Local compression and Word Equations

Artur Jeż

MPI, Germany

28 February 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?

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

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?

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

- 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?

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

- 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?

Previous results

A. Markow '50 First investigations.

- Conjecture: undecidable.

Previous results

A. Markow '50 First investigations.

- Conjecture: undecidable.

G. Makanin '77 Satisfiability is decidable.

- Long, complicated, high complexity.
- improved for 20 years (Gutiérrez EXPSPACE '98)

Previous results

A. Markow '50 First investigations.

- Conjecture: undecidable.

G. Makanin '77 Satisfiability is decidable.

- Long, complicated, high complexity.
- improved for 20 years (Gutiérrez EXPSPACE '98)

W. Plandowski & W. Rytter '98 compression, poly$(n, \log N)$ algorithm

W. Plandowski '99 Doubly-exponential bound on N

W. Plandowski '99 PSPACE algorithm

- proof is difficult, but short.

Previous results

A. Markow '50 First investigations.

- Conjecture: undecidable.

G. Makanin '77 Satisfiability is decidable.

- Long, complicated, high complexity.
- improved for 20 years (Gutiérrez EXPSPACE '98)

W. Plandowski & W. Rytter '98 compression, poly($n, \log N$) algorithm

W. Plandowski '99 Doubly-exponential bound on N

W. Plandowski '99 PSPACE algorithm

- proof is difficult, but short.

Only NP-hard.

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

- $\mathcal{O}(n \log n)$ space

This talk

A simple and natural technique of local recompression.

Yields a non-deterministic algorithm for word equations

- $\mathcal{O}(n \log n)$ space
- shows doubly-exponential bound on N
- proves exponential bound on exponent of periodicity

This talk

A simple and natural technique of local recompression.

Yields a non-deterministic algorithm for word equations

- $\mathcal{O}(n \log n)$ space
- shows doubly-exponential bound on N
- proves exponential bound on exponent of periodicity
- can be easily generalised to generator of all solutions

This talk

A simple and natural technique of local recompression.

Yields a non-deterministic algorithm for word equations

- $\mathcal{O}(n \log n)$ space
- shows doubly-exponential bound on N
- proves exponential bound on exponent of periodicity
- can be easily generalised to generator of all solutions
- for $\mathcal{O}(1)$ variables runs in $\mathcal{O}(n)$ space (context-sensitive)

Idea

How to test equality of strings?

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

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

Idea

How to test equality of strings?

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

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

Idea

How to test equality of strings?

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

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

Idea

How to test equality of strings?

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

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

Idea

How to test equality of strings?

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

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

Idea

How to test equality of strings?

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

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

Idea

How to test equality of strings?

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

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

Idea

How to test equality 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!

Idea

For both words

- replace pairs of letters
- replace maximal blocks of letters

Every letter is replaced: length is halved.

Idea

For both words

- replace pairs of letters
- replace maximal blocks of letters

Every letter is replaced: length is halved.

while $U \notin \Sigma$ and $V \notin \Sigma$ **do**
  Letters $\leftarrow$ letters from $S(U) = S(V)$
  for $a \in$ Letters **do**
    replace maximal blocks a^ℓ with a_ℓ (fresh letter)
  Pairs $\leftarrow$ pairs of letters from $S(U) = S(V)$
  for $ab \in$ Pairs **do**
    replace appearances of ab with c (fresh letter)

Idea

For both words

- replace pairs of letters
- replace maximal blocks of letters

Every letter is replaced: length is halved.

while $U \notin \Sigma$ and $V \notin \Sigma$ **do**
  Letters $\leftarrow$ letters from $S(U) = S(V)$
  for $a \in$ Letters **do**
    replace maximal blocks a^ℓ with a_ℓ (fresh letter)
  Pairs $\leftarrow$ pairs of letters from $S(U) = S(V)$
  for $ab \in$ Pairs **do**
    replace appearances of ab with c (fresh letter)

How to do this for equations?

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

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

And what about replacing ab by d?

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

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.

A pair 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.

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

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

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

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.

A pair 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.
If a is the first (last) letter of $S(X)$ then it appears in $U = V$.

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

- $XbaYb = baaababbab$ 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: $\ell \leq 2^{cn}$.

Algorithm

while $U \notin \Sigma$ and $V \notin \Sigma$ **do**
  Letters $\leftarrow$ letters from $U = V$ without crossing block ▷ Guess
  Letters′ $\leftarrow$ letters from $U = V$ with crossing blocks ▷ Guess, $\mathcal{O}(n)$
  for $a \in$ Letters **do**
    compress a blocks
  for $a \in$ Letters′ **do**
    uncross and compress a blocks

Algorithm

while $U \notin \Sigma$ and $V \notin \Sigma$ **do**
  Letters $\leftarrow$ letters from $U = V$ without crossing block ▷ Guess
  Letters$'$ $\leftarrow$ letters from $U = V$ with crossing blocks ▷ Guess, $\mathcal{O}(n)$
  for $a \in$ Letters **do**
    compress a blocks
  for $a \in$ Letters$'$ **do**
    uncross and compress a blocks
  Pairs $\leftarrow$ noncrossing pairs of letters from $U = V$ ▷ Guess
  Pairs$'$ $\leftarrow$ crossing pairs of letters from $U = V$ ▷ Guess, only $\mathcal{O}(n)$
  for $ab \in$ Pairs **do**
    compress pair ab
  for $ab \in$ Pairs$'$ **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. □

Questions and related results

Also used for

- fully compressed membership problem for NFAs [in NP]
- fully compressed pattern matching [quadratic algorithm]
- approximation of the smallest grammar [simpler algorithm]
- $\mathcal{O}(n)$ algorithm for one variable [NEW!]
- ...

Questions and related results

Also used for

- fully compressed membership problem for NFAs [in NP]
- fully compressed pattern matching [quadratic algorithm]
- approximation of the smallest grammar [simpler algorithm]
- $\mathcal{O}(n)$ algorithm for one variable [NEW!]
- ...

Questions

- What about two variables (it is in P, but quite complicated)?
- Are word equations in NP?
- Are word equations context-sensitive?