Linear-Time Algorithm for Morphic Imprimitivity Testing

Tomasz Kociumaka[1] Jakub Radoszewski[1]
Wojciech Rytter [1,2] **Tomasz Waleń**[3,1]

[1]Faculty of Mathematics, Informatics and Mechanics, University of Warsaw
`{kociumaka,jrad,rytter,walen}@mimuw.edu.pl`

[2]Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, Toruń

[3]International Institute of Molecular and Cell Biology in Warsaw

LATA 2013, 2013–04–05

Outline

1. Problem definition
2. Short introduction to existing solutions
3. Description of the new linear time solution

Problem definition

Morphic Imprimitivity Testing

For a input word $w \in \Sigma^n$, is there a *non-trivial* morphism h such that:

$$h(w) = w$$

Non-trivial means that h should not be an identity function.
The word w is *non-primitive* if such morphism exists,
otherwise it is *primitive*.

Problem definition

Morphic Imprimitivity Testing

For a input word $w \in \Sigma^n$, is there a *non-trivial* morphism h such that:

$$h(w) = w$$

Non-trivial means that h should not be an identity function.
The word w is *non-primitive* if such morphism exists,
otherwise it is *primitive*.

Previous results

- it can be solved in $O((|\Sigma| + \log n) \cdot n)$ time (S. Holub 2009),
- slightly improved to $O(|\Sigma| \cdot n)$ time (S. Holub, V. Matocha, arXiv 2012).

Example

Simple case

Let

$$w = \texttt{abaacaca}$$

Example

Simple case

Let

$$w = \texttt{abaacaca}$$

Letter b appears only once, so we can take:

$$h(a) = \epsilon \text{ (empty word)} \quad h(b) = \texttt{abaacaca} \quad h(c) = \epsilon$$

Example

Simple case

Let

$$w = \texttt{abaacaca}$$

Letter b appears only once, so we can take:

$$h(a) = \epsilon \text{ (empty word)} \quad h(b) = \texttt{abaacaca} \quad h(c) = \epsilon$$

More complicated case

Let

$$w = \texttt{aacabaaaacaacabaa}$$

Example

Simple case

Let

$$w = \texttt{abaacaca}$$

Letter b appears only once, so we can take:

$$h(a) = \epsilon \text{ (empty word)} \quad h(b) = \texttt{abaacaca} \quad h(c) = \epsilon$$

More complicated case

Let

$$w = \texttt{aac}\,\texttt{abaa}\,\texttt{aac}\,\texttt{aac}\,\texttt{abaa}$$

we can take:

$$h(a) = \epsilon \quad h(b) = \texttt{abaa} \quad h(c) = \texttt{aac}$$

Problem applications

Closely connected to several topics in formal language theory, and combinatorics on words:

- fixed points of morphisms,
- pattern languages,
- ambiguity of the morphisms.

Problem applications

Closely connected to several topics in formal language theory, and combinatorics on words:

- fixed points of morphisms,
- pattern languages,
- ambiguity of the morphisms.

Reviewer's opinion

Although I cannot think of any actual applications, I find this question to be very natural

How to solve it? - Intuition

Theorem

For a word w, if there exists non-trivial morphism h, such that $h(w) = w$, then there exists non-trivial morphism h' such that:

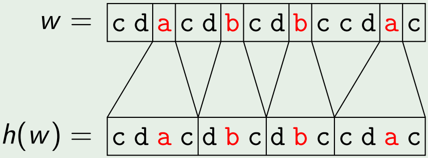
- $h'(w) = w$
- *for all immortal letters* $x \in E$*:* $h'(x) = l_x\, x\, r_x$
  (i.e. $h'(\mathtt{b}) = \mathtt{abaa}$*)*
- *for all mortal letters* $x \notin E$*:* $h'(x) = \epsilon$

How to solve it? - Intuition

Theorem

For a word w, if there exists non-trivial morphism h, such that $h(w) = w$, then there exists non-trivial morphism h' such that:

- $h'(w) = w$
- *for all immortal letters* $x \in E$: $h'(x) = l_x\, x\, r_x$
  (i.e. $h'(\mathtt{b}) = \mathtt{abaa}$*)*
- *for all mortal letters* $x \notin E$: $h'(x) = \epsilon$



$w = $ c d a c d b c d b c c d a c

$h(w) = $ c d a c d b c d b c c d a c

$h(\mathtt{a}) = \mathtt{cdac}$
$h(\mathtt{b}) = \mathtt{dbc}$
$h(\mathtt{c}) = \epsilon$
$h(\mathtt{d}) = \epsilon$

Holub's algorithm

The algorithm maintains three sets:

- E – set of candidates for *immortal* letters,
- L and R – sets of interpositions.

Holub's algorithm

The algorithm maintains three sets:

- E – set of candidates for *immortal* letters,
- L and R – sets of interpositions.

Algorithm:

- start with empty sets $E = L = R = \emptyset$,
- apply rules (a)-(e) (in any order), to obtain fixed-point.

Holub's algorithm

The algorithm maintains three sets:

- E – set of candidates for *immortal* letters,
- L and R – sets of interpositions.

Algorithm:

- start with empty sets $E = L = R = \emptyset$,
- apply rules (a)-(e) (in any order), to obtain fixed-point.

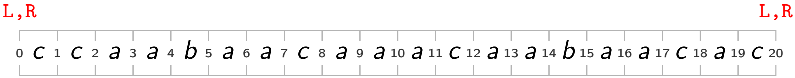
From triple (E, L, R) the actual morphism can be obtained:

- if the set $E \neq \Sigma$, then the morphism is non-trivial,
- from L, R we can deduce a way to divide input word to obtain morphism.

Holub's rule (a) – initialization of the algorithm

$L := L \cup \{0, n\},\ R := R \cup \{0, n\}$

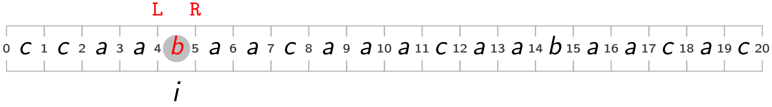
Example:



Holub's rule (b) – initialization of immortal letters

```
**if** $w[i] \in E$ **then**
    $L := L \cup \{i-1\}$ and $R := R \cup \{i\}$,
```

Example:

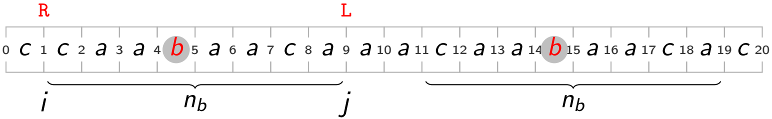


Holub's rule (c) – neighborhood marking

The neighborhood of letter x – n_x is the maximum factor that surrounds *each* occurrence of letter x in w.

```
**if** $w[i..j] = n_x$ for some $x \in E$ **then**
    $R := R \cup \{i-1\}$ and $L := L \cup \{j\}$,
```

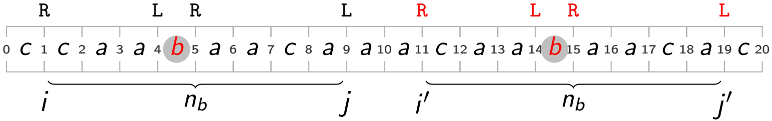
Example:



Holub's rule (d) – copying rules

```
if $w[i..j] = w[i'..j'] = n_a$ for some $a \in E$ and $i - 1 \le k \le j$ then
    if $w[k] \in L$ then $L := L \cup \{i' + (k - i)\}$
    if $w[k] \in R$ then $R := R \cup \{i' + (k - i)\}$
```

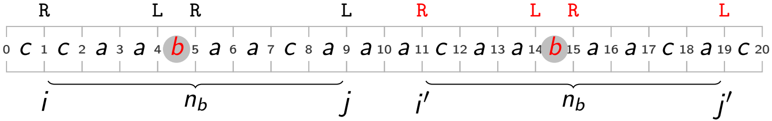
Example:



Holub's rule (d) – copying rules

```
if $w[i..j] = w[i'..j'] = n_a$ for some $a \in E$ and $i - 1 \leq k \leq j$ then
    if $w[k] \in L$ then $L := L \cup \{i' + (k - i)\}$
    if $w[k] \in R$ then $R := R \cup \{i' + (k - i)\}$
```

Example:



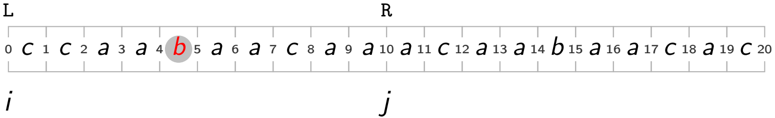
Problem

This rule is hard to implement efficiently!

Holub's rule (e) – new immortals letters

```
if $i < j$, $i \in L$, $j \in R$ then
    add $\alpha(w[(i+1)..j])$ to $E$ — letter $c \in w[(i+1)..j]$
        that has smallest number of occurrences in word $w$.
```

Example:



Holub's algorithm summary

Theorem

Extending a correct triple (E, L, R) *using any of the rules (a)-(e) leads to a correct triple. In particular, if any sequence of actions corresponding to (a)-(e) leads to* $E = \Sigma$ *then* w *is morphically primitive.*

Holub's algorithm summary

Theorem

Extending a correct triple (E, L, R) *using any of the rules (a)-(e) leads to a correct triple. In particular, if any sequence of actions corresponding to (a)-(e) leads to* $E = \Sigma$ *then* w *is morphically primitive.*

This is quite suprising that this set of simple rules, provides the solution for the problem.

Holub's algorithm summary

- simple implementation requires $O(n^2)$ time,
- this time complexity can be slightly improved using some preprocessing and data structures,
- unfortunately the obtaining linear time seems to be difficult task:
    - the non-determinism in rules choice is problematic,
    - rule (d) is the main bottleneck (it operates globally on the word).

What we have done? Outline

- modified set of rules (a),(b')–(e'), that are equivalent to Holub's rules but are easier to implement,
- strict ordering of rules application,
- new data structures to speed up the processing time.

What we have done? Outline

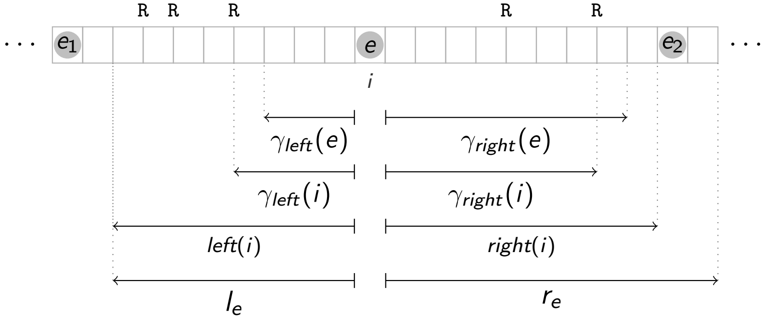
- modified set of rules (a),(b')–(e'), that are equivalent to Holub's rules but are easier to implement,
- strict ordering of rules application,
- new data structures to speed up the processing time.

Result

As a consequence we obtained $O(n)$ running time algorithm.

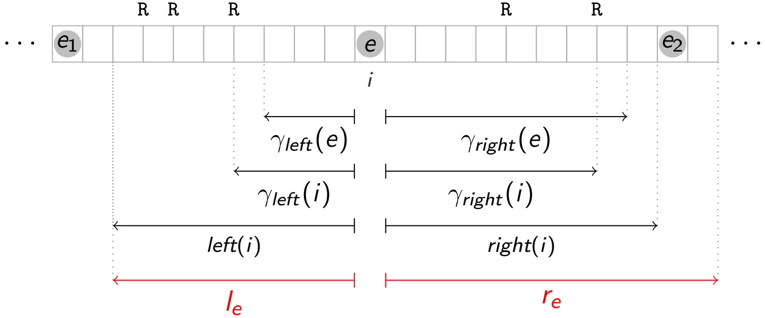
New neighborhood definitions

We introduced new definitions of neighborhood, to capture essential local neighborhood of the characters/word positions.



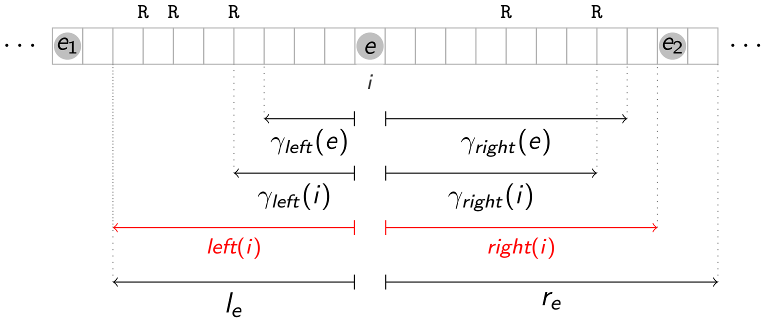
New neighborhood definitions

l_e – the length of the longest common suffix of all prefixes ending with e (minus 1) in word w.
r_e – the length of the longest common prefix of all suffixes starting with e (minus 1) in word w,



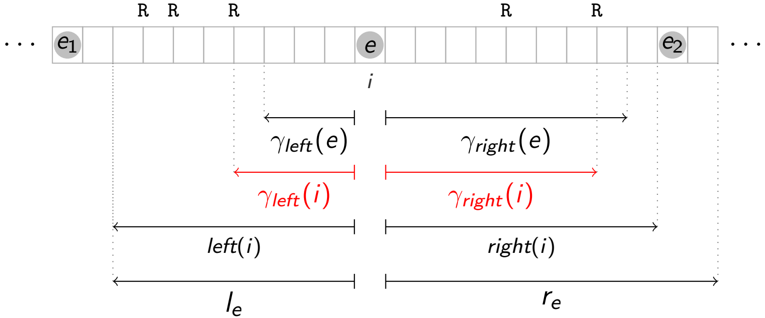
New neighborhood definitions

$$left(i) = \min(l_{w[i]},\ i - pred_E(i) - 1)$$
$$right(i) = \min(r_{w[i]},\ succ_E(i) - i - 1)$$



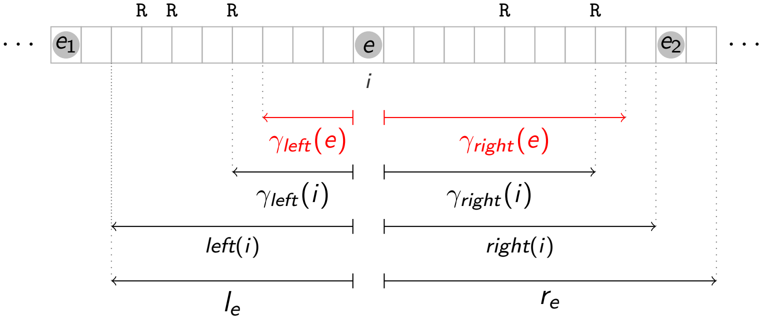
New neighborhood definitions

$$\gamma_{left}(i) = i - pred_R(i) - 1$$
$$\gamma_{right}(i) = pred_R(i + right(i) + 1) - i$$



New neighborhood definitions

$$\gamma_{left}(e) = \min\{\gamma_{left}(i) : i \in Occ(e)\}$$
$$\gamma_{right}(e) = \max\{\gamma_{right}(i) : i \in Occ(e)\}$$

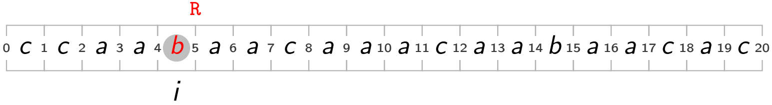


New rule (b')

Old: **if** $w[i] \in E$ **then**
$L := L \cup \{i - 1\}$ and $R := R \cup \{i\}$,

New: **if** $w[i] \in E$ **then**
$R := R \cup \{i\}$,

Example:

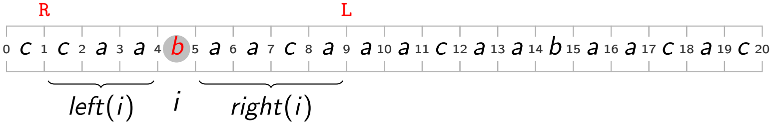


New rule (c')

Old: **if** $w[i..j] = n_x$ for some $x \in E$ **then**
$R := R \cup \{i-1\}$ and $L := L \cup \{j\}$,

New: **if** $w[i] \in E$ **then**
$R := R \cup \{i - 1 - \mathit{left}(i)\}$ and $L := L \cup \{i + \mathit{right}(i)\}$,

Example:



New rule (d')

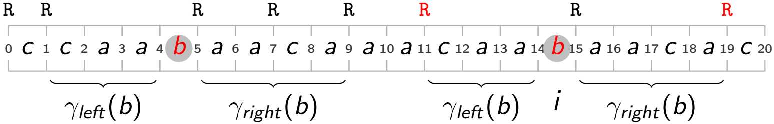
Old: **if** $w[i..j] = w[i'..j'] = n_a$ for some $a \in E$ and $i - 1 \leq k \leq j$ **then**

```
    if $w[k] \in L$ then $L := L \cup \{i' + (k - i)\}$
    if $w[k] \in R$ then $R := R \cup \{i' + (k - i)\}$
```

New: **if** $w[i] \in E$ **then**

$$R := R \cup \{i - 1 - \gamma_{left}(w[i]), i + \gamma_{right}(w[i])\}$$

Example:

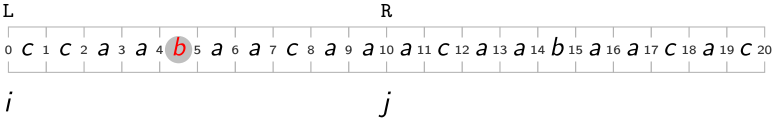


New rule (e')

Old: **if** $i < j$, $i \in L$, $j \in R$ **then**
add $\alpha(w[(i+1)..j])$ to E

New:
if $i < j$, $succ_R(i) = j$, $pred_L(j) = i$, $\{w[k] : i+1 \leq k \leq j\} \cap E = \emptyset$
then
add $\alpha(w[(i+1)..j])$ to E — letter $c \in w[(i+1)..j]$
that has smallest number of occurrences in word w.

Example:



New rules correctness

Theorem

Extending a correct triple (E, L, R) using any of the rules (a),(b')-(e') leads to a correct triple. In particular, if any sequence of actions corresponding to (a),(b')-(e') leads to $E = \Sigma$ then w is morphically primitive.

New rules correctness

Theorem

Extending a correct triple (E, L, R) using any of the rules (a),(b')-(e') leads to a correct triple. In particular, if any sequence of actions corresponding to (a),(b')-(e') leads to $E = \Sigma$ then w is morphically primitive.

Proof outline

We can show that using new rules we can simulate *essential* behavior of Holub's algorithm.

Efficient implementation

Unfortunately that's not over, we have to deal with:

Efficient implementation

Unfortunately that's not over, we have to deal with:

Non-determinism:

- This is resolved with events queues that handle the order of application of the rules. Especially we have to be careful to apply rules only when they add new elements to E, L, R.

Efficient implementation

Unfortunately that's not over, we have to deal with:

Non-determinism:

- This is resolved with events queues that handle the order of application of the rules. Especially we have to be careful to apply rules only when they add new elements to E, L, R.

Data structures:

- For answering $\alpha(i,j)$ queries in $O(1)$ time we use Range-Minimum-Queries (RMQ) data structure,
- For efficient computing the neighborhoods we use Suffix Arrays combined with Longest Common Prefix table.

Summary

- we presented a linear time algorithm for deciding if a word is morphically imprimitive,
- we started from the original quadratic algorithm by Holub, and transformed it by reducing the set of rules used by the algorithm,
- finally we proposed several efficient data structures that enabled linear-time implementation.

Thank you for your attention!