KYBERNETIKA — VOLUME *49* (2013), NUMBER 2, PAGES 216–223

COMPLEXITY OF TESTING MORPHIC PRIMITIVITY

Štěpán Holub and Vojtěch Matocha

We analyze an algorithm that decides whether a given word is a fixed point of a nontrivial morphism. We show that it can be implemented to have complexity in $\mathcal{O}(m \cdot n)$, where n is the length of the word and m the size of the alphabet.

Keywords: fixed points, morphic primitivity, complexity

Classification: 68R15

1. INTRODUCTION

The word $u = abaaba$ satisfies $f(u) = u$ where f maps b to aba and cancels a. Such words, which are fixed points of a nontrivial morphism, are called *morphically imprimitive.* On the other hand, the word $u' = abba$ can be easily verified to be *morphically primitive*, which means that the only morphism satisfying $f(u') = u'$ defined on $\{a, b\}^*$ is the identity.

Fixed points of word morphisms and morphically (im)primitive words are studied in [2, 3, 6, 5]. In [4], the first polynomial algorithm is presented (called MorphicFactorization) that decides whether a given word w is morphically primitive. Moreover, given the input word w, it finds a corresponding morphism satisfying $f(w) = w$ with minimal number of letters mapped to a nonempty word (that is, not cancelled).

The complexity of MorphicFactorization is estimated as $\mathcal{O}(m + \log n) \cdot n$ in [4]. Here we provide a more detailed analysis of the algorithm and improve the estimate to $\mathcal{O}(|E| \cdot n)$, where E is the set of those letters x for which $f(x)$ is nonempty.

2. DEFINITIONS

Let $\operatorname{alph}(w)$ denote the set of letters occurring in w and $|w|$ the length of w. For a set $S \subset \operatorname{alph}(w)$, denote by $|w|_S$ the number of all occurrences of letters from S in w; we shorten $|w|_{\{a\}}$ as $|w|_a$.

Each morphism f, satisfying $f(w) = w$, induces a factorization of w, called a *morphic factorization.* The morphic factorization consists of a set E and a sequence $(w_1, w_2, \ldots, w_k)$ such that

- $w = w_1 w_2 \cdots w_k$,
- $|w_i|_E = 1$ for each $i = 1, 2, \ldots, k$, and
- if $|w_i|_e = |w_j|_e = 1$ for some $e \in E$, then $w_i = w_j$.

It is shown in [2] that we can suppose, without loss of generality, that f is idempotent, that is, $f(a) = f(f(a))$ for each $a \in \mathrm{alph}(w)$. (It is enough to iterate a general f sufficient number of times in order to obtain its idempotent version.) Throughout the paper, we shall therefore assume that f is idempotent. The relation between f and the corresponding morphic factorization is then as follows: for each $i = 1, \ldots, k$, we have $w_i = f(e)$, where e is the unique letter from E occurring in w_i, and $f(a) = \varepsilon$ if $a \notin E$ (where ε denotes the empty word). Letters in E are called *expanding.* We say that E is a *minimal set of expanding letters* if no proper subset of E is the set of expanding letters for a morphism f' satisfying $f'(w) = w$. In [4], it is shown that all minimal sets of expanding letters have the same cardinality.

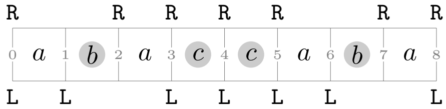
Denote the ith letter of w by $w[i]$ and write $w[i \ldots j]$, with $i \leq j$, to denote the factor $w[i]w[i+1] \cdots w[j]$ of w. We will also work with the set $\mathcal{C}_w$ of *cuts*, that is, of borders between two consecutive letters (plus the beginning and the end of w). A word w has $|w| + 1$ cuts and we represent them by integers $0, 1, \ldots, |w|$. The cut k is the border following the prefix of length k. Note that cuts i, j delimit the factor $w[i+1 \ldots j]$.

Given a word w and a morphism f such that $f(w) = w$, we say that a cut k is a *left* cut if it lies in the image of an expanding letter on its left side. More formally, the cut k is a left cut if $|f(w[1 \ldots k])| \leq k$. Similarly, we say that k is a *right* cut if $|f(w[1 \ldots k])| \geq k$. Note that inequalities are not strict, therefore a cut k can be both left and right, which happens if and only if $|f(w[1 \ldots k])| = k$. Note that cuts that are both left and right define the morphic factorization of w induced by f. We say that $w[i, j]$ is a *stretch factor* if i is a left cut and j is a right cut.

For example, the word *abaccaba* is a fixed point of the morphism

$$f : a \mapsto \varepsilon, \quad b \mapsto aba, \quad c \mapsto c.$$

Left and right cuts are given by the following figure:



An important and natural notion is the *neighborhood* of a letter a in w, denoted by $\mathbf{n}_a$. The neighborhood of a is the longest extension of a that is possible for all occurrences of a in w. More precisely, let $a\mathbf{r}_a$ be the longest common prefix of all suffixes of w starting with a. Similarly, let $\mathbf{l}_a a$ be the longest common suffix of all prefixes of w ending with a. Then $\mathbf{n}_a = \mathbf{l}_a a \mathbf{r}_a$. It is easy to see that the word $\mathbf{n}_a$ contains exactly one occurrence of a. (See Example 3.1 below for an illustration.)

Letters with minimal frequency in w that occur in a given factor play a special role in the algorithm. Therefore, we define

$$\alpha(i, j) = \min\{k \mid i < k \leq j,\ |w|_{w[k]} \leq |w|_{w[k']} \text{ for all } i < k' \leq j\}.$$

In other words, $\alpha(i, j)$ is the leftmost position of a least frequent letter in $w[i, j]$. Note that "least frequent" is measured with respect to whole w, not just with respect to $w[i, j]$.

3. DESCRIPTION OF THE ALGORITHM

The algorithm MorphicFactorization is based on the following characterization of minimal expanding sets (for proofs and more details see [4]):

Let E be a minimal set of letters, and L, R minimal sets of cuts satisfying the following *stability conditions*:

A. $\{0, |w|\} \subseteq L$, $\{0, |w|\} \subseteq R$.

B. Let $w[k] = w[k'] = a$ with $a \in E$, Then

   (a) $k - 1 \in L$ and $k \in R$;

   (b) $k + |\mathbf{r}_a| \in L$ and $k - |\mathbf{l}_a| - 1 \in R$;

   (c) for each $-|\mathbf{l}_a| - 1 \leq m \leq \mathbf{r}_a$ we have that

   - $k + m \in L$ if and only if $k' + m \in L$, and
   - $k + m \in R$ if and only if $k' + m \in R$.

C. If $i \in L$, $j \in R$ with $i < j$, then $w[\alpha(i,j)] \in E$.

Then E is a minimal set of expanding letters. For $a \in E$, the image $f(a)$ is defined as

$$f(a) = \textsc{Image}(a, E, L, R) := w[k - i, k + j],$$

where $w[k] = a$; $i \geq 0$ is the smallest integer such that $k - i - 1 \in R$; and $j \geq 0$ is the largest integer such that

- $k + j \in R$, and
- $k + j' \notin L$ holds for each $j' \leq j$.

Stability conditions guarantee that $f(a)$ is well defined, in particular, it is independent of the choice of k, and that the resulting morphism satisfies $f(w) = w$. Moreover, all cuts in R are right cuts of the factorization, and cuts in L are left cuts. In view of the fact that sets E, L and R represent expanding letters, left cuts and right cuts respectively, stability conditions can be rephrased informally as follows:

A. the extremal cuts are both left and right;

B. (a) an expanding letter is delimited by a left and a right cut;

   (b) neighborhood of an expanding letter is delimited by a right and a left cut (the left border is a right cut and vice versa);

   (c) neighborhoods of expanding letters are synchronized with respect to left and right cuts;

C. the leftmost least frequent letter in each stretch factor is expanding.

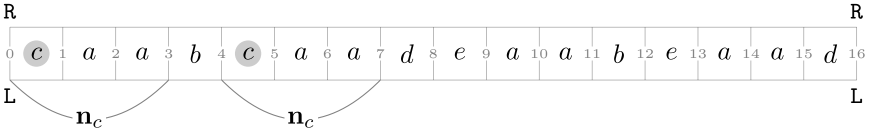
The core procedure of the algorithm MorphicFactorization consists in the construction of sets E, L and R satisfying stability conditions. Given a subset $\mathtt{E}$ of $\mathrm{alph}(w)$, we define subsets $\mathtt{L}(\mathtt{E})$ and $\mathtt{R}(\mathtt{E})$ of $\mathcal{C}_w$ as the smallest sets satisfying stability conditions (A) and (B). Similarly, for two subsets $\mathtt{L}$ and $\mathtt{R}$ of $\mathcal{C}_w$, we define $\mathtt{E}(\mathtt{L},\mathtt{R})$ as the smallest subset of $\mathrm{alph}(w)$ satisfying the stability condition (C). We are looking for a set $\mathtt{E}$ satisfying $\mathtt{E} = \mathtt{E}(\mathtt{L}(\mathtt{E}),\mathtt{R}(\mathtt{E}))$. If $\mathtt{E} \neq \mathtt{E}(\mathtt{L},\mathtt{R})$, then there exist cuts i, j violating the condition (C), that is, the letter $w[\alpha(i,j)]$ is not an element of E. Denote such a letter by $\mathtt{New}(\mathtt{E},\mathtt{L},\mathtt{R})$. The algorithm is now described by the following simple pseudocode.

```
MorphicFactorization($w$)
E ← ∅; L ← {0, |w|}; R ← {0, |w|};
while E ≠ E(L, R)
   do E ← E ∪ {New(E, L, R)};
      L ← L(E); R ← R(E);
for each $a \in \mathrm{alph}(w)$
   do if $a \in$ E
         then $f(a)$ ← Image($a$, E, L, R);
         else $f(a) \leftarrow \varepsilon$;
return $f$.
```

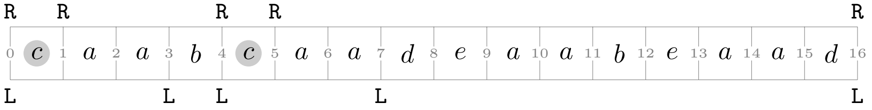
Several examples illustrating the work of the algorithm can be found in [4]. It can be also tested and visualized on [7]. Here we add one more example. It can be also understood as a replacement of Example 7 in [4], which is mistaken.

Example 3.1. Consider $w = caabcaadeaabeaad$, where $\mathbf{n}_a = a$, $\mathbf{n}_b = aab$, $\mathbf{n}_c = caa$, $\mathbf{n}_d = aad$ and $\mathbf{n}_e = eaa$. Let us follow the run of the algorithm. At the beginning we set $\mathtt{E} = \emptyset$ and $\mathtt{L} = \mathtt{R} = \{0, 16\}$. Rounds of the **while** loop yield the following:

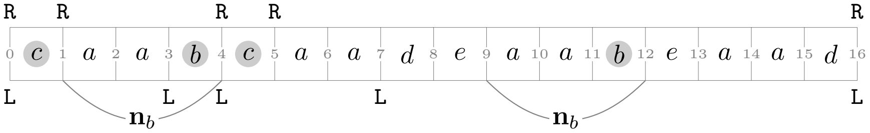
Round 1.
- (C) implies $\mathtt{New}(\mathtt{E},\mathtt{L},\mathtt{R}) = w[\alpha(0,16)] = w[1] = c$;


R
R
0 c 1 a 2 a 3 b 4 c 5 a 6 a 7 d 8 e 9 a 10 a 11 b 12 e 13 a 14 a 15 d 16
L
L
$\mathbf{n}_c$
$\mathbf{n}_c$

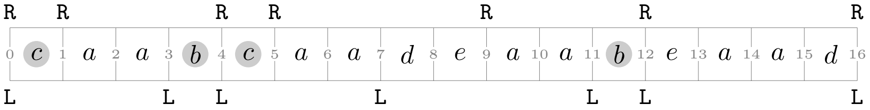
- since $c \in \mathtt{E}$, (Ba) implies $0, 4 \in \mathtt{L}$, $1, 5 \in \mathtt{R}$, and (Bb) implies $3, 7 \in \mathtt{L}$, $0, 4 \in \mathtt{R}$; the condition (Bc) is satisfied.


R R R R R
0 c 1 a 2 a 3 b 4 c 5 a 6 a 7 d 8 e 9 a 10 a 11 b 12 e 13 a 14 a 15 d 16
L L L L L

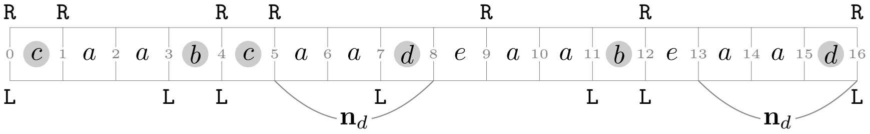
Round 2. • (C) implies $\texttt{New}(\texttt{E},\texttt{L},\texttt{R}) = w[\alpha(3,4)] = w[4] = b$;


R R R R R
0 c 1 a 2 a 3 b 4 c 5 a 6 a 7 d 8 e 9 a 10 a 11 b 12 e 13 a 14 a 15 d 16
L L L L L
$\mathbf{n}_b$
$\mathbf{n}_b$

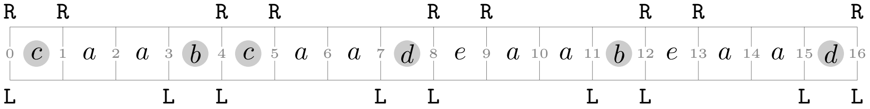
- since $b \in \texttt{E}$, (Ba) implies $3, 11 \in \texttt{L}$, $4, 12 \in \texttt{R}$, and (Bb) implies $4, 12 \in \texttt{L}$, $1, 9 \in \texttt{R}$; the condition (Bc) is satisfied.


R R R R R R R
0 c 1 a 2 a 3 b 4 c 5 a 6 a 7 d 8 e 9 a 10 a 11 b 12 e 13 a 14 a 15 d 16
L L L L L L L

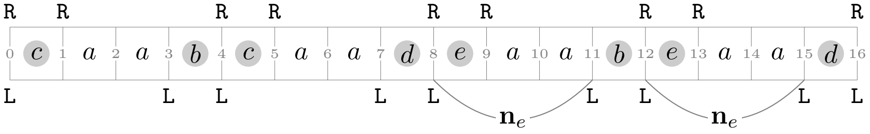
Round 3. • (C) implies $\texttt{New}(\texttt{E},\texttt{L},\texttt{R}) = w[\alpha(7,9)] = w[8] = d$;


R R R R R R R
0 c 1 a 2 a 3 b 4 c 5 a 6 a 7 d 8 e 9 a 10 a 11 b 12 e 13 a 14 a 15 d 16
L L L L L L L
$\mathbf{n}_d$
$\mathbf{n}_d$

- since $d \in \texttt{E}$, (Ba) implies $7, 15 \in \texttt{L}$, $8, 16 \in \texttt{R}$, and (Bb) implies $8, 16 \in \texttt{L}$, $5, 12 \in \texttt{R}$; the condition (Bc) is satisfied.


R R R R R R R R R
0 c 1 a 2 a 3 b 4 c 5 a 6 a 7 d 8 e 9 a 10 a 11 b 12 e 13 a 14 a 15 d 16
L L L L L L L L L

Round 4. • (C) implies $\texttt{New}(\texttt{E},\texttt{L},\texttt{R}) = w[\alpha(8,9)] = w[9] = e$;


R R R R R R R R R
0 c 1 a 2 a 3 b 4 c 5 a 6 a 7 d 8 e 9 a 10 a 11 b 12 e 13 a 14 a 15 d 16
L L L L L L L L L
$\mathbf{n}_e$
$\mathbf{n}_e$

- all conditions (B) are satisfied.

The remaining part of the algorithm MorphicFactorization defines

$$f : a \mapsto \varepsilon, \quad b \mapsto aab, \quad c \mapsto c, \quad d \mapsto aad, \quad e \mapsto e.$$

Note that also

$$f : a \mapsto \varepsilon, \quad b \mapsto ab, \quad c \mapsto ca, \quad d \mapsto ad, \quad e \mapsto ea,$$

and

$$f : a \mapsto \varepsilon, \quad b \mapsto b, \quad c \mapsto caa, \quad d \mapsto d, \quad e \mapsto eaa$$

are possible morphisms with the same set of expanding letters.

4. COMPLEXITY ANALYSIS

In this section, we show that the complexity of the algorithm is in $\mathcal{O}(m \cdot n)$, where n is the length of the analyzed word, and m is the number of its letters. More precisely, we show that the complexity is in

$$\mathcal{O}(|E| \cdot n),$$

where E is a minimal set of expanding letters.

The core of the algorithm is the **while** loop. The condition $\mathtt{E} = \mathtt{E}(\mathtt{L},\mathtt{R})$ is checked $|E|+1$ times and the loop is performed $|E|$ times since in each round one letter is added to $\mathtt{E}$. Therefore, we have to prove that each round of the loop can be performed in $\mathcal{O}(n)$.

It is convenient to calculate, during the initialization phase, the value of $|w|_a$ for each $a \in \mathrm{alph}(w)$, and also an array $\mathtt{Pos}[a,i]$, which yields the position of the ith occurrence of a in w. The preprocessing is linear: it is enough to read the input once.

4.1. Evaluation of the loop condition

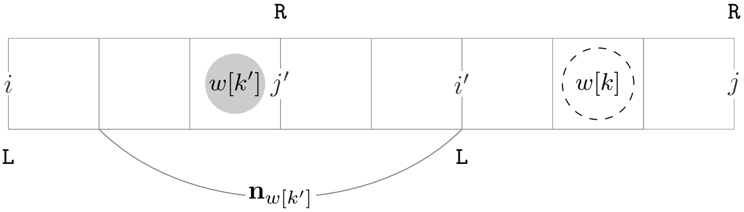
Evaluation of the loop condition consists in checking whether the stability condition (C) is satisfied. If it is not, then the evaluation also outputs the letter $\mathtt{New}(\mathtt{E},\mathtt{L},\mathtt{R})$.

The condition seems to require that a possibly quadratic number of pairs (i,j) is checked. Here we describe how the verification can be done in linear time (which is claimed but not proved in [4]).

Look through cuts $\mathtt{l}$ in $\mathtt{L}$ in increasing order and for each $\mathtt{l}$ find the smallest cut $\mathtt{r} \in \mathtt{R}$ strictly larger than $\mathtt{l}$, and $\mathtt{k} = \alpha(\mathtt{l},\mathtt{r})$. Here we exploit the preprocessing, which allows to obtain $|w|_{w[i]}$ in constant time. If $w[\mathtt{k}] \notin \mathtt{E}$, then we have found $\mathtt{New}(\mathtt{E},\mathtt{L},\mathtt{R})$ and start the next round of the **while** loop. If $\mathtt{l} = n$ and no violation of (C) was detected, return $\mathtt{E} = \mathtt{E}(\mathtt{L},\mathtt{R})$.

Note that $\mathtt{r}$ and $\mathtt{k}$ can never decrease, therefore the procedure is in $\mathcal{O}(n)$. However, not all factors $w[i,j]$ with $i \in \mathtt{L}$ and $j \in \mathtt{R}$ are checked; hence it has to be shown that the stability condition (C) is verified correctly.

Suppose, for a contradiction, that our procedure outputs $\mathtt{E} = \mathtt{E}(\mathtt{L},\mathtt{R})$, although $i \in \mathtt{L}$ and $j \in \mathtt{R}$ violate the stability condition (C). Assume that $j-i$ is as small as possible. Let $j' < j$ be the smallest cut in $\mathtt{R}$ strictly larger than i and let $k' = \alpha(i,j')$. Since the stretch factor $w[i,j']$ has been checked by the procedure, we deduce $w[k'] \in \mathtt{E}$ (and $k' = j'$). On the other hand, by assumption, we have $w[k] \notin \mathtt{E}$, where $k = \alpha(i,j)$. Hence $k' < k$ and $|w|_k < |w|_{k'}$. The stability condition (Bb) implies $i' = k' + |\mathbf{r}_{w[k']}| \in \mathtt{L}$. From $|w|_k < |w|_{k'}$, it is easy to conclude that the letter $w[k]$ is not in $\mathbf{n}_{w[k']}$ and therefore $i' < k$.



Clearly, $k = \alpha(i,j) = \alpha(i',j)$, whence the factor $w[i',j]$ violates (C) too, a contradiction with minimality of $j-i$.

4.2. Construction of $\mathtt{L}$ and $\mathtt{R}$

The construction of sets $\mathtt{L}$ and $\mathtt{R}$ in each round consists in checking the stability condition (B) (the stability condition (A) is fulfilled by the first line of the algorithm).

The condition (Ba) says that, for a new letter $a \in \mathtt{E}$, we have to add positions immediately before occurrences of a to the set $\mathtt{L}$, and positions immediately after its occurrences to the set $\mathtt{R}$. This can be done in $\mathcal{O}(|w|_a)$.

Similarly, the condition (Bb) adds starting positions of $\mathbf{n}_a$ to $\mathtt{R}$, and ending positions to $\mathtt{L}$, where a is a letter newly added to $\mathtt{E}$. This requires to calculate $\mathbf{n}_a$, which is done as follows. In order to calculate $|\mathbf{r}_a|$, check, for growing $k \geq 1$, whether all letters

$$w[\mathtt{Pos}[a,i]+k], \quad i = 1,2,\dots,|w|_a$$

agree, until a mismatch is encountered for $k = |\mathbf{r}_a| + 1$. Similarly, with decreasing $k \leq -1$, it is possible to calculate $|\mathbf{l}_a|$. The notion of a neighborhood implies that neighborhoods of different occurrences of the same letter cannot overlap too much; each position lies in at most two distinct neighborhoods of the same letter: once in its left part and once in its right part. The number of positions visited during the calculation is therefore at most $2n$. We conclude that the cost of calculating $\mathbf{n}_a$ and of satisfying (Bb) is in $\mathcal{O}(n)$.

The stability condition (Bc) is the most complex one. It can be concisely described as keeping all neighborhoods $\mathbf{n}_a$ of the same letter a from $\mathtt{E}$ synchronized. The underlying structure is an undirected graph with vertices $\mathcal{C}_w$ satisfying the following condition:

$$\text{cuts } \mathtt{Pos}[a,i]+k \text{ and } \mathtt{Pos}[a,i']+k$$

are connected for each

$$a \in \mathtt{E},\ 1 \leq i,i' \leq |w|_a \text{ and } -|\mathbf{l}_a| - 1 \leq k \leq |\mathbf{r}_a|.$$

The condition (Bc) then requires that connected cuts either all are, or all are not elements of $\mathtt{L}$ (of $\mathtt{R}$ resp.). In other words, being in $\mathtt{L}$ (in $\mathtt{R}$ resp.) is a property of a connected component rather than of an individual cut. We shall represent this information as a forest of rooted trees of height one. Each cut is linked to its parent, which is the root representing the connected component. The root also keeps the information whether the component is in sets $\mathtt{L}$, $\mathtt{R}$. Checking whether the cut is in $\mathtt{L}$ (in $\mathtt{R}$ resp.) therefore requires constant time.

When a new letter a is added to $\mathtt{E}$, new edges synchronizing neighborhoods of a have to be added too, and the graph becomes more complex. To satisfy the condition (Bc) as it is formulated in the previous paragraph, it is enough to add edges

$$(\mathtt{Pos}[a,1]+k, \mathtt{Pos}[a,i]+k)$$

for $i = 2,\dots,|w|_a$ and $-|\mathbf{l}_a| - 1 \leq k \leq |\mathbf{r}_a|$. The number of new edges can be bounded by an argument similar to the one used above when calculating neighborhoods: each cut is the second vertex of a new edge at most two times. This implies that the number of new edges is less than $2n$. After new edges have been added, the algorithm searches the whole graph and compresses the connected components back to the forest of height one. Since the graph has at most n old vertices and at most $2n$ new ones, this can be done in $\mathcal{O}(n)$.

The final definition of f is clearly in $\mathcal{O}(n)$, which completes the proof.

5. CONCLUSION

We have shown that morphic primitivity can be tested in linear time for a fixed alphabet. This may be surprising compared with the fact that a similar problem, checking the existence of a morphism between two *distinct* words, is NP-complete (cf. [1]).

If the alphabet is not fixed, the algorithm is at worst quadratic, consider for example the family of morphically primitive words

$$w_n = a_1 a_2 \cdots a_{n-1} a_n a_n a_{n-1} \cdots a_2 a_1,$$

for which the main loop of the algorithm runs $n/2$ rounds. On the other hand, our analysis implies that it can be checked in linear time that all letters in w_n have trivial neighborhoods, whence the morphic primitivity follows. Precise complexity in the uniform case therefore remains unclear.

(Received July 20, 2012)

REFERENCES

[1] A. Ehrenfeucht and G. Rozenberg: Finding a homomorphism between two words is NP-complete. Inform. Process. Lett. *9* (1979), 86–88.

[2] T. Head: Fixed languages and the adult languages of OL schemes. Internat. J. Comput. Math. *10* (1981/82), 103–107.

[3] T. Head and B. Lando: Fixed and stationary ω-words and ω-languages. In: The Book of L (G. Rozenberg and A. Salomaa, eds.), Springer-Verlag, Berlin – Heidelberg 1986, pp. 147–156.

[4] Š. Holub: Polynomial algorithm for fixed points of nontrivial morphisms. Discrete Math. *309* (2009), 5069–5076.

[5] D. Reidenbach and J. C. Schneider: Morphically primitive words. Theoret. Comput. Sci. *410* (2009), 2148–2161.

[6] J. Shallit and M. W. Wang: On two-sided infinite fixed points of morphisms. Theoret. Comput. Sci. *270* (2002), 659–675.

[7] `http://www.karlin.mff.cuni.cz/~holub/soubory/Vizual/stranka2.html`

Štěpán Holub, Department of Algebra, Charles University, Sokolovská 83, 186 75 Praha 8. Czech Republic.
e-mail: holub@karlin.mff.cuni.cz

Vojtěch Matocha, Department of Algebra, Charles University, Sokolovská 83, 186 75 Praha 8. Czech Republic.
e-mail: matochav@volny.cz