Local recompression Word Equations and Beyond

Artur Jeż

Max Planck Institute for Informatics

21 June 2013

Word Equations

Definition

Given equation $U = V$, where $U, V \in (\Sigma \cup \mathcal{X})^*$.
Is there an assignment $S : \mathcal{X} \mapsto \Sigma^*$ satisfying the solution?

Word Equations

Definition

Given equation $U = V$, where $U, V \in (\Sigma \cup \mathcal{X})^*$.
Is there an assignment $S : \mathcal{X} \mapsto \Sigma^*$ satisfying the solution?

- Considered to be important
  - unification
  - equations in free semigroup
  - interesting in general
  - (helpful in equations in free group)
- . . . and hard

Word Equations

Definition

Given equation $U = V$, where $U, V \in (\Sigma \cup \mathcal{X})^*$.
Is there an assignment $S : \mathcal{X} \mapsto \Sigma^*$ satisfying the solution?

- Considered to be important
  - unification
  - equations in free semigroup
  - interesting in general
  - (helpful in equations in free group)
- . . . and hard

Is this decidable at all?

Makanin's algorithm

Makanin 1977

Rewriting procedure. Difficult termination.

Makanin's algorithm

Makanin 1977

Rewriting procedure. Difficult termination.

Improved over the years

- Jaffar [1990] Schulz [1990] 4-NEXPTIME
- Kościelski and Pacholski 3-NEXPTIME [1990]
- Diekert to 2-EXPSPACE [unpublished]
- Gutiérrez EXPSPACE [1998].

Makanin's algorithm

Makanin 1977

Rewriting procedure. Difficult termination.

Improved over the years

- Jaffar [1990] Schulz [1990] 4-NEXPTIME
- Kościelski and Pacholski 3-NEXPTIME [1990]
- Diekert to 2-EXPSPACE [unpublished]
- Gutiérrez EXPSPACE [1998].

Only NP-hard.

New approach

Theorem (Plandowski and Rytter, 1998)

Length minimal solution of length N is compressible *into* $\mathrm{poly}(\log N)$*.*
This yields a $\mathrm{poly}(n, \log N)$ *algorithm.*

New approach

Theorem (Plandowski and Rytter, 1998)

Length minimal solution of length N is compressible into $\text{poly}(\log N)$*.*
This yields a $\text{poly}(n, \log N)$ *algorithm.*

N is only known to be triply exponential (from Makanin's algorithm).

New approach

Theorem (Plandowski and Rytter, 1998)

Length minimal solution of length N is compressible *into* $\mathrm{poly}(\log N)$*.*
This yields a $\mathrm{poly}(n, \log N)$ *algorithm.*

N is only known to be triply exponential (from Makanin's algorithm).

Theorem (Plandowski 1999)

The size N of the minimal solution is at most doubly exponential.
This yields a NEXPTIME *algorithm.*

New approach

Theorem (Plandowski and Rytter, 1998)

Length minimal solution of length N is compressible into $\text{poly}(\log N)$. *This yields a* $\text{poly}(n, \log N)$ *algorithm.*

N is only known to be triply exponential (from Makanin's algorithm).

Theorem (Plandowski 1999)

The size N of the minimal solution is at most doubly exponential. This yields a NEXPTIME *algorithm.*

Theorem (Plandowski 1999)

PSPACE *algorithm.*

This talk

A simple and natural technique of local recompression.

This talk

A simple and natural technique of local recompression.

Yields a non-deterministic algorithm for word equations

- linear space (improving Plandowski PSPACE algorithm), NLinSPACE(n)
- poly($n, \log N$) time (improving Plandowski and Rytter algorithm)
- can be used to prove exponential bound on exponent of periodicity
- can be used to show the doubly-exponential bound on N
- can be easily generalised to generator of all solutions
- for one variable becomes deterministic and runs in $\mathcal{O}(n)$

Equality and Compression of Strings

$$a\ a\ a\ b\ a\ b\ c\ a\ b\ a\ b\ b\ a\ b\ c\ b\ a$$

$$a\ a\ a\ b\ a\ b\ c\ a\ b\ a\ b\ b\ a\ b\ c\ b\ a$$

Equality and Compression of Strings

$$a\ a\ a\ b\ a\ b\ c\ a\ b\ a\ b\ b\ a\ b\ c\ b\ a$$

$$a\ a\ a\ b\ a\ b\ c\ a\ b\ a\ b\ b\ a\ b\ c\ b\ a$$

Equality and Compression of Strings

$a_3 \quad b \; a \; b \; c \; a \; b \; a \; b \; b \; a \; b \; c \; b \; a$

$a_3 \quad b \; a \; b \; c \; a \; b \; a \; b \; b \; a \; b \; c \; b \; a$

Equality and Compression of Strings

$$a_3 \; b \; a \; b \; c \; a \; b \; a \; b_2 \; a \; b \; c \; b \; a$$

$$a_3 \; b \; a \; b \; c \; a \; b \; a \; b_2 \; a \; b \; c \; b \; a$$

Equality and Compression of Strings

$$a_3 \quad b \quad d \quad c \quad d \quad a \quad b_2 \quad d \quad c \; b \; a$$

$$a_3 \quad b \quad d \quad c \quad d \quad a \quad b_2 \quad d \quad c \; b \; a$$

Equality and Compression of Strings

$$a_3 \quad b \quad d \quad c \quad d \quad a \quad b_2 \quad d \quad c \quad e$$

$$a_3 \quad b \quad d \quad c \quad d \quad a \quad b_2 \quad d \quad c \quad e$$

Equality and Compression of Strings

$$a_3 \quad b \quad d \quad c \quad d \quad a \quad b_2 \quad d \quad c \quad e$$

$$a_3 \quad b \quad d \quad c \quad d \quad a \quad b_2 \quad d \quad c \quad e$$

Iterate!

Equality and Compression of Strings

$$a_3 \quad b \quad d \quad c \quad d \quad a \quad b_2 \quad d \quad c \quad e$$

$$a_3 \quad b \quad d \quad c \quad d \quad a \quad b_2 \quad d \quad c \quad e$$

Iterate!

Intuition: recompression

- Think of new letters as nonterminals of a grammar
- We build SLPs for both strings, bottom-up.
- Everything is compressed in the same way!

Compression

$P \leftarrow$ all pairs from $S(U)$, $L \leftarrow$ all letters from $S(U)$
for each $a \in L$ **do**
    replace each maximal block a^ℓ by a_ℓ ▷ A fresh letter
for each $ab \in P$ **do**
    replace each ab by c ▷ A fresh letter

Compression

$P \leftarrow$ all pairs from $S(U)$, $L \leftarrow$ all letters from $S(U)$
for each $a \in L$ **do**
    replace each maximal block a^ℓ by a_ℓ ▷ A fresh letter
for each $ab \in P$ **do**
    replace each ab by c ▷ A fresh letter

Lemma

Each subword shortens by a constant factor (U_i, V_j, $S(X)$, $S(U)$, $\dots$).

Proof.

Two consecutive letters: we tried to compress them;
fail: one is already compressed. □

Idea at work

Working example

$XbaYb = ba^3bab^2ab$ has a solution $S(X) = ba^3$, $S(Y) = b^2a$

Idea at work

Working example

$XbaYb = ba^3bab^2ab$ has a solution $S(X) = ba^3$, $S(Y) = b^2a$

We want to replace pair ba by a new letter c. Then

$$XbaYb = baaababbab \qquad \text{for } S(X) = baaa \; S(Y) = bba$$
$$XcYb = caacbcb \qquad \text{for } S(X) = caa \; S(Y) = bc$$

Idea at work

Working example

$XbaYb = ba^3bab^2ab$ has a solution $S(X) = ba^3$, $S(Y) = b^2a$

We want to replace pair ba by a new letter c. Then

$$XbaYb = baaababbab \qquad \text{for } S(X) = baaa\ S(Y) = bba$$
$$XcYb = caacbcb \qquad \text{for } S(X) = caa\ S(Y) = bc$$

And what about replacing ab by d?

$$XbaYb = baaababbab \qquad \text{for } S(X) = baaa\ S(Y) = bba$$

Idea at work

Working example

$XbaYb = ba^3bab^2ab$ has a solution $S(X) = ba^3$, $S(Y) = b^2a$

We want to replace pair ba by a new letter c. Then

$$XbaYb = baaababbab \qquad \text{for } S(X) = baaa \; S(Y) = bba$$
$$XcYb = caacbcb \qquad \text{for } S(X) = caa \; S(Y) = bc$$

And what about replacing ab by d?

$$XbaYb = baaababbab \qquad \text{for } S(X) = baaa \; S(Y) = bba$$

There is a problem with ‘crossing pairs’. We will fix!

Pair types

Definition (Pair types)

Appearance of ab is

- **explicit** it comes from U or V;
- **implicit** comes solely from $S(X)$;
- **crossing** in other case.

ab is crossing if it has a crossing appearance, non-crossing otherwise.

Pair types

Definition (Pair types)

Appearance of ab is

- **explicit** it comes from U or V;
- **implicit** comes solely from $S(X)$;
- **crossing** in other case.

ab is crossing if it has a crossing appearance, non-crossing otherwise.

$XbaYb = baaababbbab$ with $S(X) = baaa$ $S(Y) = bba$

- $baaababbbab$ $[XbaYb]$
- $baaababbbab$ $[XbaYb]$
- $baaababbbab$ $[XbaYb]$

Pair types

Definition (Pair types)

Appearance of ab is

- explicit: it comes from U or V;
- implicit: comes solely from $S(X)$;
- crossing: in other case.

ab is crossing if it has a crossing appearance, non-crossing otherwise.

$XbaYb = baaababbab$ with $S(X) = baaa$ $S(Y) = bba$

- $baaababbab$ $[XbaYb]$
- $baaababbab$ $[XbaYb]$
- $baaababbab$ $[XbaYb]$

Lemma (Length-minimal solutions)

If ab has an implicit appearance, then it has crossing or explicit one.

Compression of non-crossing pairs

PairComp

1: let $c \in \Sigma$ be an unused letter
2: replace each explicit ab in U and V by c

Compression of non-crossing pairs

PairComp

1: let $c \in \Sigma$ be an unused letter
2: replace each explicit ab in U and V by c

- $XbaYa = baaababbaa$ has a solution $S(X) = baaa$, $S(Y) = bba$
- ba is non-crossing
- $XcYa = caacbca$ has a solution $S(X) = caa$, $S(Y) = bc$

Lemma

The PairComp(a, b) *properly compresses noncrossing pairs.*

Lemma

The PairComp(a, b) *properly compresses noncrossing pairs.*

- transforms satisfiable to satisfiable,
- transforms unsatisfiable to unsatisfiable,

Lemma

The PairComp(a, b) *properly compresses noncrossing pairs.*

- transforms satisfiable to satisfiable,
- transforms unsatisfiable to unsatisfiable,

Proof.

Every ab in $S(U) = S(V)$ is replaced:

explicit pairs replaced explicitly

implicit pairs replaced implicitly (in the solution)

crossing there are none □

Dealing with crossing pairs

ab is a crossing pair

There is X such that $S(X) = bw$ and aX appears in $U = V$ (or symmetric).

Dealing with crossing pairs

ab is a crossing pair

There is X such that $S(X) = bw$ and aX appears in $U = V$ (or symmetric).

- replace X with bX
  (implicitly change solution $S(X) = bw$ to $S(X) = w$)

Dealing with crossing pairs

ab is a crossing pair

There is X such that $S(X) = bw$ and aX appears in $U = V$ (or symmetric).

- replace X with bX
  (implicitly change solution $S(X) = bw$ to $S(X) = w$)
- If $S(X) = \epsilon$ then remove X.

Dealing with crossing pairs

ab is a crossing pair

There is X such that $S(X) = bw$ and aX appears in $U = V$ (or symmetric).

- replace X with bX
  (implicitly change solution $S(X) = bw$ to $S(X) = w$)
- If $S(X) = \epsilon$ then remove X.

Lemma

After performing this for all variables, ab is no longer crossing.

Dealing with crossing pairs

ab is a crossing pair

There is X such that $S(X) = bw$ and aX appears in $U = V$ (or symmetric).

- replace X with bX
  (implicitly change solution $S(X) = bw$ to $S(X) = w$)
- If $S(X) = \epsilon$ then remove X.

Lemma

After performing this for all variables, ab is no longer crossing.

Compress the pair!

Example

- $XbaYb = baaababbbab$ for $S(X) = baaa$ $S(Y) = bba$
- ab is a crossing pair

Example

- $XbaYb = baaababbab$ for $S(X) = baaa$ $S(Y) = bba$
- ab is a crossing pair
- replace X with Xa, Y with bYa
  (new solution: $S(X) = baa$, $S(Y) = b$)
- $XababYab = baaababbab$ for $S(X) = baa$ $S(Y) = b$

Example

- $XbaYb = baaababbab$ for $S(X) = baaa$ $S(Y) = bba$
- ab is a crossing pair
- replace X with Xa, Y with bYa
  (new solution: $S(X) = baa$, $S(Y) = b$)
- $XababYab = baaababbab$ for $S(X) = baa$ $S(Y) = b$
- ab is not longer crossing, we replace it by c
- $XccYc = baaccbc$ for $S(X) = baa$ $S(Y) = b$

Maximal blocks

Definition (maximal block of a)

- When a^ℓ appears in $S(U) = S(V)$ and cannot be extended.
- Block appearance can be explicit, implicit or crossing.
- Letter a has crossing block if there is a crossing ℓ-block of a.

Maximal blocks

Definition (maximal block of a)

- When a^ℓ appears in $S(U) = S(V)$ and cannot be extended.
- Block appearance can be explicit, implicit or crossing.
- Letter a has crossing block if there is a crossing ℓ-block of a.

- Equivalents of pairs.
- Compress them similarly.
- Pop whole prefixes/suffixes, not single letters

Maximal blocks

Definition (maximal block of a)

- When a^ℓ appears in $S(U) = S(V)$ and cannot be extended.
- Block appearance can be explicit, implicit or crossing.
- Letter a has crossing block if there is a crossing ℓ-block of a.

- Equivalents of pairs.
- Compress them similarly.
- Pop whole prefixes/suffixes, not single letters

Lemma (Length-minimal solutions)

For maximal a^ℓ block: $\ell \leq 2^{cn}$.

Blocks compression

Definition (Crossing block)

maximal block is crossing iff
it is contained in $S(U)$ ($S(V)$) but not in explicit words nor in any $S(X)$.

Blocks compression

Definition (Crossing block)

maximal block is crossing iff
it is contained in $S(U)$ ($S(V)$) but not in explicit words nor in any $S(X)$.

When a has no crossing block

1: **for** all maximal blocks a^ℓ of a **do**
2: let $a_\ell \in \Sigma$ be a unused letter
3: replace each explicit maximal a^ℓ in $U = V$ by a_ℓ

What about crossing blocks?

Idea

- change the equation
- X defines $a^{\ell_X} w a^{r_X}$: change it to w
- replace X in equation by $a^{\ell_X} X a^{r_X}$

What about crossing blocks?

Idea

- change the equation
- X defines $a^{\ell_X} w a^{r_X}$: change it to w
- replace X in equation by $a^{\ell_X} X a^{r_X}$

CutPrefSuff(a)

for $X \in \mathcal{X}$ **do**
    guess and remove a-prefix a^{ℓ_i} and a-suffix a^{r_X} of $S(X)$
    replace each X in rules bodies by $a^{\ell_X} X a^{r_X}$

What about crossing blocks?

Idea

- change the equation
- X defines $a^{\ell_X} w a^{r_X}$: change it to w
- replace X in equation by $a^{\ell_X} X a^{r_X}$

CutPrefSuff(a)

for $X \in \mathcal{X}$ **do**
  guess and remove a-prefix a^{ℓ_i} and a-suffix a^{r_X} of $S(X)$
  replace each X in rules bodies by $a^{\ell_X} X a^{r_X}$

Lemma

After CutPrefSuff(a) *letter a has no crossing block.*

What about crossing blocks?

Idea

- change the equation
- X defines $a^{\ell_X} w a^{r_X}$: change it to w
- replace X in equation by $a^{\ell_X} X a^{r_X}$

CutPrefSuff(a)

1: **for** $X \in \mathcal{X}$ **do**
2: guess and remove a-prefix a^{ℓ_i} and a-suffix a^{r_X} of $S(X)$
3: replace each X in rules bodies by $a^{\ell_X} X a^{r_X}$

Lemma

After CutPrefSuff(a) *letter a has no crossing block.*

So a's blocks can be easily compressed.

What about crossing blocks?

Idea

- change the equation
- X defines $a^{\ell_X} w b^{r_X}$: change it to w
- replace X in equation by $a^{\ell_X} X b^{r_X}$

CutPrefSuff

for $X \in \mathcal{X}$ **do**
    let X begin with a and end with b
    calculate and remove a-prefix a^{ℓ_X} and b-suffix b^{r_X} of X
    replace each X in rules bodies by $a^{\ell_X} X b^{r_X}$

Lemma

After CutPrefSuff *no letter has a crossing block.*

So all blocks can be easily compressed.

Algorithm

while $U \notin \Sigma$ and $V \notin \Sigma$ **do**

  L $\leftarrow$ letters from $U = V$

  uncross the blocks

  for $a \in$ L **do**

    compress a blocks

Algorithm

while $U \notin \Sigma$ and $V \notin \Sigma$ **do**
  L $\leftarrow$ letters from $U = V$
  uncross the blocks
  for $a \in$ L **do**
    compress a blocks
  P $\leftarrow$ noncrossing pairs of letters from $U = V$ ▷ Guess
  P′ $\leftarrow$ crossing pairs of letters from $U = V$ ▷ Guess, only $\mathcal{O}(n)$
  for $ab \in$ P **do**
    compress pair ab
  for $ab \in$ P′ **do**
    uncross and compress pair ab

Crucial property

Theorem (Main property: shortens the solution)

Let ab be a string in $U = V$ or in $S(X)$ (for a length-minimal S). At least one of a, b is compressed in one phase.

Crucial property

Theorem (Main property: shortens the solution)

Let ab be a string in $U = V$ or in $S(X)$ (for a length-minimal S). At least one of a, b is compressed in one phase.

Proof.

$a = b$ By block compression.

$a \neq b$ Pair compression tries to compress ab.
Fails, when one was compressed already. □

Crucial property

Theorem (Main property: shortens the solution)

Let ab be a string in $U = V$ or in $S(X)$ (for a length-minimal S). At least one of a, b is compressed in one phase.

Proof.

$a = b$ By block compression.

$a \neq b$ Pair compression tries to compress ab.
Fails, when one was compressed already. □

Corollary (Running time)

The algorithm has $\mathcal{O}(\log N)$ phases.

Space consumption

Corollary (Space consumption)

The equation has length $\mathcal{O}(n^2)$.

Space consumption

Corollary (Space consumption)

The equation has length $\mathcal{O}(n^2)$.

Proof.

- we introduce $\mathcal{O}(n)$ letters per uncrossing
- $\mathcal{O}(n)$ uncrossings in one phase: $\mathcal{O}(n^2)$ new letters
- and we shorten it by a constant factor in each phase.

$$|U'| + |V'| \leq \frac{2}{3}(|U| + |V|) + cn^2$$

- Gives quadratic upper bound on the whole equation. □

Solution upper bound

Idea

- Running time is at most $(cn^2)^{cn^2}$.
- there are $\mathcal{O}(\log N)$ phases

So $\log N \sim (cn^2)^{cn^2}$.

Solution upper bound

Idea

- Running time is at most $(cn^2)^{cn^2}$.
- there are $\mathcal{O}(\log N)$ phases

So $\log N \sim (cn^2)^{cn^2}$.

Lemma

There are $\Omega(\log N)/\text{poly}(n)$ *phases*

Solution upper bound

Idea

- Running time is at most $(cn^2)^{cn^2}$.
- there are $\mathcal{O}(\log N)$ phases

So $\log N \sim (cn^2)^{cn^2}$.

Lemma

There are $\Omega(\log N)/\mathrm{poly}(n)$ *phases*

Proof.

We do not shorten too much (at most 2^{cn} letters into one). □

Solution upper bound

Idea

- Running time is at most $(cn^2)^{cn^2}$.
- there are $\mathcal{O}(\log N)$ phases

So $\log N \sim (cn^2)^{cn^2}$.

Lemma

There are $\Omega(\log N)/\text{poly}(n)$ *phases*

Proof.

We do not shorten too much (at most 2^{cn} letters into one). □

$$\log N/\text{poly}(n) \leq (cn^2)^{cn^2}$$

Linear space consumption

Aim at $\mathcal{O}(n)$ space consumption

- $\mathcal{O}(1)$ pair-uncrossing per variable
- smarter block compression

Improving pair compression

Partition of pairs

- Σ_ℓ and Σ_r are disjoint:
  we can compress pairs from $\Sigma_\ell \Sigma_r$ in parallel

Improving pair compression

Partition of pairs

- Σ_ℓ and Σ_r are disjoint:
  we can compress pairs from $\Sigma_\ell \Sigma_r$ in parallel
- choose partition that covers many appearances:
  think of random partition, it covers half of pairs in equation
- only $\mathcal{O}(1)$ uncrossing per pair

Improving pair compression

Partition of pairs

- Σ_ℓ and Σ_r are disjoint:
  we can compress pairs from $\Sigma_\ell \Sigma_r$ in parallel
- choose partition that covers many appearances:
  think of random partition, it covers half of pairs in equation
- only $\mathcal{O}(1)$ uncrossing per pair

This given $\mathcal{O}(n)$ long equation.

Block compression

Idea

- when we replace *a* blocks, only equality matters, not length
- pop $a^{\ell x}$ and b^{rx} from X but treat them as parameters

Block compression

Idea

- when we replace a blocks, only equality matters, not length
- pop $a^{\ell x}$ and b^{rx} from X but treat them as parameters
- guess the equal blocks
- check if they can be equal
- replace them

Block compression

Idea

- when we replace a blocks, only equality matters, not length
- pop a^{ℓ_X} and b^{r_X} from X but treat them as parameters
- guess the equal blocks
- check if they can be equal
- replace them

Length of a block

$$b\,a\,a\,\underbrace{a\,a\,b\,c\,c\,d\,a\,a}_{X}\,a\,a\,\underbrace{a\,a}_{Y}\,a\,a\,\underbrace{a\,a\,b\,c\,c\,d\,a\,a}_{X}\,a$$

Block compression

Idea

- when we replace a blocks, only equality matters, not length
- pop a^{ℓ_X} and b^{r_X} from X but treat them as parameters
- guess the equal blocks
- check if they can be equal
- replace them

Length of a block

$$b\,a\,a\,\underbrace{a\,a\,b\,c\,c\,d\,a\,a}_{X}\,a\,a\,\underbrace{a\,a}_{Y}\,a\,a\,\underbrace{a\,a\,b\,c\,c\,d\,a\,a}_{X}\,a$$

Linear combination of $\{\ell_X, r_X\}_{X \in \mathcal{X}}$ and constants.

Verification

Guessed equalities $\iff$ system of linear Diophantine equations in $\{\ell_X, r_X\}_{X \in \mathcal{X}}$

Verification

Guessed equalities $\iff$ system of linear Diophantine equations in $\{\ell_X, r_X\}_{X \in \mathcal{X}}$

- has size proportional to equation
  - encode variables as in the equation
  - encode constants in unary

Verification

Guessed equalities $\iff$ system of linear Diophantine equations in $\{\ell_X, r_X\}_{X \in \mathcal{X}}$

- has size proportional to equation
  - encode variables as in the equation
  - encode constants in unary
- can be verified in linear space (nondeterministically)
  - iteratively guess parity

Verification

Guessed equalities $\iff$ system of linear Diophantine equations in $\{\ell_X, r_X\}_{X \in \mathcal{X}}$

- has size proportional to equation
  - encode variables as in the equation
  - encode constants in unary
- can be verified in linear space (nondeterministically)
  - iteratively guess parity

Linear space.

Linear Space

- Equation is of length $\mathcal{O}(n)$.
- each letter may be different, so $\mathcal{O}(n \log n)$ bits
- want true linear space

Linear Space

- Equation is of length $\mathcal{O}(n)$.
- each letter may be different, so $\mathcal{O}(n \log n)$ bits
- want true linear space

Tools

- special encoding of letters: represented by fragments of the original equation
  - letters representing only original letters: appropriate tree
  - those representing other letters: depend only on XwY, encode them like that

Linear Space

- Equation is of length $\mathcal{O}(n)$.
- each letter may be different, so $\mathcal{O}(n \log n)$ bits
- want true linear space

Tools

- special encoding of letters: represented by fragments of the original equation
  - letters representing only original letters: appropriate tree
  - those representing other letters: depend only on XwY, encode them like that
- improve the pair compression (special pairing by Sakamoto)
- quite technical

Exponent of periodicity

Definition

$\text{per}(w) = k \iff u^k$ is a substring of w but ${u'}^{k+1}$ is not.
$\text{per}_\Sigma(w) = k \iff a^k$ is a substring of w but b^{k+1} is not.

Exponent of periodicity

Definition

$\text{per}(w) = k \iff u^k$ is a substring of w but ${u'}^{k+1}$ is not.
$\text{per}_\Sigma(w) = k \iff a^k$ is a substring of w but b^{k+1} is not.

- We do not fully use $\text{per}(S(U))$, only $\text{per}_\Sigma(S(U))$.
- $\text{per}_\Sigma(S(U))$ is the length of maximal block.

Exponent of periodicity

Definition

$\text{per}(w) = k \iff u^k$ is a substring of w but ${u'}^{k+1}$ is not.
$\text{per}_\Sigma(w) = k \iff a^k$ is a substring of w but b^{k+1} is not.

- We do not fully use $\text{per}(S(U))$, only $\text{per}_\Sigma(S(U))$.
- $\text{per}_\Sigma(S(U))$ is the length of maximal block.
- Those are (components of) solution of a Diophantine system in $\{\ell_X, r_X\}_{X \in \mathcal{X}}$
- They are at most exponential (standard algebra and analysis).
- So $\text{per}_\Sigma(S(U))$ is at most exponential.

Reducing per to per_Σ

Lemma

Consider $S(U)$ and $S(U')$ obtained by compression of pairs (or blocks). Let u^k be a substring of $S(U)$. Then either

Reducing per to per_Σ

Lemma

Consider $S(U)$ and $S(U')$ obtained by compression of pairs (or blocks). Let u^k be a substring of $S(U)$. Then either

- $u \in a^*$ *or*
- u'^{k-1} *is a substring of $S'(U')$ (for some u')*

Reducing per to per_Σ

Lemma

Consider $S(U)$ and $S(U')$ obtained by compression of pairs (or blocks). Let u^k be a substring of $S(U)$. Then either

- $u \in a^*$ *or*
- u'^{k-1} *is a substring of $S'(U')$ (for some u')*

Proof.

If u^k is not a block then compression does not affect u^k too much, u'^{k-1} can be chosen. □

Reducing per to per$_\Sigma$

Lemma

Consider $S(U)$ and $S(U')$ obtained by compression of pairs (or blocks). Let u^k be a substring of $S(U)$. Then either

- $u \in a^*$ *or*
- u'^{k-1} *is a substring of* $S'(U')$ *(for some* u'*)*

Proof.

If u^k is not a block then compression does not affect u^k too much, u'^{k-1} can be chosen. □

- There are $\mathcal{O}((cn)^{cn})$ compression steps.
- In each of them $\text{per}(U = V) = \text{per}_\Sigma(U = V)$ (exponential) or it drops by a constant.
- So $\text{per}(U = V)$ is at most exponential.

Univariate equations

Form of the equation $\mathcal{A} = \mathcal{B}$

$$A_0 X A_1 \dots A_{k-1} X A_k = X B_1 \dots B_{k-1} X B_k,$$

where $A_i, B_i \in \Sigma^*$, $A_0 \neq \epsilon$.

Univariate equations

Form of the equation $\mathcal{A} = \mathcal{B}$

$$A_0 X A_1 \dots A_{k-1} X A_k = X B_1 \dots B_{k-1} X B_k,$$

where $A_i, B_i \in \Sigma^*$, $A_0 \neq \epsilon$.

Nondeterminism dissappears

- only $S(X) \neq \epsilon$
- first (last) letter of $S(X)$ is known
- $S(X) \in a^*$ are easy to check;
- otherwise a-prefix of $S(X)$ and A_0 have the same length

Univariate equations

Form of the equation $\mathcal{A} = \mathcal{B}$

$$A_0 X A_1 \ldots A_{k-1} X A_k = X B_1 \ldots B_{k-1} X B_k,$$

where $A_i, B_i \in \Sigma^*$, $A_0 \neq \epsilon$.

Nondeterminism dissappears

- only $S(X) \neq \epsilon$
- first (last) letter of $S(X)$ is known
- $S(X) \in a^*$ are easy to check;
- otherwise a-prefix of $S(X)$ and A_0 have the same length

Whenever we pop, we test some solution.

Generating all solutions

We want a finite (graph-like) representation of all solutions.

Not all solutions are length minimal.

Generating all solutions

We want a finite (graph-like) representation of all solutions.

Not all solutions are length minimal.

Lemma (Plandowski)

It is enough to consider minimal solutions.

Minimal under homomorphism.

Generating all solutions

We want a finite (graph-like) representation of all solutions.

Not all solutions are length minimal.

Lemma (Plandowski)

It is enough to consider minimal solutions.

Minimal under homomorphism.

Lemma (Minimal solutions)

If ab has an implicit appearance in $S(U)$ for a minimal S then ab has crossing or explicit one.

Transforming solutions

Definition (Transforming solutions)

An operation changing $U = V$ to $U' = V'$ transforms solutions if we can associate an operator H with it such that

- when S' is a solution of $U' = V'$ then $S = H[S']$
- each S is of this form

Transforming solutions

Definition (Transforming solutions)

A nondeterministic operation changing $U = V$ to $U' = V'$ transforms solutions if
we can associate a family of operators $\mathcal{H}$ depending on choices with it such that

- when S' is a solution of $U' = V'$ then $S = H[S']$ for each $H \in \mathcal{H}$
- each S is of this form for some nondeterministic choices and some $H \in \mathcal{H}$

Transforming solutions

Definition (Transforming solutions)

An operation changing $U = V$ to $U' = V'$ transforms solutions if we can associate an operator H (depending on choices) with it such that

- when S' is a solution of $U' = V'$ then $S = H[S']$
- each S is of this form

Transforming solutions

Definition (Transforming solutions)

An operation changing $U = V$ to $U' = V'$ transforms solutions if we can associate an operator H (depending on choices) with it such that

- when S' is a solution of $U' = V'$ then $S = H[S']$
- each S is of this form

Lemma

All our operations transform solutions.
Operators are easy to define (morphisms).

Representation of solutions

Definition ($\mathcal{G}$)

- nodes: equations,
- edges: $U = V$ is transformed to $U' = V'$
- label: operator H transforming the solution

Representation of solutions

Definition ($\mathcal{G}$)

- nodes: equations,
- edges: $U = V$ is transformed to $U' = V'$
- label: operator H transforming the solution

- trivial equations have simple solutions
- each solution corresponds to a path in $\mathcal{G}$ and vice-versa

Representation of solutions

Definition ($\mathcal{G}$)

- nodes: equations,
- edges: $U = V$ is transformed to $U' = V'$
- label: operator H transforming the solution

- trivial equations have simple solutions
- each solution corresponds to a path in $\mathcal{G}$ and vice-versa

Construction

- Verify the nodes' existance.
- Verify edges' existance.
- Labels are natural to deduce from the algorithm.

PSPACE [Matching Plandowski's construction]

Open questions, related research, etc.

Also used for

- fully compressed membership problem for NFAs [in NP]
- fully compressed pattern matching [quadratic algorithm]
- approximation of the smallest grammar [simpler algorithm]
- . . . ?

Open questions, related research, etc.

Also used for

- fully compressed membership problem for NFAs [in NP]
- fully compressed pattern matching [quadratic algorithm]
- approximation of the smallest grammar [simpler algorithm]
- . . . ?

Open questions

- what about two variables (it is in P, but quite complicated)?
- is it in NP?
- is the solution at most exponential?