\right| = y_j$. If the label c of u at position i is the ℓ-th occurrence of the letter c in u, then, by Lemma 6, both r_i^u and s_i^u end up at the position with the ℓ-th occurrence of the letter c in v. This shows $r_i^u(v) = s_i^u(v)$. Similarly, we see that $r_j^u(v) = s_j^u(v)$. Let q_i and q_j be the words defining the rankers s_i^u and s_j^u, respectively.

First, let $x_i + y_j \leq k$. Then $p_i q_j \preccurlyeq u$ yields $p_i q_j \preccurlyeq v$ since $u \sim_k v$ and $|p_i q_j| = x_i + y_j \leq k$. This shows $r_i^u(v) < s_j^u(v) = r_j^u(v)$, as desired. Let now $x_j + y_i \leq k$ and assume $r_i^u(v) > r_j^u(v)$. Then $p_j q_i \preccurlyeq v$ yields $p_j q_i \preccurlyeq u$ and, thus, $j = r_j^u(u) < s_i^u(u) = i$. This is a contradiction; hence, $r_i^u(v) < r_j^u(v)$. ◀

Using the previous lemma, we can finally show that iterating the commutation procedure from Lemma 9 and Proposition 10 yields the desired shortlex normal form.

▶ **Theorem 13.** *Let $u = a_1 \cdots a_n$ with $a_i \in A$ be a length-minimal word within its $\sim_k$-class. Suppose that the attributes (x_i, y_i) for all positions $i < n$ satisfy the following implication:*

$$\text{If } (x_i, y_i) = (x_{i+1}, y_{i+1}) \text{ and } x_i + y_i = k + 1, \text{ then } a_i \leq a_{i+1}. \tag{1}$$

Then u is the shortlex normal form of its $\sim_k$-class.

Proof. Let v be the shortlex normal form of the $\sim_k$-class of u. We want to show that $u = v$. Let ρ map position i of u to position $r_i^u(v)$ of v. As we have seen in the proof of Theorem 7, the function ρ is bijective. It remains to show that ρ is order-preserving. By contradiction, assume that there are positions i and j of u with $i < j$ such that $\rho(i) > \rho(j)$; let i be minimal with this property and let $i = \rho(j)$, *i.e.*, we choose j to be the preimage of position i in v. We already know that $\rho(i) < \rho(j)$ in all of the following cases:

- $a_i = a_j$ (by Lemma 6),
- $(x_i, y_i) \neq (x_j, y_j)$ (by Lemma 12),
- $x_i + y_i < k + 1$ (again by Lemma 12).

Therefore, the only remaining case is $a_i \neq a_j$, $(x_i, y_i) = (x_j, y_j)$ and $x_i + y_i = k + 1$. First, suppose that $(x_i, y_i) = (x_\ell, y_\ell)$ for all $\ell \in \{i, \ldots, j\}$. Then, by the implication in Equation (1), we have $a_i \leq \cdots \leq a_j$. Since $a_i \neq a_j$, we have $a_i < a_j$. Now, u has the prefix $a_1 \cdots a_i$ and v has the prefix $a_1 \cdots a_{i-1} a_j$. In particular, u is lexicographically smaller than v; this is a contradiction. Next, suppose that there exists a position $\ell \in \{i, \ldots, j\}$ with $(x_i, y_i) \neq (x_\ell, y_\ell)$. Note that $i < \ell < j$. By Lemma 12, we have $\rho(i) < \rho(\ell)$ and $\rho(\ell) < \rho(j)$. In particular, we have $\rho(i) < \rho(j)$ in contradiction to our assumption. Altogether, this shows that $i < j$ and $\rho(i) > \rho(j)$ is not possible, *i.e.*, ρ is order-preserving. Hence, $u = v$. ◀

We summarize our knowledge on shortest elements of a $\sim_k$-class as follows. A word u has minimal length within its $\sim_k$-class if and only if all attributes (x_i, y_i) satisfy $x_i + y_i \leq k + 1$. The canonical rankers define a bijective mapping between any two shortest words u and v of a common $\sim_k$-class. This map preserves the labels and the attributes. It is almost order preserving, with the sole exception that $i < j$ could lead to $r_i^u(v) > r_j^u(v)$ whenever the attributes in u satisfy both $x_i + y_i = k + 1$ and $(x_i, y_i) = (x_\ell, y_\ell)$ for all $\ell \in \{i, \ldots, j\}$.

6 Computing shortlex normal forms

The results from the previous sections immediately lead to the following algorithm for computing shortlex normal forms. First, we successively delete single letters of the input word until the length is minimal. Let $a_1 \cdots a_n$ be the resulting word. In the second step, we lexicographically sort maximal factors $a_i \cdots a_j$ with attributes $(x_i, y_i) = \cdots = (x_j, y_j)$ and $x_i + y_i = k + 1$. We now improve the first step of this algorithm.

Algorithm 1 Computing the x-coordinates of $a_1 \cdots a_n$.

```
1: for all $a \in A$ do $n_a \leftarrow 1$
2: for $i \leftarrow 1, \ldots, n$ do
3:     suppose $a_i = c$
4:     $x_i \leftarrow n_c$
5:     $n_c \leftarrow n_c + 1$
6:     for all $a \in A$ do $n_a \leftarrow \min(n_a, n_c)$
```

The following lemma proves the correctness of Algorithm 1. Its running time is in $\mathcal{O}(|A|\, n)$ since there are n iterations of the main loop, and each iteration updates $|A|$ counters.

▶ **Lemma 14.** *Algorithm 1 computes the correct x-coordinates of the attributes of $a_1 \cdots a_n$.*

Proof. The algorithm reads the input word from left to right, letter by letter. In each step it updates some of its counters n_a. The semantics of the counters n_a is as follows: if the next letter a_i is c, then x_i is n_c. This invariant is true after the initialization in the first line.

Algorithm 2 Computing the y-coordinates of $a_1 \cdots a_n$ plus deletion.

```
1: for all $a \in A$ do $n_a \leftarrow 1$
2: for $i \leftarrow n, \ldots, 1$ do
3:     suppose $a_i = c$
4:     if $x_i + n_c \leq k + 1$ then
5:         $y_i \leftarrow n_c$
6:         $n_c \leftarrow n_c + 1$
7:         for all $a \in A$ do $n_a \leftarrow \min(n_a, n_c)$
8:     else
9:         position $i$ is marked for deletion
```

Suppose that we start an iteration of the loop at letter $a_i = c$. Then the invariant tells us that $x_i = n_c$. If a_{i+1} were c, then one more step X_c would be needed for a ranker to reach position $i + 1$, hence $n_c \leftarrow n_c + 1$. If a_{i+1} were some letter $a \neq c$, then we could either use the ranker corresponding to the old value n_a or we could use the ranker going to position i and from there do an X_a-modality; the latter would yield a ranker whose length is the new value of n_c. We choose the shorter of these two options. Since all counter values were correct before reading position i, there is no other counter n_a which needs to be updated before proceeding with position $i + 1$. ◀

With Algorithm 2, we give a procedure for computing the y-coordinates of $a_1 \cdots a_n$ similar to Algorithm 1, but with the modification that we mark some letters for deletion. The positions marked for deletion depend on the number k in Simon's congruence $\sim_k$. The computed y-coordinates are those where all marked letters are actually deleted. We assume that the x-coordinates of the input word are already known.

The algorithm correctly computes the y-coordinates of the word where all marked letters are deleted. This follows from the left-right dual of Lemma 14 and the fact that the counters remain unchanged if a position is marked for deletion.

▶ **Lemma 15.** *Let u be the input for Algorithm 2 and let v be the word with all marked letters removed. Then $u \sim_k v$.*

Proof. Whenever a position i with label c is marked for deletion, the value x_i is correct since no letter to the left of position i is marked for deletion. The counter n_c would be the correct y-coordinate for position i if we deleted all positions which have been marked so far. By Proposition 2 we know that each deletion preserves the $\sim_k$-class. ◀

It remains to show that the x-coordinates are still correct for the resulting word in which all marked letters are deleted.

▶ **Lemma 16.** *Consider a word $u = a_1 \cdots a_n$ with x-coordinate x_ℓ at position ℓ. Let i be the maximal position of u such that $x_i + y_i > k + 1$ and let $v = a_1 \cdots a_{i-1} a_{i+1} \cdots a_n$. The x-coordinate of position ℓ of v is denoted by x'_ℓ. Then, for all $j \in \{i + 1, \ldots, n\}$, we have $x'_{j-1} = x_j$.*

Proof. It suffices to prove the statement $x'_{j-1} = x_j$ for all positions j of u reachable by a ranker of the form $r\mathsf{X}_c$ with $r(u) = i$ and $c \in A$. By contradiction, suppose that there exists some position $j = r\mathsf{X}_c(u)$ with $x'_{j-1} \neq x_j$ where $r(u) = i$ and $c \in A$; we choose $c \in A$ such that j is minimal with this property. Let $b = a_i$. We have to distinguish two cases.

First suppose that there is no b-position f with $i < f < j$ in u. The ranker r_j^u has to visit position i in u; otherwise $r_j^u(v) = j-1$ and $r_{j-1}^v(u) = j$, a contradiction to $x'_{j-1} \neq x_j$. This implies $x_i < x_j$. Moreover, position i is reachable from j in u with a single Y_b-modality, and hence, we have $x_i + y_i \leq (x_j - 1) + (y_j + 1) \leq k+1$. This contradicts the choice of i.

Next, let f be the minimal b-position with $i < f < j$. In particular, we have $b \neq c$ because $j \neq f$ is the smallest c-position of u greater than i. Let $r\mathsf{X}_c$ be an X-ranker of length x_j such that $r\mathsf{X}_c(u) = j$. If $r(u) = i$, then $r(v) = f-1$ and hence $r\mathsf{X}_c(v) = j-1$. If $r(u) < i$, then the ranker $r\mathsf{X}_c$ does not visit the position i in u and we have $r\mathsf{X}_c(v) = j-1$. Finally, if $r(u) > i$, then (by choice of c) the position $r(u) < j$ keeps its x-coordinate. In other words, there exists an X-ranker r' with $|r| = |r'|$ and $r'(v) = r(u) - 1$. It follows that $r'\mathsf{X}_c(v) = j-1$. Therefore, in any case, there exists a ranker s of length at most x_j such that $s(v) = j-1$. This shows $x'_{j-1} \leq x_j$, and together with $x'_{j-1} \neq x_j$ we obtain $x'_{j-1} < x_j$.

Consider an X-ranker $s\mathsf{X}_c$ of length $x'_{j-1} < x_j$ with $s\mathsf{X}_c(v) = j-1$. We are still in the situation that there exists a b-position f in u with $i < f < j$. We cannot have $s(v) < i$ since otherwise $s(u) = s(v)$ and, thus, $s\mathsf{X}_c(u) = j$; the latter uses the fact that $b \neq c$. Let now $s(v) \geq i$ and write $s = t\mathsf{X}_d$. We have $t(v) < i$ since otherwise $t\mathsf{X}_c$ would be a shorter X-ranker with $t\mathsf{X}_c(v) = j-1$. We have $d = b$: if $d \neq b$, then $s(v) = s(u) - 1$ and $s\mathsf{X}_c(u) = j$; this would show $x'_{j-1} \geq x_j$, thereby contradicting $x'_{j-1} < x_j$. It follows that $s(u) = i$ and $s\mathsf{X}_c(u) = j$. As before, this is a contradiction. This completes the proof that $x'_{j-1} = x_j$. ◀

▶ **Example 17.** Let $u = bacbaabada$ be the word from Example 1 and let $k = 3$. Suppose that the alphabet $A = \{a, b, c, d\}$ is ordered by $a < b < c < d$. The attributes of u are as follows:

$$\begin{array}{cccccccccc} 12 & 12 & 11 & 22 & 23 & 32 & 31 & 42 & 11 & 21 \\ b & a & c & b & a & a & b & a & d & a \end{array}$$

Note that each of the attributes (x_i, y_i) at positions $i \in \{5, 6, 8\}$ satisfies the condition $x_i + y_i > k+1$. As seen in Example 3 we must not delete all these positions. The algorithm only marks positions 6 and 8 for deletion and takes these deletions into account when computing the y-coordinates of the remaining letters:

$$\begin{array}{cccccccccc} 12 & 12 & 11 & 22 & 22 & 3 & 31 & 4 & 11 & 21 \\ b & a & c & b & a & a & b & a & d & a \end{array}$$

The letters are now sorted as in Example 11 and the resulting normal form is $bacabbda$.

The following lemma allows us to improve the estimated time for the sorting step of the main algorithm by showing that any sequence of letters which needs to be sorted contains every letter at most once.

▶ **Lemma 18.** *Consider a word $uaav$ with $a \in A$ and $|ua| = i$. Then $x_i \neq x_{i+1}$ and $y_i \neq y_{i+1}$.*

Proof. Suppose $x_i = x_{i+1}$. Let r_i and r_{i+1} be X-rankers with $r_i(uaav) = i$, $|r_i| = x_i$, $r_{i+1}(uaav) = i+1$, and $|r_{i+1}| = x_{i+1} = x_i$. Let $r_{i+1} = s\mathsf{X}_a$. If $s(uaav) < i$, then $i + 1 = r_{i+1}(uaav) = s\mathsf{X}_a(uaav) \leq i$. If $s(uaav) = i$, then $|s| = x_i - 1 < x_i = |r_i|$ contradicts the definition of x_i. Therefore, we cannot have $x_i = x_{i+1}$. Symmetrically, we cannot have $y_i = y_{i+1}$. ◀

We are now able to state our main result.

▶ **Theorem 19.** *One can compute the shortlex normal form of a word w of length n, including all attributes of the normal form, with $\mathcal{O}(|A|\, n)$ arithmetic operations and with bit complexity $\mathcal{O}(|A|\, n \log n)$. Alternatively, the computation can be done in deterministic space $\mathcal{O}(|A| \log n)$.*

Proof. The attributes of the normal form can be computed as described in Algorithms 1 and 2. The normal form itself is obtained by filtering out all positions i where the corresponding attribute (x_i, y_i) satisfies $x_i + y_i \leq k + 1$ and by sorting blocks of letters with the same attributes satisfying $x_i + y_i = k + 1$. By Lemma 18, the sorting step can be performed by reading each such block of letters, storing all letters appearing in the block and only outputting all these letters in sorted order once the next block is reached.

If we assume that the comparison of two letters and the modification of the counters is possible in constant time, then running Algorithm 2 on the output of Algorithm 1 takes $\mathcal{O}(|A|\, n)$ steps for input words of length n over alphabet A: for each position of the input word, we need to update $|A|$ counters. Over a fixed alphabet, the resulting algorithm runs in linear time – even if k is part of the input. We could bound all arithmetic operations by $k + 2$, *i.e.*, by replacing the usual addition by $n \oplus m = \min(k + 2, n + m)$. This way, each counter and all results of arithmetic operations would require only $\mathcal{O}(\log k) \subseteq \mathcal{O}(\log n)$ bits. Similarly, $\mathcal{O}(\log |A|) \subseteq \mathcal{O}(\log n)$ bits are sufficient to encode the letters. This leads to a bit complexity of $\mathcal{O}(|A|\, n \log n)$. Note that if $k > n$, then the $\sim_k$-class of the input is a singleton and we can immediately output the input without any further computations. If $|A| > n$, then we could replace A by the letters which occur in the input word.

For the $\mathcal{O}(|A| \log n)$ space algorithm, one can again use Algorithms 1 and 2 to compute the attributes of each position. To compute the shortlex normal form, we do not store all the attributes but use the standard recomputation technique to decide whether a letter gets deleted. The sorting step can be implemented by repeatedly scanning each block of positions with common attributes (x, y) satisfying $x + y = k + 1$. A single scan checks, for a fixed letter $a \in A$, whether a occurs in the block. This is repeated for every $a \in A$ in ascending order. The attributes of the currently investigated block and the current letter a can be stored in space $\mathcal{O}(\log n)$. ◀

7 Summary and Outlook

We considered Simon's congruence $\sim_k$ for piecewise testable languages. The main contribution of this paper is an $\mathcal{O}(|A|\, n)$ algorithm for computing the shortlex normal form of a word of length n within its $\sim_k$-class; surprisingly, this bound also holds if k is part of the input. The algorithm can be adapted to work in deterministic logarithmic space over a fixed alphabet. As a consequence, on input u, v, k, one can test in time $\mathcal{O}(|A|\, |uv|)$ whether $u \sim_k v$ holds. The main tool are the minimal lengths of X-rankers and Y-rankers reaching any position of a word. The key ingredient in the proofs are the so-called canonical rankers.

It would be interesting to see whether the space complexity for an arbitrary alphabet can be further improved from $\mathcal{O}(|A| \log n)$ to nondeterministic log-space or even deterministic log-space if the alphabet A is part of the input. In addition, we still lack corresponding lower bounds for the computation of shortlex normal forms and for the test of whether $u \sim_k v$ holds.

References

1 Sung Cho and Dung T. Huynh. Finite automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88:96–116, 1991.

2 Jie Fu, Jeffrey Heinz, and Herbert G. Tanner. An algebraic characterization of strictly piecewise languages. In Mitsunori Ogihara and Jun Tarui, editors, *Theory and Applications of Models of Computation*, volume 6648 of *LNCS*, pages 252–263, Berlin, Heidelberg, 2011. Springer.

3 Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society. Third Series*, 2:326–336, 1952.

4 Stepan Holub, Tomás Masopust, and Michaël Thomazo. Alternating towers and piecewise testable separators. *CoRR*, abs/1409.3943, 2014.

5 J. E. Hopcroft and R. Karp. A linear algorithm for testing equivalence of finite automata. Technical Report 71-114, Dept. of Computer Science, Cornell U, December 1971.

6 P. Karandikar, M. Kufleitner, and Ph. Schnoebelen. On the index of Simon's congruence for piecewise testability. *Information Processing Letters*, 115(4):515–519, 2015.

7 Kamilla Kátai-Urbán, Péter Pál Pach, Gabriella Pluhár, András Pongrácz, and Csaba Szabó. On the word problem for syntactic monoids of piecewise testable languages. In *Semigroup Forum*, volume 84, pages 323–332. Springer, 2012.

8 O. Klíma, M. Kunc, and L. Polák. Deciding k-piecewise testability, submitted.

9 Ondřej Klíma and Libor Polák. Alternative automata characterization of piecewise testable languages. In Marie-Pierre Béal and Olivier Carton, editors, *DLT 2013, Proceedings*, volume 7907 of *Lecture Notes in Computer Science*, pages 289–300. Springer, 2013.

10 Leonid Aryeh Kontorovich, Corinna Cortes, and Mehryar Mohri. Kernel methods for learning languages. *Theoretical Computer Science*, 405(3):223–236, 2008.

11 Tomás Masopust and Michaël Thomazo. On the complexity of k-piecewise testability and the depth of automata. In *DLT 2015, Proceedings*, volume 9168 of *Lecture Notes in Computer Science*, pages 364–376. Springer, 2015.

12 Péter Pál Pach. Solving equations under Simon's congruence. In *JHSDM 2015, Proceedings*, pages 201–206, 2015.

13 Péter Pál Pach. Normal forms under Simon's congruence. *Semigroup Forum*, Dec 2017.

14 James Rogers, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Visscher, David Wellcome, and Sean Wibel. On languages piecewise testable in the strict sense. In Christian Ebert, Gerhard Jäger, and Jens Michaelis, editors, *The Mathematics of Language*, pages 255–265, Berlin, Heidelberg, 2010. Springer.

15 José Ruiz and Pedro García. Learning k-piecewise testable languages from positive data. In Laurent Miclet and Colin de la Higuera, editors, *Grammatical Interference: Learning Syntax from Sentences*, pages 203–210, Berlin, Heidelberg, 1996. Springer.

16 Thomas Schwentick, Denis Thérien, and Heribert Vollmer. Partially-ordered two-way automata: A new characterization of DA. In *DLT 2001, Proceedings*, volume 2295 of *LNCS*, pages 239–250. Springer, 2002.

17 Imre Simon. *Hierarchies of events with dot-depth one.* PhD thesis, University of Waterloo, 1972.

18 Imre Simon. Piecewise testable events. In *Autom. Theor. Form. Lang., 2nd GI Conf.*, volume 33 of *LNCS*, pages 214–222. Springer, 1975.

19 Jacques Stern. Characterization of some classes of regular events. *Theor. Comput. Sci.*, 35:17–42, 1985.

20 Wolfgang Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, 25:360–376, 1982.

21 A. N. Trahtman. Piecewise and local threshold testability of DFA. In Rusins Freivalds, editor, *FCT 2001, Proceedings*, volume 2138 of *LNCS*, pages 347–358. Springer, 2001.

22 Philipp Weis and Neil Immerman. Structure theorem and strict alternation hierarchy for FO^2 on words. *Log. Methods Comput. Sci.*, 5(3):1–23, 2009.