Compression and Word Equations

26.0 6.2013

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?

Working example

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

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?

Working example

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

Write $S(U)$, $S(V)$ with an obvious meaning, i.e.
$S(XbaYb) = S(X)baS(Y)b$.

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?

Working example

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

Write $S(U)$, $S(V)$ with an obvious meaning, i.e.
$S(XbaYb) = S(X)baS(Y)b$.

First attempts

- Markov: Hilbert 10th problem $\geq_r$ word equations.
- wanted to show undecidability

Why

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

Why

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

Is this decidable at all?

Makanin's algorithm

Makanin 1977

Rewriting procedure. Difficult termination.

Did not care about complexity.

Makanin's algorithm

Makanin 1977

Rewriting procedure. Difficult termination.

Did not care about complexity.

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- Diekert to 2-EXPSPACE [unpublished]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.

Did not care about complexity.

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- Diekert to 2-EXPSPACE [unpublished]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 $\operatorname{poly}(\log N)$. This yields a $\operatorname{poly}(n, \log N)$ algorithm.

A minimal solution has a LZ77 encoding of size $\mathcal{O}(n \log N)$.

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.*

A minimal solution has a LZ77 encoding of size $\mathcal{O}(n \log N)$.

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.

A minimal solution has a LZ77 encoding of size $\mathcal{O}(n \log N)$.

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 $\mathrm{poly}(\log N)$.
This yields a $\mathrm{poly}(n, \log N)$ *algorithm.*

A minimal solution has a LZ77 encoding of size $\mathcal{O}(n \log N)$.

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.*

Restricted cases

Simpler subcases

Some easier subcases (in P)?

Restricted cases

Simpler subcases

Some easier subcases (in P)?

Number of variables

- for two variables (currently best $\mathcal{O}(n^6)$ [Plandowski])
- for one variables (currently best $\mathcal{O}(n + \#_X \log n)$ [Plandowski & Dąbrowski]
- three variables: nothing is known (perhaps in NP, perhaps in P)

This lecture

A simple and natural technique of local recompression.

This lecture

A simple and natural technique of local recompression.

Yields a non-deterministic algorithm for word equations

- linear space
- poly($n, \log N$) time
- 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 CFGs 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 $\triangleright$ A fresh letter
for each $ab \in P$ **do**
    replace each ab by c $\triangleright$ 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, $S(X)$, $S(U)$, $\dots$).

Compression

```
P ← all pairs from S(U), L ← all letters from S(U)
for each a ∈ L do
    replace each maximal block a^ℓ by a_ℓ          ▷ A fresh letter
for each ab ∈ P do
    replace each ab by c                          ▷ A fresh letter
```

Lemma

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

Consider ab in U_i (or any other).
If $a=b$ then block compression replaces it
If $a \neq b$ then pair compression tries to compress it (as $ab \in P$)
Fails only when one was 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

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

And what about replacing ab by d?

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

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.

$XbaYb = baaababbab$ 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.

Lemma (Length-minimal solutions)

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

Let ab be an implicit pair in $S(U)$.
Suppose that no crossing nor explicit appearances.

So each ab in $S(U)$ (and $S(V)$) comes from some $S(X)$ (for some variable X).
Define new substitution S'.
$S'(X) =$ "$S(X)$ with all ab removed"
Then $S'(U) =$ "$S(U)$ with all ab rem."
$S'(V) =$ "$S(V)$ with all ab rem."
So $S'(U) = S'(V)$ and it is shorter.

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

If $U = V$ has a solution S such that ab is non-crossing then $\mathrm{PairComp}(a, b)$ *returns an equation $U' = V'$ with a solution S' such that $S(U')$ is obtained by replacing each ab by c in $S(U)$.*
If $a'b'$ appeared in $S(U)$ and was a non-crosing pair for S then it is for S'.

Lemma

If $U = V$ has a solution S such that ab is non-crossing then PairComp(a, b) *returns an equation $U' = V'$ with a solution S' such that $S(U')$ is obtained by replacing each ab by c in $S(U)$.*
If $a'b'$ appeared in $S(U)$ and was a non-crosing pair for S then it is for S'.

Let $U' = V'$ be the new eq.
Define $S'(X) = S(X)$ with each ab replaced by c.
Consider $S'(U')$ and an app. of ab in $S(U)$:
implicit : replaced
explicit : replaced
crossing : none by case assumption
Why $a'b'$ is non-crossing: no new app. of a', b' were introduced

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).

Pop(a, b)

for $X \in \mathcal{X}$ **do**
  if $S(X) = bw$ **then** ▷ Guess
    replace X with bX
      ▷ implicitly change solution $S(X) = bw$ to $S(X) = w$
  if $S(X) = \epsilon$ **then** ▷ Guess
    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).

Pop(a, b)

for $X \in \mathcal{X}$ **do**
  if $S(X) = bw$ **then** ▷ Guess
    replace X with bX
      ▷ implicitly change solution $S(X) = bw$ to $S(X) = w$
  if $S(X) = \epsilon$ **then** ▷ Guess
    remove X

Lemma

After Pop(a, b) *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).

Pop(a, b)

```
for X ∈ 𝒳 do
    if S(X) = bw then                                         ▷ Guess
        replace X with bX
                  ▷ implicitly change solution S(X) = bw to S(X) = w
    if S(X) = ε then                                          ▷ Guess
        remove X
```

Lemma

After Pop(a, b) *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 $S(X) = baa$ $S(Y) = b$

Lemma

If $U = V$ has a solution S such that ab is crossing then *after* $\mathrm{Pop}(a, b)$ *the returned equation $U' = V'$ has a solution S' such that $S(U) = S'(U')$ and ab is non crossing for S'.*

Lemma

If $U = V$ has a solution S such that ab is crossing ten after $\mathrm{Pop}(a, b)$ the returned equation $U' = V'$ has a solution S' such that $S(U) = S'(U')$ and ab is non crossing for S'.

Define $S'(X)$, such that $bS'(X)a = S(X)$ (or $bS'(X) = S(X)$, $S'(X)a = S(X)$, $S'(X) = S(X)$, depending on cases).
Then $S'(U') = S(U)$ and $S'(V') = S(V)$.

Suppose ab is crossing, say aX appears in $U' = V'$ and $S'(X)$ starts with b.

- we left-popped b from X: contradiction, as the letter to the left is $a \neq b$.
- we did not. Then the first letter of $S(X)$ and $S'(X)$ is the same and it is not 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

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

Lemma

If $U = V$ has a solution S such that a has no crossing blocks then BlockComp(a) *returns an equation $U' = V'$ with a solution S' such that $S(U')$ is obtained by replacing each a^ℓ by a_ℓ in $S(U)$.*

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

Lemma

If $U = V$ has a solution S such that a has no crossing blocks then BlockComp(a) *returns an equation $U' = V'$ with a solution S' such that $S(U')$ is obtained by replacing each a^ℓ by a_ℓ in $S(U)$.*

The same as for pair compression.

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 $a^{\ell_X} X_i a^{r_X}$

CutPrefSuff(a)

1: **for** $X \in \mathcal{X}$ **do**
2: guess and remove a-prefix a^{ℓ_X} and a-suffix a^{r_X} of $S(X)$
3: 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 $a^{\ell_X} X_i a^{r_X}$

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.*

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 $a^{\ell_X} X_i a^{r_X}$

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 $a^{\ell x} w b^{rx}$: change it to w
- replace X in equation by $a^{\ell x} X_i b^{rx}$

CutPrefSuff

for $X \in \mathcal{X}$ **do**
  let X begin with a and end with b
  calculate and remove a-prefix $a^{\ell x}$ and b-suffix b^{rx} of X
  replace each X by $a^{\ell x} X b^{rx}$

Lemma

After CutPrefSuff *no letter has a crossing block.*

So all blocks can be easily compressed.

Lemma

If $U = V$ has a solution S then after BlockComp(a) *the returned equation $U' = V'$ has a solution S' such that $S(U) = S'(U')$ and a has no crossing blocks for S'.*

Lemma

If $U = V$ has a solution S then after BlockComp(a) *the returned equation $U' = V'$ has a solution S' such that $S(U) = S'(U')$ and a has no crossing blocks for S'.*

Easier version (only popping as).

Let $S'(X)$ be such that $a^{\ell_X} S'(X) a^{r_X} = S(X)$.

Then $S(U) = S'(U')$ and $S(V) = S'(V')$.

The first letter of $S(X)$ is not "a",

so a cannot have a crossing block.

Slightly harder, when we pop a^{ℓ_X} and b^{r_X}.

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). For appropriat non-deterministic choices the returned equation $U' = V'$ has a solution S' such that at least one of a, b is compressed in it.

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). For appropriat non-deterministic choices the returned equation $U' = V'$ has a solution S' such that at least one of a, b is compressed in it.

Proof.

- $a = b$ By block compression.
- $a \neq b$ Pair compression tries to compress ab. Fails, when one was compressed already.

$\square$

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).
For appropriat non-deterministic choices the returned equation $U' = V'$ has a solution S' such that at least one of a, b is compressed in it.*

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)

For appropriate non-deterministic choices the equation has length $\mathcal{O}(n^2)$.

Space consumption

Corollary (Space consumption)

For appropriate non-deterministic choices the equation has length $\mathcal{O}(n^2)$.

How many letters are popped?

Block compression: $2n$ letters can be crossing (2 per variable).

For each we pop prefix and suffix: perhaps long, but we replace each by 1 letter

So $2n \times 2n = 4n^2$.

Pairs: $2n$ pairs (n app. of variables), for each $2n$ letters $2n \times 2n = 4n^2$.

$8n^2$

Space consumption

Corollary (Space consumption)

For appropriate non-deterministic choices the equation has length $\mathcal{O}(n^2)$.

$8n^2$ new letters.

Each block of letters is compressed (among 2 letters one is compressed).

So among 4 cons. letters a pair is compr.

$$|U'| + |V'| \leq \frac{3}{4}(|U| + |V|) + 8n^2.$$

Easy to show $\mathcal{O}(n^2)$ bound.

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 $\Omega(\log N)/\mathrm{poly}(n)$ *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). $\square$

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). □

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

Let S - len. min. solution, len. N

$(|S(u)| = N)$.

S' - sol. to which it was compressed

$$\frac{|S(u)|}{|S'(u')|} \leq c \cdot 2^{cn}$$

(pair and blocks)

Let S_1' - len. min. for $u' = v'$, len N'.

It was obtained from some S_1: sol of $u=v$,

$$\frac{|S_1(u)|}{|S_1'(u')|} \leq c \cdot 2^{cn}$$

$$\log_{c \cdot 2^{cn}} N \leq \text{numb. of ph.}$$

$$\vee\!/$$

$$\frac{|S(u)|}{|S_1'(u')|}$$

Block compression

Idea

- when we replace a blocks, only equality matters, not length
- pop a^{ℓ_X} and a^{r_X} 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 a^{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 a^{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 a^{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

Consider a maximal block.

Contains const. and popped pref./suff.

Each such pref./suffix is contained in one block

We encode l_X, r_X as $X + O(1)$ bits.

Constants in unary.

Size is the same

Verification :

for each var. guess its parity.
Replace x by $2x+b_x \Leftarrow$ parity bit.

check parity.
Divide by 2. $\rightarrow$ coefficient is the same.
If there are no const $\rightarrow$ YES.

Can be shown to use linear space
(When the sum of coefficient is c,
we add c in one round to const.,
but divide by 2. So $c + \frac{c}{2} + \frac{c}{4} + \dots$)

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 $\mathcal{A} = \mathcal{B}$

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

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

Whenever we pop, we test some solution.

1) If number of var. on sides differ, there is a unique candidate.

2) First letter: first of A_0

Last : of B_k or A_k

3) If $S(X) \notin a^*$ then we can calculate the a-prefix of $S(u)$ and $S(v)$. This yields the a-prefix length.
Also b-suffix

No non-determinism

4) We want $S(X) \neq \varepsilon$.
If we pop, this can be spoiled.
Before popping we make a verification.
A not ignore this S afterwords.