Uniwersytet Wrocławski

# Recompression for Word Equations

Artur Jeż
University of Wrocław
Developments in Language Theory
Warsaw 08.08.2019

# DLT: always first

DLT 2007 my first conference

DLT 2013 my first invited talk

DLT 2019 my first life-achievement award

# DLT: always first

DLT 2007 my first conference

DLT 2013 my first invited talk on word equations/recompression

DLT 2019 my first life-achievement award

# DLT: always first

DLT 2006 my first rejection

DLT 2007 my first conference

DLT 2013 my first invited talk on word equations/recompression

DLT 2019 my first life-achievement award

# Word Equations

## Definition

Systems of equations $\{U_i = V_i\}_{i \in I}$, where $U_i, V_i \in (\Sigma \cup \mathcal{X})^*$.
Often just one equation.

# Word Equations

## Definition

Systems of equations $\{U_i = V_i\}_{i \in I}$, where $U_i, V_i \in (\Sigma \cup \mathcal{X})^*$.
Often just one equation.

Interest came from different sources.

- mathematics: free semigroup and its properties
- computer science: formalization tool

# Word Equations

## Definition

Systems of equations $\{U_i = V_i\}_{i \in I}$, where $U_i, V_i \in (\Sigma \cup \mathcal{X})^*$.
Often just one equation.

Interest came from different sources.

- mathematics: free semigroup and its properties
- computer science: formalization tool

## Natural questions

- satisfiability
- independence of the system
- nontriviality of solutions
- relations definability

# Word Equations

## Definition

Systems of equations $\{U_i = V_i\}_{i \in I}$, where $U_i, V_i \in (\Sigma \cup \mathcal{X})^*$.
Often just one equation.

Interest came from different sources.

- mathematics: free semigroup and its properties
- computer science: formalization tool

## Natural questions

- satisfiability
- independence of the system
- nontriviality of solutions
- relations definability

# Satisfiability

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

# Satisfiability

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

**Wrong**

Decidable [Makanin '77].

# Satisfiability and word combinatorics

## Makanin's algorithm

Heavily based on word combinatorics.

Exponent of periodicity, ...

## Makanin's algorithm

Heavily based on word combinatorics.
Exponent of periodicity, ...

Used previously:

- parametrization of solutions
- two variables

# Satisfiability and word combinatorics

## Makanin's algorithm

Heavily based on word combinatorics.
Exponent of periodicity, ...

Used previously:

- parametrization of solutions
- two variables

## Satisfiability

Progress, generalizations,

- better complexity (EXPSPACE)
- free groups, traces, ...
- regular constraints
- restricted cases
- ...

# Satisfiability and word combinatorics

## Makanin's algorithm

Heavily based on word combinatorics.
Exponent of periodicity, ...

Used previously:

- parametrization of solutions
- two variables

## Satisfiability

Progress, generalizations,

- better complexity (EXPSPACE)
- free groups, traces, ...
- regular constraints
- restricted cases
- ...

... but no breakthrough.

# Compression and word equations

## Plandowski & Rytter

A length-minimal solution of size $N$ a word equation of size $n$ has an SLP of size $poly(n, \log N)$.

# Compression and word equations

## Plandowski & Rytter

A length-minimal solution of size $N$ a word equation of size $n$ has an SLP of size $poly(n, \log N)$.

## Straight Line Program (SLP)

A context free grammar generating exactly one word.

# Compression and word equations

## Plandowski & Rytter

A length-minimal solution of size $N$ a word equation of size $n$ has an SLP of size $poly(n, \log N)$.

## Straight Line Program (SLP)

A context free grammar generating exactly one word.

## How to solve word equations?

Guess compressed representation of variables and verify the guess. Depends on external guarantees for $N$ (Makanin's algorithm).

# Compression and word equations

## Plandowski & Rytter

A length-minimal solution of size $N$ a word equation of size $n$ has an SLP of size $poly(n, \log N)$.

## Straight Line Program (SLP)

A context free grammar generating exactly one word.

## How to solve word equations?

Guess compressed representation of variables and verify the guess.
Depends on external guarantees for $N$ (Makanin's algorithm).

## Remark

Compress and compute paradigm turns out to be useful all around.

# Plandowski & Rytter: Idea

## Cut

$S : \mathcal{X} \to \Sigma^+$, extend to $S : (\mathcal{X} \cup \Sigma)^+ \to \Sigma^+$

$S(U)$: a solution word

# Plandowski & Rytter: Idea

## Cut

$S : \mathcal{X} \to \Sigma^+$, extend to $S : (\mathcal{X} \cup \Sigma)^+ \to \Sigma^+$

$S(U)$: a solution word

For $S(U)$ a cut is a position between a letter/variable and letter/variable.

## Cut

$S : \mathcal{X} \to \Sigma^+$, extend to $S : (\mathcal{X} \cup \Sigma)^+ \to \Sigma^+$

$S(U)$: a solution word

For $S(U)$ a cut is a position between a letter/variable and letter/variable.

$$\begin{array}{cccccccccccccccc} \left|a\right.\!\left|a\ a\ a\right.\!\left|a\ a\ a\right.\!\left|a\ a\ a\right| & & \left|a\ a\ a\right.\!\left|a\right.\!\left|a\ a\right.\!\left|a\ a\right.\!\left|a\ a\right| \\ a & X & X & X & = & X & a & Y & Y & Y \end{array}$$

# Plandowski & Rytter: Idea

## Cut

$S : \mathcal{X} \to \Sigma^+$, extend to $S : (\mathcal{X} \cup \Sigma)^+ \to \Sigma^+$

$S(U)$: a solution word

For $S(U)$ a cut is a position between a letter/variable and letter/variable.

$$\begin{array}{cccccccccccccccc} |a|a\ a\ a|a\ a\ a|a\ a\ a| & & |a\ a\ a|a|a\ a|a\ a|a\ a| \\ a & X & X & X & = & X & a & Y & Y & Y \end{array}$$

There are $\leq |U| + |V|$ cuts.

# Plandowski & Rytter: Idea

## Cut

$S : \mathcal{X} \to \Sigma^+$, extend to $S : (\mathcal{X} \cup \Sigma)^+ \to \Sigma^+$

$S(U)$: a solution word

For $S(U)$ a cut is a position between a letter/variable and letter/variable.

$$\begin{array}{ccccccccccccccc} |a|a\ a\ a|a\ a\ a|a\ a\ a| & & |a\ a\ a|a|a\ a|a\ a|a\ a| \\ a & X & X & X & = & X & a & Y & Y & Y \end{array}$$

There are $\leq |U| + |V|$ cuts.

## Main observation

If $S$ is length-minimal and $w$ occurs in $S(u)$, then it "touches" a cut. (i.e. the cut is inside or at one of the ends)

# Plandowski & Rytter: Idea

## Cut

$S : \mathcal{X} \to \Sigma^+$, extend to $S : (\mathcal{X} \cup \Sigma)^+ \to \Sigma^+$

$S(U)$: a solution word

For $S(U)$ a cut is a position between a letter/variable and letter/variable.

$$\begin{array}{cccccccccccccccc} |a|a\ a\ a|a\ a\ a|a\ a\ a| & & |a\ a\ a|a|a\ a|a\ a|a\ a| \\ a & X & X & X & = & X & a & Y & Y & Y \end{array}$$

There are $\leq |U| + |V|$ cuts.

## Main observation

If $S$ is length-minimal and $w$ occurs in $S(u)$, then it "touches" a cut. (i.e. the cut is inside or at one of the ends)
Otherwise the word could be removed.

# Plandowski & Rytter proof

For a cut $\alpha$ let $(\alpha)_k$: the word $2^{k-1}$ to the left and right of the cut.

# Plandowski & Rytter proof

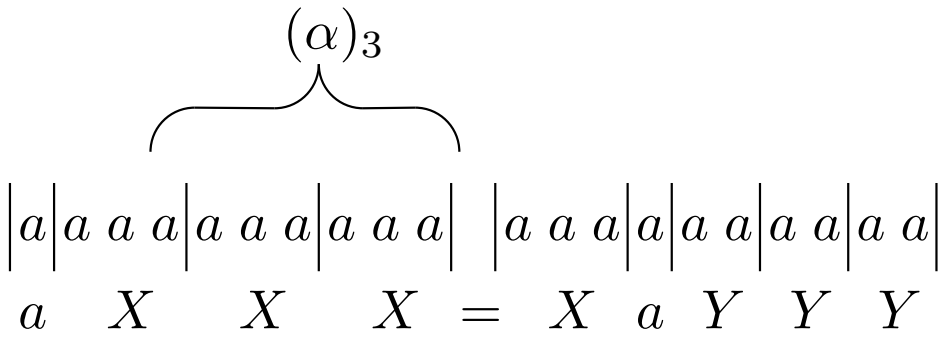
For a cut $\alpha$ let $(\alpha)_k$: the word $2^{k-1}$ to the left and right of the cut.

$$\overbrace{|a|a\ a\ a|a\ a\ a|a\ a\ a|}^{(\alpha)_3}\ |a\ a\ a|a|a\ a|a\ a|a\ a|$$

$$a\quad X\quad X\quad X\ =\ X\ a\ Y\ Y\ Y$$

# Plandowski & Rytter proof

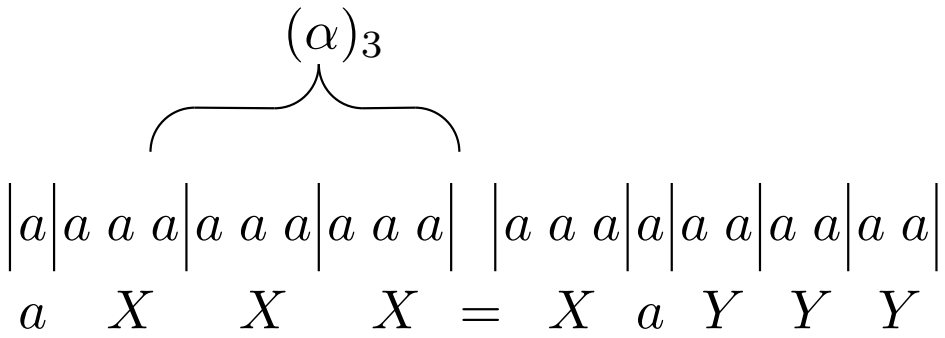
For a cut $\alpha$ let $(\alpha)_k$: the word $2^{k-1}$ to the left and right of the cut.




$$(\alpha)_{k+1} = w(\alpha_k)w', \quad |w|, |w'| \leq 2^{k-1}$$

# Plandowski & Rytter proof

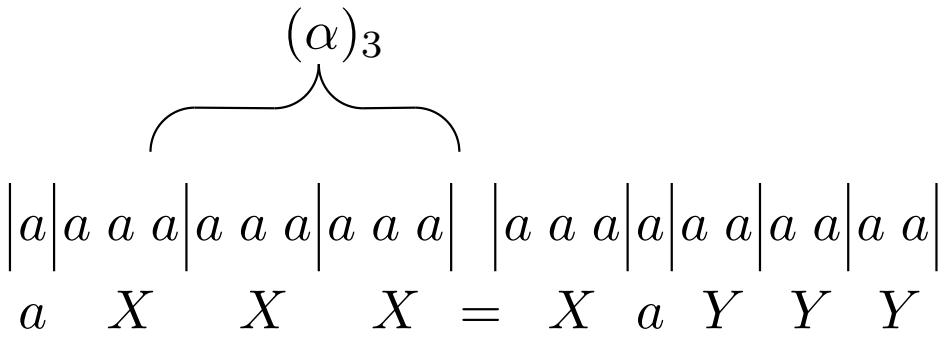
For a cut $\alpha$ let $(\alpha)_k$: the word $2^{k-1}$ to the left and right of the cut.




$$(\alpha)_{k+1} = w(\alpha_k)w', \quad |w|, |w'| \leq 2^{k-1}$$
$$w = (\beta)_k[i\,..\,j]$$
$$w' = (\beta')_k[i'\,..\,j']$$

# Plandowski & Rytter proof

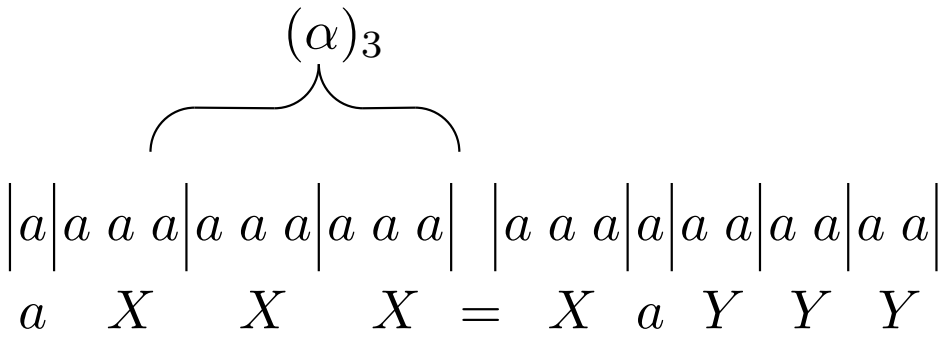
For a cut $\alpha$ let $(\alpha)_k$: the word $2^{k-1}$ to the left and right of the cut.




$$(\alpha)_{k+1} = w(\alpha_k)w', \quad |w|, |w'| \leq 2^{k-1}$$
$$w = (\beta)_k[i\,..\,j]$$
$$w' = (\beta')_k[i'\,..\,j']$$
$$\Longrightarrow\ (\alpha)_{k+1} = (\beta)_k[i\,..\,j](\alpha_k)(\beta')_k[i'\,..\,j']$$

# Plandowski & Rytter proof

For a cut $\alpha$ let $(\alpha)_k$: the word $2^{k-1}$ to the left and right of the cut.




$$(\alpha)_{k+1} = w(\alpha_k)w', \quad |w|, |w'| \leq 2^{k-1}$$
$$w = (\beta)_k[i\,..\,j]$$
$$w' = (\beta')_k[i'\,..\,j']$$
$$\Longrightarrow\ (\alpha)_{k+1} = (\beta)_k[i\,..\,j](\alpha_k)(\beta')_k[i'\,..\,j']$$

Almost an SLP: can be transformed (quadratic size increase).

# Plandowski & Rytter proof

For a cut $\alpha$ let $(\alpha)_k$: the word $2^{k-1}$ to the left and right of the cut.

$$\overbrace{\big|a\big|a\ a\ a\big|a\ a\ a\big|a\ a\ a\big|}^{(\alpha)_3}\ \big|a\ a\ a\big|a\big|a\ a\big|a\ a\big|a\ a\big|$$
$$a\quad X\quad X\quad X\ =\ X\ a\ Y\ Y\ Y$$

$$(\alpha)_{k+1} = w(\alpha_k)w', \quad |w|, |w'| \leq 2^{k-1}$$
$$w = (\beta)_k[i\,..\,j]$$
$$w' = (\beta')_k[i'\,..\,j']$$
$$\Longrightarrow\ (\alpha)_{k+1} = (\beta)_k[i\,..\,j](\alpha_k)(\beta')_k[i'\,..\,j']$$

Almost an SLP: can be transformed (quadratic size increase).
Size $((|U| + |V|)\log N)^2$.

# Towards PSPACE

**Theorem (Plandowski 1999)**

*Length-minimal solution is at most doubly exponential.*

# Towards PSPACE

**Theorem (Plandowski 1999)**

*Length-minimal solution is at most doubly exponential.*
*Via: word factorization, tailored for* $(\alpha)_k$.

# Towards PSPACE

**Theorem (Plandowski 1999)**

*Length-minimal solution is at most doubly exponential.*
*Via: word factorization, tailored for* $(\alpha)_k$.

**Theorem (Plandowski 1999)**

*Satisfiability of word equations is in PSPACE.*

# Towards PSPACE

**Theorem (Plandowski 1999)**

*Length-minimal solution is at most doubly exponential.*
*Via: word factorization, tailored for* $(\alpha)_k$.

**Theorem (Plandowski 1999)**

*Satisfiability of word equations is in PSPACE.*
*Via: even better factorization and merging together.*

# What next

State of the art for 20 years.
Extended (free groups, regular constraints, representation of all solutions...)

# What next

State of the art for 20 years.
Extended (free groups, regular constraints, representation of all solutions...)

The approach of compress and verify used independently.

# What next

State of the art for 20 years.
Extended (free groups, regular constraints, representation of all solutions...)

The approach of compress and verify used independently.

## In the retrospect

We prove the existence of the SLP by constructing it top-down.

# What next

State of the art for 20 years.
Extended (free groups, regular constraints, representation of all solutions...)

The approach of compress and verify used independently.

## In the retrospect

We prove the existence of the SLP by constructing it top-down.

## Why not bottom-up?

There has to be a rule $X \to ab$.
We can build SLP bottom-up.

# Building SLP bottom-up

Build SLP bottom-up for the solution.

- choose a pair $ab$ and replace it in the solution word
  There are only $\mathcal{O}(|U| + |V|)$ of them.

# This is not how it happened

## Turku 2011

I gave a talk on *Fully Compressed Membership problem for NFAs*.

# This is not how it happened

## Turku 2011

I gave a talk on *Fully Compressed Membership problem for NFAs*.

Problem stated by Plandowski & Rytter in *Jewels are forever*
In honor of Arto Salomaa.

# This is not how it happened

## Turku 2011

I gave a talk on *Fully Compressed Membership problem for NFAs*.

Problem stated by Plandowski & Rytter in *Jewels are forever*
In honor of Arto Salomaa.

Approach influenced by Mehlhorn, Sundar, Uhrig *Maintaining Dynamic Sequences under Equality Tests in Polylogarithmic Time.*

# This is not how it happened

## Turku 2011

I gave a talk on *Fully Compressed Membership problem for NFAs*.

Problem stated by Plandowski & Rytter in *Jewels are forever*
In honor of Arto Salomaa.

Approach influenced by Mehlhorn, Sundar, Uhrig *Maintaining Dynamic Sequences under Equality Tests in Polylogarithmic Time.*

Juhani: connected with Plandowski & Rytter paper.

# This is not how it happened

## Turku 2011

I gave a talk on *Fully Compressed Membership problem for NFAs*.

Problem stated by Plandowski & Rytter in *Jewels are forever*
In honor of Arto Salomaa.

Approach influenced by Mehlhorn, Sundar, Uhrig *Maintaining Dynamic Sequences under Equality Tests in Polylogarithmic Time.*

Juhani: connected with Plandowski & Rytter paper.

I read it later and realized it can be done simpler.

# Building SLP bottom-up

- A solution word for length-minimal solution
- factorization: chosen substrings + letters—factors (size: number of factors)
- original cuts ma be inside

# Building SLP bottom-up

- A solution word for length-minimal solution
- factorization: chosen substrings + letters—factors (size: number of factors)
- original cuts ma be inside

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

# Building SLP bottom-up

- A solution word for length-minimal solution
- factorization: chosen substrings + letters—factors (size: number of factors)
- original cuts ma be inside

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

# Building SLP bottom-up

- A solution word for length-minimal solution
- factorization: chosen substrings + letters—factors (size: number of factors)
- original cuts ma be inside

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

- choose a pair of factors and replace all their occurrences

# Building SLP bottom-up

- A solution word for length-minimal solution
- factorization: chosen substrings + letters—factors
  (size: number of factors)
- original cuts ma be inside

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

- choose a pair of factors and replace all their occurrences
- If there are $\leq p(|U| + |V|)$ ($p$—polynomial) different pairs of factors:
  some shortens factorization size by fraction $\frac{1}{p(|U|+|V|)}$.

# Building SLP bottom-up

- A solution word for length-minimal solution
- factorization: chosen substrings + letters—factors
  (size: number of factors)
- original cuts ma be inside

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

- choose a pair of factors and replace all their occurrences
- If there are $\leq p(|U| + |V|)$ ($p$—polynomial) different pairs of factors:
  some shortens factorization size by fraction $\frac{1}{p(|U|+|V|)}$.
- After $p(|U| + |V|)$: $\left(1 - \frac{1}{p(|U|+|V|)}\right)^{p(|U|+|V|)} \approx \frac{1}{e}$.

# Building SLP bottom-up

- A solution word for length-minimal solution
- factorization: chosen substrings + letters—factors
  (size: number of factors)
- original cuts ma be inside

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

- choose a pair of factors and replace all their occurrences
- If there are $\leq p(|U|+|V|)$ ($p$—polynomial) different pairs of factors:
  some shortens factorization size by fraction $\frac{1}{p(|U|+|V|)}$.
- After $p(|U|+|V|)$: $\left(1-\frac{1}{p(|U|+|V|)}\right)^{p(|U|+|V|)} \approx \frac{1}{e}$.
- SLP size: $\mathcal{O}(p(|U|+|V|)\log N)$.

# Building SLP bottom-up

- A solution word for length-minimal solution
- factorization: chosen substrings + letters—factors (size: number of factors)
- original cuts ma be inside

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

- choose a pair of factors and replace all their occurrences
- If there are $\leq p(|U|+|V|)$ ($p$—polynomial) different pairs of factors: Not so easy
  some shortens factorization size by fraction $\frac{1}{p(|U|+|V|)}$.
- After $p(|U|+|V|)$: $\left(1-\frac{1}{p(|U|+|V|)}\right)^{p(|U|+|V|)} \approx \frac{1}{e}$.
- SLP size: $\mathcal{O}(p(|U|+|V|)\log N)$.

# Slightly different (and in PSPACE)

- Proceed iteratively
- modify the equation: factors are the new letters
- shorten the variables (was already in Plandowski's paper)

# Slightly different (and in PSPACE)

- Proceed iteratively
- modify the equation: factors are the new letters
- shorten the variables (was already in Plandowski's paper)
- problematic bound trivializes

# Replacing pairs

## PairComp$(a, b)$

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

# Replacing pairs

## PairComp$(a, b)$

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

### Working example

$XbaYb = baaababbbab$ has a solution $S(X) = baaa$, $S(Y) = bba$

# Replacing pairs

## PairComp$(a, b)$

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

### Working example

$XbaYb = baaababbbab$ has a solution $S(X) = baaa$, $S(Y) = bba$

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

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

# Replacing pairs

## PairComp$(a, b)$

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

### Working example

$XbaYb = baaababbbab$ has a solution $S(X) = baaa$, $S(Y) = bba$

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

$$Xba Yb = baaababbbab \quad \text{for } S(X) = baa \; S(Y) = bba$$
$$X\,c\,Yb = c\,aa\,c\,b\,c\,b \quad \text{for } S'(X) = caa \;\; S'(Y) = bc$$

# Replacing pairs

## PairComp$(a, b)$

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

### Working example

$XbaYb = baaababbbab$ has a solution $S(X) = baaa$, $S(Y) = bba$

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

$$XbaYb{=}baaababbbab \quad \text{for } S(X) = baaa\ S(Y) = bba$$
$$X\ c\ Yb{=}\ c\ aa\ c\ b\ c\ b \quad \text{for } S'(X) = caa\ \ S'(Y) = bc$$

And what about replacing $ab$ by $d$?

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

# Pair types

## Definition (Pair types)

Occurrence of $ab$, $a \neq b$, in a solution word (so for a fixed solution) is

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

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

# Pair types

## Definition (Pair types)

Occurrence of $ab$, $a \neq b$, in a solution word (so for a fixed solution) is

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

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

$$X\ \ baa\ \ Y\ \ b = baaabaabbbab \qquad S(X) = baaa\ S(Y) = bba$$

# Pair types

## Definition (Pair types)

Occurrence of $ab$, $a \neq b$, in a solution word (so for a fixed solution) is

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

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

$$X\ baa\ Y\ b = baaabaabbbab \qquad S(X) = baaa\ S(Y) = bba$$

$$baaa\ baa\ bba\ b = baaabaabbbab \quad \text{explicit}$$

$$baaa\ baa\ bba\ b = baaabaabbbab \quad \text{implicit}$$

$$baaa\ baa\ bba\ b = baaabaabbbab \quad \text{crossing}$$

# Correctness

**Lemma**

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

# Correctness

## Lemma

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

complete if the old equation has a solution then the new one has

sound if the new equation has a solution then the old one has

# Correctness.

## Complete

$S'(U')$ is $S(U)$ with every $ab$ replaced; similarly $S'(V')$:

explicit pairs replaced explicitly

implicit pairs replaced implicitly (in the solution)

crossing there are none

# Correctness.

## Complete

$S'(U')$ is $S(U)$ with every $ab$ replaced; similarly $S'(V')$:

explicit pairs replaced explicitly

implicit pairs replaced implicitly (in the solution)

crossing there are none

$X\ \ baa\ Y\ b{=}baaabaabbbab \quad S(X) = baaa\ S(Y) = bba$

$baaabaabbbab{=}baaabaabbbab$

# Correctness.

## Complete

$S'(U')$ is $S(U)$ with every $ab$ replaced; similarly $S'(V')$:

explicit pairs replaced explicitly

implicit pairs replaced implicitly (in the solution)

crossing there are none

$$X\ baa\ Y\ b{=}baaabaabbab \quad S(X) = baaa\ S(Y) = bba$$
$$baaabaabbab{=}baaabaabbab$$
$$c\,aa\ c\,ab\,c\,b{=}\,c\,aa\,c\,ab\,c\,b$$
$$X\ c\,a\,Y\ b{=}\,c\,aa\,c\,ab\,c\,b \quad S'(X) = caa\ S'(Y) = bc$$

# Correctness.

## Complete

$S'(U')$ is $S(U)$ with every $ab$ replaced; similarly $S'(V')$:

explicit pairs replaced explicitly

implicit pairs replaced implicitly (in the solution)

crossing there are none

## Sound

If the new equation is satisfiable: roll back the changes.

# Correctness.

## Complete

$S'(U')$ is $S(U)$ with every $ab$ replaced; similarly $S'(V')$:

explicit pairs replaced explicitly

implicit pairs replaced implicitly (in the solution)

crossing there are none

## Sound

If the new equation is satisfiable: roll back the changes.

$$c\,aa\;\; c\;ab\;c\;b = c\;aa\;c\;ab\;c\;b$$
$$X\;\;\; c\;a\;Y\;b = c\;aa\;c\;ab\;c\;b \qquad S'(X) = caa\;S'(Y) = bc$$

# Correctness.

## Complete

$S'(U')$ is $S(U)$ with every $ab$ replaced; similarly $S'(V')$:

explicit pairs replaced explicitly

implicit pairs replaced implicitly (in the solution)

crossing there are none

## Sound

If the new equation is satisfiable: roll back the changes.

$$S(X) = baaa \; S(Y) = bba$$

$$c\,aa \;\; c\,ab\,c\,b = c\,aa\,c\,ab\,c\,b$$

$$X \;\; c\,a\,Y\,b = c\,aa\,c\,ab\,c\,b \qquad S'(X) = caa \; S'(Y) = bc$$

# Correctness.

## Complete

$S'(U')$ is $S(U)$ with every $ab$ replaced; similarly $S'(V')$:

explicit pairs replaced explicitly

implicit pairs replaced implicitly (in the solution)

crossing there are none

## Sound

If the new equation is satisfiable: roll back the changes.

$$X\ baa\ Y\ b{=}baaabaabbbab \qquad S(X) = baaa\ S(Y) = bba$$
$$baaabaabbbab{=}baaabaabbbab$$
$$c\,aa\ \ c\,ab\,c\,b{=}\ c\,aa\,c\,ab\,c\,b$$
$$X\ \ c\,a\,Y\ b{=}\ c\,aa\,c\,ab\,c\,b \qquad S'(X) = caa\ S'(Y) = bc$$

# Dealing with crossing pairs

$ab$ is a crossing pair

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

# Dealing with crossing pairs

$ab$ is a crossing pair

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

## Uncrossing$(a, b)$

1: **for** $X \in \mathcal{X}$ **do**
2: **if** first letter of $S(X)$ is $b$ **then**
3: replace each occurrence of $X$ by $bX$ ▷ Pop
▷ Change $S$ accordingly
4: **if** $S(X) = \epsilon$ **then** remove $X$ from the equation
5: ▷ perform symmetrically for the last letter and $a$

# Dealing with crossing pairs

$ab$ is a crossing pair

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

## Uncrossing$(a, b)$

1: **for** $X \in \mathcal{X}$ **do**
2: **if** first letter of $S(X)$ is $b$ **then**
3: replace each occurrence of $X$ by $bX$ ▷ Pop
▷ Change $S$ accordingly
4: **if** $S(X) = \epsilon$ **then** remove $X$ from the equation
5: ▷ perform symmetrically for the last letter and $a$

## Lemma

*After uncrossing $ab$ is no longer crossing.*

# Dealing with crossing pairs

$ab$ is a crossing pair

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

## Uncrossing$(a, b)$

1: **for** $X \in \mathcal{X}$ **do**
2: **if** first letter of $S(X)$ is $b$ **then**
3: replace each occurrence of $X$ by $bX$ ▷ Pop
▷ Change $S$ accordingly
4: **if** $S(X) = \epsilon$ **then** remove $X$ from the equation
5: ▷ perform symmetrically for the last letter and $a$

## Lemma

*After uncrossing $ab$ is no longer crossing.*

We can compress it.

# Uncrossing: example

## Uncrossing $ab$

$$X\ \ baa\ \ Y\ \ b{=}baaabaabbbab \quad S(X) = baaa\ S(Y) = bba$$

# Uncrossing: example

## Uncrossing $ab$

$$X\ ba\ Y\ b{=}baaabaabbbab \quad S(X) = baaa\ S(Y) = bba$$
$$baaabaa\ bba\ b{=}baaabaabbbab$$

# Uncrossing: example

## Uncrossing $ab$

$$X\ ba\ Y\ b{=}baaabaabbbab \qquad S(X) = baaa\ S(Y) = bba$$
$$baaabaa\ bba\ b{=}baaabaabbbab$$

$$bXabaabYab{=}baaabaabbbab \qquad S'(X) = aa\ S'(Y) = b$$

# Uncrossing: example

## Uncrossing $ab$

$$X\ baa\ Y\ b{=}baaabaabbbab \qquad S(X) = baaa\ S(Y) = bba$$
$$baaabaa\,bba\,b{=}baaabaabbbab$$
$$baaabaaab\ ba\,b{=}baaabaabbbab$$
$$bXabaabYab{=}baaabaabbbab \qquad S'(X) = aa\ S'(Y) = b$$

# Maximal blocks

## Maximal block

$a^k$ is a maximal block in $w$ if it cannot be extended.

# Maximal blocks

## Maximal block

$a^k$ is a maximal block in $w$ if it cannot be extended.

For a solution $S$ there are $\mathcal{O}(|U| + |V|)$ different maximal blocks in $S(U)$.

# Maximal blocks

## Maximal block

$a^k$ is a maximal block in $w$ if it cannot be extended.

For a solution $S$ there are $\mathcal{O}(|U| + |V|)$ different maximal blocks in $S(U)$.

- Block occurrence can be explicit, implicit or crossing.
- Letter $a$ is crossing (has a crossing block) if there is a crossing block of $a$.

# Maximal blocks

## Maximal block

$a^k$ is a maximal block in $w$ if it cannot be extended.

For a solution $S$ there are $\mathcal{O}(|U| + |V|)$ different maximal blocks in $S(U)$.

- Block occurrence can be explicit, implicit or crossing.
- Letter $a$ is crossing (has a crossing block) if there is a crossing block of $a$.

$$X \quad baa\, Y\, b = baabbaabbb \qquad S(X) = baab\; S(Y) = bb$$
$$baab\, baa\, bb\, b = baabbaabbb$$

# Maximal blocks

## Maximal block

$a^k$ is a maximal block in $w$ if it cannot be extended.

For a solution $S$ there are $\mathcal{O}(|U| + |V|)$ different maximal blocks in $S(U)$.

- Block occurrence can be explicit, implicit or crossing.
- Letter $a$ is crossing (has a crossing block) if there is a crossing block of $a$.

$$X \quad baa\, Y\, b = baabbaabbb \qquad S(X) = baab\; S(Y) = bb$$
$$baab\, baa\, bb\, b = baabbaabbb$$

## Lemma (Length-minimal solutions)

*If $a^\ell$ is a maximal block in a length-minimal solution of $u = v$ then $\ell \leq 2^{c|uv|}$.*

# Blocks compression

## When $a$ has no crossing block

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

# Blocks compression

## When $a$ has no crossing block

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

## Lemma

*The* ChainNCr$(a)$ *properly compresses noncrossing blocks of* $a$.

# Blocks compression

## When $a$ has no crossing block

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

## Lemma

*The* ChainNCr$(a)$ *properly compresses noncrossing blocks of* $a$.

$$X \;\; baaYbaaa{=}baabbaabbbaaa \qquad S(X) = baab \; S(Y) = bb$$

# Blocks compression

## When $a$ has no crossing block

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

## Lemma

*The* ChainNCr$(a)$ *properly compresses noncrossing blocks of* $a$.

$X\ \ baaYbaaa{=}baabbaabbbaaa \qquad S(X) = baab\ S(Y) = bb$
$baabbaabbbaaa{=}baabbaabbbaaa$

# Blocks compression

## When $a$ has no crossing block

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

## Lemma

*The* ChainNCr$(a)$ *properly compresses noncrossing blocks of* $a$.

$$X\ \ baaYbaaa{=}baabbaabbbaaa \qquad S(X) = baab\ S(Y) = bb$$
$$baabbaabbbaaa{=}baabbaabbbaaa$$

$$X\ \ ba_2Yb\ a_3\ {=}ba_2bba_2bbb\ a_3 \qquad S'(X) = ba_2b\ S'(Y) = bb$$

# Blocks compression

## When $a$ has no crossing block

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

## Lemma

*The* ChainNCr$(a)$ *properly compresses noncrossing blocks of* $a$.

$$X\ \ baaYbaaa{=}baabbaabbbaaa \qquad S(X) = baab\ S(Y) = bb$$
$$baabbaabbbaaa{=}baabbaabbbaaa$$
$$ba_2bba_2bbb\ a_3 {=} ba_2bba_2bbb\ a_3$$
$$X\ \ ba_2Yb\ a_3 {=} ba_2bba_2bbb\ a_3 \qquad S'(X) = ba_2b\ S'(Y) = bb$$

# Crossing $a$-blocks?

- Crossing $a$-block: similar to crossing $ab$.

# Crossing $a$-blocks?

- Crossing $a$-block: similar to crossing $ab$.
- pop whole $a$-prefix and $a$-suffix:
  $S(X) = a^{\ell_X} w a^{r_X}$: change it to $S(X) = w$

# Crossing $a$-blocks?

- Crossing $a$-block: similar to crossing $ab$.
- pop whole $a$-prefix and $a$-suffix:
  $S(X) = a^{\ell_X} w a^{r_X}$: change it to $S(X) = w$

1: **for** $X \in \mathcal{X}$ **do**
2: replace each occurrence of $X$ by $a^\ell X a^r$ ▷ $\ell, r \geq 0$
3: ▷ $a^\ell$ and $a^r$ are the $a$-prefix and suffix of $S(X)$
4: **if** $S(X) = \epsilon$ **then**
5: remove $X$ from the equation

# Crossing $a$-blocks?

- Crossing $a$-block: similar to crossing $ab$.
- pop whole $a$-prefix and $a$-suffix:
  $S(X) = a^{\ell_X} w a^{r_X}$: change it to $S(X) = w$

1: **for** $X \in \mathcal{X}$ **do**
2: replace each occurrence of $X$ by $a^\ell X a^r$ ▷ $\ell, r \geq 0$
3: ▷ $a^\ell$ and $a^r$ are the $a$-prefix and suffix of $S(X)$
4: **if** $S(X) = \epsilon$ **then**
5: remove $X$ from the equation

**Lemma**

*After uncrossing $a$ is no longer crossing.*

# Algorithm

**while** $U \notin \Sigma$ and $V \notin \Sigma$ **do**
    $\Gamma \leftarrow$ letters from $U = V$
    choose a pair of letters or a block from $\Gamma$
        ▷ Length minimal solutions use only $\Gamma$
    **if** it is crossing **then**
        Uncross it
    Compress it

# Algorithm

**while** $U \notin \Sigma$ and $V \notin \Sigma$ **do**
    $\Gamma \leftarrow$ letters from $U = V$
    choose a pair of letters or a block from $\Gamma$
                ▷ Length minimal solutions use only $\Gamma$
    **if** it is crossing **then**
        Uncross it
    Compress it

The problem with occurrences over the cut disappeared.
But equation changes, we need to bound its size.

# Equation's size

We show that

- we stay in $\mathcal{O}(n^2)$ space.
- After each operation the length-minimal solution shortens.

# Equation's size

We show that

- we stay in $\mathcal{O}(n^2)$ space.
- After each operation the length-minimal solution shortens.

So we terminate on positive instances.

# Equation's size

We show that

- we stay in $\mathcal{O}(n^2)$ space.
- After each operation the length-minimal solution shortens.

So we terminate on positive instances.

# Strategy

**Lemma**

*Compression of a non-crossing pair/block decreases equation's size.*

**Proof.**

Something is compressed in the equation. □

# Strategy

## Lemma

*Compression of a non-crossing pair/block decreases equation's size.*

## Proof.

Something is compressed in the equation. □

## Strategy

- If there is something non-crossing: compress it.
- If there are only crossing: choose one that minimises the equation.

## Lemma (Fixed solution)

*There are at most $2n$ different crossing pairs and blocks.*

## Lemma (Fixed solution)

*There are at most $2n$ different crossing pairs and blocks.*

Each is associated with a side of an occurrence of a variable.

**Lemma (Fixed solution)**

*There are at most $2n$ different crossing pairs and blocks.*

Each is associated with a side of an occurrence of a variable.

**Lemma (Fixed solution)**

*Uncrossing introduces at most $2n$ letters to the equation.*

**Lemma (Fixed solution)**

*There are at most $2n$ different crossing pairs and blocks.*

Each is associated with a side of an occurrence of a variable.

**Lemma (Fixed solution)**

*Uncrossing introduces at most $2n$ letters to the equation.*

Each variable pops left and right one letter
for $a$-chains: it is compressed immediately afterwards.

**Lemma (Fixed solution)**

*There are at most $2n$ different crossing pairs and blocks.*

Each is associated with a side of an occurrence of a variable.

**Lemma (Fixed solution)**

*Uncrossing introduces at most $2n$ letters to the equation.*

Each variable pops left and right one letter
for $a$-chains: it is compressed immediately afterwards.

**Lemma**

*There is always some choice to be $\leq 8n^2$.*

**Lemma (Fixed solution)**

*There are at most $2n$ different crossing pairs and blocks.*

Each is associated with a side of an occurrence of a variable.

**Lemma (Fixed solution)**

*Uncrossing introduces at most $2n$ letters to the equation.*

Each variable pops left and right one letter
for $a$-chains: it is compressed immediately afterwards.

**Lemma**

*There is always some choice to be $\leq 8n^2$.*

There are $m \leq 8n^2$ letters and $k \leq 2n$ different crossing blocks/pairs.
Some covers $\geq m/k$ letters.
Its compression removes $\geq m/2k$ letters and introduces $2n$ letters.
We are left with at most

$$(1 - 1/2k) \cdot m + 2n \leq (1 - 1/4n) \cdot 8n^2 + 2n = 8n^2 \quad .$$

# So what?

## Valid concern:

- We still have PSPACE, known for 20 years.
- Simplicity lies in the eye of the author
  (earlier version was more complex)

- I think it is simpler.
- No word combinatorics: much more robust.
  - easier to generalize
  - easier to adapt

# Representation of all solutions

## Plandowski

Graph-like representation

- much more involved

# Representation of all solutions

## Plandowski

Graph-like representation

- much more involved

## Recompression

Rather easy, various possible approach:

- Change the notion of minimality (like Plandowski): natural
- tricks with the alphabet (Diekert)

# Representation of all solutions

## Plandowski

Graph-like representation

- much more involved

## Recompression

Rather easy, various possible approach:

- Change the notion of minimality (like Plandowski): natural
- tricks with the alphabet (Diekert)

## EDT0L

The set of all solutions is an EDT0L language.
Requires algebraic reformulation, follows naturally.

# Free groups

Involution + regular constraints $\Longrightarrow$ equations in free group

## Free groups

- PSPACE algorithm (known)
- exponential representation of all solutions (new)
- set of all solutions is an EDT0L language

# Involution, regular constraints

## Involution

$\bar{\cdot} : \Sigma \to \Sigma$, where $\bar{\bar{\cdot}}$ is an identity

# Involution, regular constraints

## Involution

$\bar{\cdot} : \Sigma \to \Sigma$, where $\bar{\bar{\cdot}}$ is an identity

When we replace $ab$ we replace $\overline{ab}$ as well.
Some technicalities with self-involuting pairs.

# Involution, regular constraints

## Involution

$\bar{\cdot} : \Sigma \to \Sigma$, where $\bar{\bar{\cdot}}$ is an identity

When we replace $ab$ we replace $\overline{ab}$ as well.
Some technicalities with self-involuting pairs.

## Regular constraints

Keep transition matrix for each letter.

# Involution, regular constraints

## Involution

$\bar{\cdot} : \Sigma \to \Sigma$, where $\bar{\bar{\cdot}}$ is an identity

When we replace $ab$ we replace $\overline{ab}$ as well.
Some technicalities with self-involuting pairs.

## Regular constraints

Keep transition matrix for each letter.Some problems with the

length-minimal solutions
(we cannot erase letters)

# Required more work

- partial commutation [Diekert, J., Kufleitner]
- twisted word equations (permutation of letters) [Diekert, Elder]
- nondeterministic linear space = context sensitive language [J.]
- context unification (term equations) [J.]
- linear time for one variable [J.]

# Efficient variants

- equality of SLPs
- smallest grammar problem
- smallest tree grammar problem
- compressed membership for NFAs
- compressed pattern matching

# Open questions

## Open questions

- Are word equations in NP?
- are smallest solutions at most doubly exponential?