Minimal Forbidden Words and Applications

Gabriele Fici

Dipartimento di Matematica e Informatica
Università di Palermo

December 9th, 2013

Languages

Given a language L over a finite alphabet A we say that L is:

Languages

Given a language L over a finite alphabet A we say that L is:

- factorial if L contains all the factors of its words, i.e. $uv \in L \Rightarrow u, v \in L$

Languages

Given a language L over a finite alphabet A we say that L is:

- factorial if L contains all the factors of its words, i.e. $uv \in L \Rightarrow u, v \in L$
- anti-factorial if no word in L is factor of another word in L, i.e. $uv \in L \Rightarrow u, v \notin L$

Languages

Given a language L over a finite alphabet A we say that L is:

- factorial if L contains all the factors of its words, i.e. $uv \in L \Rightarrow u, v \in L$
- anti-factorial if no word in L is factor of another word in L, i.e. $uv \in L \Rightarrow u, v \notin L$

For example, the set of factors of a (finite or infinite) word is a factorial language.

Minimal Forbidden Words

Definition

Given a factorial language L, we say that $w \in A^*$ is a minimal forbidden word for L if:

1. $w \notin L$;
2. every proper factor of w is in L.

Minimal Forbidden Words

Definition

Given a factorial language L, we say that $w \in A^*$ is a minimal forbidden word for L if:

1. $w \notin L$;
2. every proper factor of w is in L.

The (antifactorial) set of mfw for L is denoted by $\mathcal{MF}(L)$.

Minimal Forbidden Words

Definition

Given a factorial language L, we say that $w \in A^*$ is a minimal forbidden word for L if:

1. $w \notin L$;
2. every proper factor of w is in L.

The (antifactorial) set of mfw for L is denoted by $\mathcal{MF}(L)$.

If L is the set of factors of a word w, the set $\mathcal{MF}(w) = \mathcal{MF}(L)$ is usually called the set of minimal forbidden factors of w.

Minimal Forbidden Words

Definition

Given a factorial language L, we say that $w \in A^*$ is a minimal forbidden word for L if:

1. $w \notin L$;
2. every proper factor of w is in L.

The (antifactorial) set of mfw for L is denoted by $\mathcal{MF}(L)$.

If L is the set of factors of a word w, the set $\mathcal{MF}(w) = \mathcal{MF}(L)$ is usually called the set of minimal forbidden factors of w.

Example

Over $A = \{a, b\}$ let $w = aabbbaa$. We have:

$$\mathcal{MF}(w) = \{aaa, bbbb, aba, abba, bab, baab\}$$

Languages

The map $\mu : L \mapsto \mathcal{MF}(L)$ is injective, i.e., different languages have different sets of minimal forbidden words.

Languages

The map $\mu : L \mapsto \mathcal{MF}(L)$ is injective, i.e., different languages have different sets of minimal forbidden words.

Conversely, given an antifactorial language M, we can define $\mathcal{L}(M)$ as the largest (factorial) language avoiding M, i.e. $\mathcal{L}(M) = (A^* M A^*)^c$.

Languages

The map $\mu : L \mapsto \mathcal{MF}(L)$ is injective, i.e., different languages have different sets of minimal forbidden words.

Conversely, given an antifactorial language M, we can define $\mathcal{L}(M)$ as the largest (factorial) language avoiding M, i.e. $\mathcal{L}(M) = (A^*MA^*)^c$. The map $\lambda : M \mapsto \mathcal{L}(M)$ is injective and is the inverse of the map μ.

Languages

The map $\mu : L \mapsto \mathcal{MF}(L)$ is injective, i.e., different languages have different sets of minimal forbidden words.

Conversely, given an antifactorial language M, we can define $\mathcal{L}(M)$ as the largest (factorial) language avoiding M, i.e. $\mathcal{L}(M) = (A^*MA^*)^c$. The map $\lambda : M \mapsto \mathcal{L}(M)$ is injective and is the inverse of the map μ.

In fact, $\mathcal{L}(\mathcal{MF}(L)) = L$ and $\mathcal{MF}(\mathcal{L}(M)) = M$.

Languages

The map $\mu : L \mapsto \mathcal{MF}(L)$ is injective, i.e., different languages have different sets of minimal forbidden words.

Conversely, given an antifactorial language M, we can define $\mathcal{L}(M)$ as the largest (factorial) language avoiding M, i.e. $\mathcal{L}(M) = (A^*MA^*)^c$. The map $\lambda : M \mapsto \mathcal{L}(M)$ is injective and is the inverse of the map μ.

In fact, $\mathcal{L}(\mathcal{MF}(L)) = L$ and $\mathcal{MF}(\mathcal{L}(M)) = M$.

Theorem (Crochemore, Mignosi, Restivo [1])

There is a one-to-one correspondence between factorial and antifactorial languages.

Languages

The map $\mu : L \mapsto \mathcal{MF}(L)$ is injective, i.e., different languages have different sets of minimal forbidden words.

Conversely, given an antifactorial language M, we can define $\mathcal{L}(M)$ as the largest (factorial) language avoiding M, i.e. $\mathcal{L}(M) = (A^*MA^*)^c$. The map $\lambda : M \mapsto \mathcal{L}(M)$ is injective and is the inverse of the map μ.

In fact, $\mathcal{L}(\mathcal{MF}(L)) = L$ and $\mathcal{MF}(\mathcal{L}(M)) = M$.

Theorem (Crochemore, Mignosi, Restivo [1])

There is a one-to-one correspondence between factorial and antifactorial languages.

Moreover, this correspondence preserves the regularity, i.e., a language is regular iff its set of mfw is regular [1].

CMR Algorithm

If M is finite, then it can be represented on a trie (tree-like automaton) $\mathcal{T}(M)$.

CMR Algorithm

If M is finite, then it can be represented on a trie (tree-like automaton) $\mathcal{T}(M)$.

Theorem (Crochemore, Restivo, Mignosi, [1])

A determinitic automaton $\mathcal{A}(M)$ accepting $\mathcal{L}(M)$ can be computed from $\mathcal{T}(M)$ in linear time.
Moreover, if $M = \mathcal{MF}(w)$, then $\mathcal{A}(M)$ is the factor automaton (DAWG) of w, i.e., it is minimal.

CMR Algorithm

If M is finite, then it can be represented on a trie (tree-like automaton) $\mathcal{T}(M)$.

Theorem (Crochemore, Restivo, Mignosi, [1])

A determinitic automaton $\mathcal{A}(M)$ accepting $\mathcal{L}(M)$ can be computed from $\mathcal{T}(M)$ in linear time.
Moreover, if $M = \mathcal{MF}(w)$, then $\mathcal{A}(M)$ is the factor automaton (DAWG) of w, i.e., it is minimal.

Theorem (Crochemore, Restivo, Mignosi, [1])

Given the factor automaton of a word w, a trie accepting $\mathcal{MF}(w)$ can be computed in linear time.

BCMRS Algorithm

Theorem (Béal, Crochemore, Mignosi, Restivo, Sciortino [4])

Given a deterministic automaton $\mathcal{A}(L)$ accepting a factorial language L, it is possible to build in quadratic time (which is optimal in the worst case) a deterministic automaton accepting $\mathcal{MF}(L)$.

BCMRS Algorithm

Theorem (Béal, Crochemore, Mignosi, Restivo, Sciortino [4])

Given a deterministic automaton $\mathcal{A}(L)$ accepting a factorial language L, it is possible to build in quadratic time (which is optimal in the worst case) a deterministic automaton accepting $\mathcal{MF}(L)$.

Actually, if the input is the factor automaton of a word w, i.e., the minimal deterministic automaton accepting $Fact(w)$, the previous algorithm takes linear time.

BCMRS Algorithm

Theorem (Béal, Crochemore, Mignosi, Restivo, Sciortino [4])

Given a deterministic automaton $\mathcal{A}(L)$ accepting a factorial language L, it is possible to build in quadratic time (which is optimal in the worst case) a deterministic automaton accepting $\mathcal{MF}(L)$.

Actually, if the input is the factor automaton of a word w, i.e., the minimal deterministic automaton accepting $Fact(w)$, the previous algorithm takes linear time.

Corollary

The bijective correspondence between w and $\mathcal{MF}(w)$ can be computed in linear time in each direction.

Combinatorial properties of MFW

Given a word w, the repetition index $r(w)$ is the length of the longest factor of w that has more than one occurrences in w.

Combinatorial properties of MFW

Given a word w, the repetition index $r(w)$ is the length of the longest factor of w that has more than one occurrences in w.

Proposition

Let $w \in A^*$ be generated by a memoryless source with identical symbol probabilities. Then the probability that $r(w) \leq 3 \log_{|A|} |w|$ tends to 1 as $|w|$ tends to infinity.

Combinatorial properties of MFW

Given a word w, the repetition index $r(w)$ is the length of the longest factor of w that has more than one occurrences in w.

Proposition

Let $w \in A^*$ be generated by a memoryless source with identical symbol probabilities. Then the probability that $r(w) \leq 3 \log_{|A|} |w|$ tends to 1 as $|w|$ tends to infinity.

Proposition

Let $m(w)$ be the length of the longest mff of w. Then $m(w) = r(w) + 2$.

Combinatorial properties of MFW

Given a word w, the repetition index $r(w)$ is the length of the longest factor of w that has more than one occurrences in w.

Proposition

Let $w \in A^*$ be generated by a memoryless source with identical symbol probabilities. Then the probability that $r(w) \leq 3 \log_{|A|} |w|$ tends to 1 as $|w|$ tends to infinity.

Proposition

Let $m(w)$ be the length of the longest mff of w. Then $m(w) = r(w) + 2$.

Example

Let $w = aabbbaa$. Then $r(w) = 2$ since every factor of length 3 is unioccurrent. A longest mff for w has length 4, that is, $m(w) = 4$.

Data Compression using Antidictionaries

Minimal forbidden words can be used to compress a text.

Data Compression using Antidictionaries

Minimal forbidden words can be used to compress a text.

Definition

An antidictionary for a word w is a subset of $\mathcal{MF}(w)$.

Data Compression using Antidictionaries

Minimal forbidden words can be used to compress a text.

Definition

An antidictionary for a word w is a subset of $\mathcal{MF}(w)$.

For example, let $w = 0100101001$. Then $AD = \{000, 10101, 11\}$ is an antidictionary for w.

Data Compression using Antidictionaries

Minimal forbidden words can be used to compress a text.

Definition

An antidictionary for a word w is a subset of $\mathcal{MF}(w)$.

For example, let $w = 0100101001$. Then $AD = \{000, 10101, 11\}$ is an antidictionary for w.

The idea is to eliminate redundant letters of w, which can be retrieved from AD.

Data Compression using Antidictionaries

Minimal forbidden words can be used to compress a text.

Definition

An antidictionary for a word w is a subset of $\mathcal{MF}(w)$.

For example, let $w = 0100101001$. Then $AD = \{000, 10101, 11\}$ is an antidictionary for w.

The idea is to eliminate redundant letters of w, which can be retrieved from AD.

Crochemore, Mignosi, Restivo and Salemi [2] proposed a lossless antidictionary-based compressor.

ENCODER (AD, $w \in \{0,1\}^*$)
1. $v \leftarrow \varepsilon$; $\gamma \leftarrow \varepsilon$;
2. **for** $a \leftarrow$ first to last letter of w
3. **if** $\forall$ suffix v' of v, $v'0$ and $v'1 \notin AD$
4. $\gamma \leftarrow \gamma a$;
5. $v \leftarrow va$;
6. **return** $(|v|, \gamma)$;

Example: $w = 0100101001$.

$v = \varepsilon$	$\gamma(w) = \varepsilon$	
$v = 0$	$\gamma(w) = 0$	
$v = 01$	$\gamma(w) = 01$	$v' = 11 \in AD$
$v = 010$	$\gamma(w) = 01$	
$v = 0100$	$\gamma(w) = 010$	$v' = 000 \in AD$
$v = 01001$	$\gamma(w) = 010$	$v' = 11 \in AD$
$v = 010010$	$\gamma(w) = 010$	
$v = 0100101$	$\gamma(w) = 0101$	$v' = 11 \in AD$
$v = 01001010$	$\gamma(w) = 0101$	$v' = 10101 \in AD$
$v = 010010100$	$\gamma(w) = 0101$	$v' = 000 \in AD$
$v = 0100101001$	$\gamma(w) = 0101$	$v' = 11 \in AD$

Data Compression using Antidictionaries

DECODER (AD, γ, n)

$v \leftarrow \varepsilon$;
while $|v| < n$
    if for some v' suffix of v and letter a, $v'a \in AD$
        $v \leftarrow v\bar{a}$;
    else
        $a \leftarrow$ next letter of γ;
        $v \leftarrow va$;
return (v);

$v = \varepsilon$	$\gamma(w) = \varepsilon$	
$v = 0$	$\gamma(w) = 0$	
$v = 01$	$\gamma(w) = 01$	$v' = 11 \in AD$
$v = 010$	$\gamma(w) = 01$	
$v = 0100$	$\gamma(w) = 010$	$v' = 000 \in AD$
$v = 01001$	$\gamma(w) = 010$	$v' = 11 \in AD$
$v = 010010$	$\gamma(w) = 010$	
$v = 0100101$	$\gamma(w) = 0101$	$v' = 11 \in AD$
$v = 01001010$	$\gamma(w) = 0101$	$v' = 10101 \in AD$
$v = 010010100$	$\gamma(w) = 0101$	$v' = 000 \in AD$
$v = 0100101001$	$\gamma(w) = 0101$	$v' = 11 \in AD$

Word Reconstruction

Another application of mfw concerns the reconstruction of a word from a set of factors [6].
This is a theoretical simplified model for the Fragment Assembly Problem.

Word Reconstruction

Another application of mfw concerns the reconstruction of a word from a set of factors [6].
This is a theoretical simplified model for the Fragment Assembly Problem.

Definition

Given a finite set of words $\mathcal{I}$, we say that a word w is $\mathcal{I}$-compatible if:

1. $\mathcal{I} \subset Fact(w)$;
2. every factor of w shorter than $m(w)$ appears in some word of $\mathcal{I}$.

Example

$\mathcal{I} = \{abb, bba\}$. Then $abba$ is $\mathcal{I}$-compatible.
$\mathcal{I} = \{ab, bb, ba\}$. Then no word is $\mathcal{I}$-compatible.

Word Reconstruction

Another application of mfw concerns the reconstruction of a word from a set of factors [6].
This is a theoretical simplified model for the Fragment Assembly Problem.

Definition

Given a finite set of words $\mathcal{I}$, we say that a word w is $\mathcal{I}$-compatible if:

1. $\mathcal{I} \subset Fact(w)$;
2. every factor of w shorter than $m(w)$ appears in some word of $\mathcal{I}$.

Example

$\mathcal{I} = \{abb, bba\}$. Then $abba$ is $\mathcal{I}$-compatible.
$\mathcal{I} = \{ab, bb, ba\}$. Then no word is $\mathcal{I}$-compatible.

Theorem

For any $\mathcal{I}$, there exists at most one $\mathcal{I}$-compatible word.

Word Reconstruction

The algorithm for the reconstruction takes a set $\mathcal{I}$ in input, and in linear time on $|\mathcal{I}|$ reconstructs an $\mathcal{I}$-compatible word if this exists, or gives a negative answer.

Word Reconstruction

The algorithm for the reconstruction takes a set $\mathcal{I}$ in input, and in linear time on $|\mathcal{I}|$ reconstructs an $\mathcal{I}$-compatible word if this exists, or gives a negative answer.

Idea: if we are able to retrieve the set $\mathcal{MF}(w)$, then we can retrieve w.

Word Reconstruction

The algorithm for the reconstruction takes a set $\mathcal{I}$ in input, and in linear time on $|\mathcal{I}|$ reconstructs an $\mathcal{I}$-compatible word if this exists, or gives a negative answer.

Idea: if we are able to retrieve the set $\mathcal{MF}(w)$, then we can retrieve w.

So, first we construct the word

$$w_1 = \$ i_1 \$ i_2 \$ \cdots \$ i_n \$$$

where $i_1, \ldots, i_n = \mathcal{I}$ and $\$ \notin A$. Then we compute the set $\mathcal{MF}(w_1)$.

Word Reconstruction

The algorithm for the reconstruction takes a set $\mathcal{I}$ in input, and in linear time on $|\mathcal{I}|$ reconstructs an $\mathcal{I}$-compatible word if this exists, or gives a negative answer.

Idea: if we are able to retrieve the set $\mathcal{MF}(w)$, then we can retrieve w.

So, first we construct the word

$$w_1 = \$i_1\$i_2\$\cdots\$i_n\$$$

where $i_1, \ldots, i_n = \mathcal{I}$ and $\$ \notin A$. Then we compute the set $\mathcal{MF}(w_1)$.

But how can we retrieve $\mathcal{MF}(w)$ from $\mathcal{MF}(w_1)$?

Word Reconstruction

The algorithm for the reconstruction takes a set $\mathcal{I}$ in input, and in linear time on $|\mathcal{I}|$ reconstructs an $\mathcal{I}$-compatible word if this exists, or gives a negative answer.

Idea: if we are able to retrieve the set $\mathcal{MF}(w)$, then we can retrieve w.

So, first we construct the word

$$w_1 = \$i_1\$i_2\$\cdots\$i_n\$$$

where $i_1, \ldots, i_n = \mathcal{I}$ and $\$ \notin A$. Then we compute the set $\mathcal{MF}(w_1)$.

But how can we retrieve $\mathcal{MF}(w)$ from $\mathcal{MF}(w_1)$?

Proposition

If w is $\mathcal{I}$-compatible, then $\mathcal{MF}(w) = \mathcal{MF}(w_1) \cap A^{\leq m(w)}$.

Word Reconstruction

The algorithm for the reconstruction takes a set $\mathcal{I}$ in input, and in linear time on $|\mathcal{I}|$ reconstructs an $\mathcal{I}$-compatible word if this exists, or gives a negative answer.

Idea: if we are able to retrieve the set $\mathcal{MF}(w)$, then we can retrieve w.

So, first we construct the word

$$w_1 = \$i_1\$i_2\$\cdots\$i_n\$$$

where $i_1, \ldots, i_n = \mathcal{I}$ and $\$ \notin A$. Then we compute the set $\mathcal{MF}(w_1)$.

But how can we retrieve $\mathcal{MF}(w)$ from $\mathcal{MF}(w_1)$?

Proposition

If w is $\mathcal{I}$-compatible, then $\mathcal{MF}(w) = \mathcal{MF}(w_1) \cap A^{\leq m(w)}$.

Wonderful, but we don't know the value $m(w)$...

Word Reconstruction

Let S be the set of words $aub \in \mathcal{MF}(w_1) \cap A^*$ such that:

1. $au\$, \$ub \in Fact(w_1)$;
2. $aux, xub \notin Fact(w_1)$ for any $x \in A$.

Word Reconstruction

Let S be the set of words $aub \in \mathcal{MF}(w_1) \cap A^*$ such that:

1. $au\$, \$ub \in Fact(w_1)$;
2. $aux, xub \notin Fact(w_1)$ for any $x \in A$.

Proposition

Let l_1, l_2 be the lengths of the shortest and second shortest words in S. If w is $\mathcal{I}$-compatible, then either $\mathcal{MF}(w) = \mathcal{MF}(w_1) \cap A^{l_1}$ or $\mathcal{MF}(w) = \mathcal{MF}(w_1) \cap A^{l_2}$.

Word Reconstruction

Let S be the set of words $aub \in \mathcal{MF}(w_1) \cap A^*$ such that:

1. $au\$, \$ub \in Fact(w_1)$;
2. $aux, xub \notin Fact(w_1)$ for any $x \in A$.

Proposition

Let l_1, l_2 be the lengths of the shortest and second shortest words in S. If w is $\mathcal{I}$-compatible, then either $\mathcal{MF}(w) = \mathcal{MF}(w_1) \cap A^{l_1}$ or $\mathcal{MF}(w) = \mathcal{MF}(w_1) \cap A^{l_2}$.

So, the algorithm is the following:
- Try with l_1: if the set $\mathcal{MF}(w_1) \cap A^{l_1}$ is the set of mff of a finite word, retrieve the word;
- otherwise, try with l_2: if the set $\mathcal{MF}(w_1) \cap A^{l_2}$ is the set of mff of a finite word, retrieve the word;
- otherwise, no $\mathcal{I}$-compatible word exists.

[1] M. Crochemore, F. Mignosi, A. Restivo. Automata and Forbidden Words. *Inform. Proc. Lett.* 67: 111–117, 1998.

[2] M. Crochemore, F. Mignosi, A. Restivo, S. Salemi. Text Compression Using Antidictionaries. *ICALP '99. Lecture Notes Comput. Sci.* 1644: 261–270, 1999.

[3] F. Mignosi, A. Restivo, M. Sciortino. Words and forbidden factors. *Theoret. Comput. Sci.* 273: 99–117, 2002.

[4] M.-P. Béal, M. Crochemore, F. Mignosi, A. Restivo, M. Sciortino. Computing forbidden words of regular languages. *Fundam. Inform.* 20: 1–15, 2003.

[5] M.-P. Béal, M. Crochemore, G. Fici. Presentations of Constrained Systems With Unconstrained Positions. *IEEE Trans. Inform. Theory* 51: 1891–1900, 2005.

[6] G. Fici, F. Mignosi, A. Restivo, M. Sciortino. Word Assembly through Minimal Forbidden Words. *Theoret. Comput. Sci.* 359: 214–230, 2006.

Thank You