Word Equations and Straight-Line Programs
Lecture 1: Satisfiability of Word Equations

Artur Jeż

University of Wrocław

PhD Open
Warsaw 25.11.2021

Word Equations

Word equation

$$U = V \quad \text{where } U, V \in (\Sigma \cup \mathcal{X})^*$$

- Σ (finite) set of letters
- $\mathcal{X}$ (finite) set of variables

Word Equations

Word equation

$$U = V \quad \text{where } U, V \in (\Sigma \cup \mathcal{X})^*$$

- Σ (finite) set of letters
- $\mathcal{X}$ (finite) set of variables
- $S : \mathcal{X} \to \Sigma^*$ (substitution)
  extend to $S : (\mathcal{X} \cup \Sigma)^* \to \Sigma^*$
- solution: $S(U) = S(V)$ (solution word)
- length-minimal solution: the one minimizing $|S(U)|$
  (other notions are OK as well)

Word Equations

Word equation

$$U = V \quad \text{where } U, V \in (\Sigma \cup \mathcal{X})^*$$

- Σ (finite) set of letters
- $\mathcal{X}$ (finite) set of variables
- $S : \mathcal{X} \to \Sigma^*$ (substitution)
  extend to $S : (\mathcal{X} \cup \Sigma)^* \to \Sigma^*$
- solution: $S(U) = S(V)$ (solution word)
- length-minimal solution: the one minimizing $|S(U)|$
  (other notions are OK as well)

Example

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

Word Equations

Word equation

$$U = V \quad \text{where } U, V \in (\Sigma \cup \mathcal{X})^*$$

- Σ (finite) set of letters
- $\mathcal{X}$ (finite) set of variables
- $S : \mathcal{X} \to \Sigma^*$ (substitution)
  extend to $S : (\mathcal{X} \cup \Sigma)^* \to \Sigma^*$
- solution: $S(U) = S(V)$ (solution word)
- length-minimal solution: the one minimizing $|S(U)|$
  (other notions are OK as well)

Example

$XbaYb = baaababbbab$ has a solution $S(X) = baaa$, $S(Y) = bba$
$aX = Xa$ has a solution $S(X) = a^n$

Word Equations

Word equation

$$U = V \quad \text{where } U, V \in (\Sigma \cup \mathcal{X})^*$$

- Σ (finite) set of letters
- $\mathcal{X}$ (finite) set of variables
- $S : \mathcal{X} \to \Sigma^*$ (substitution)
  extend to $S : (\mathcal{X} \cup \Sigma)^* \to \Sigma^*$
- solution: $S(U) = S(V)$ (solution word)
- length-minimal solution: the one minimizing $|S(U)|$
  (other notions are OK as well)

Example

$XbaYb = baaababbbab$ has a solution $S(X) = baaa$, $S(Y) = bba$
$aX = Xa$ has a solution $S(X) = a^n$
$aXXX = XaYYY$ has a solution $S(X) = a^{3n}, S(Y) = a^{2n}$

Word Equations

Extensions

- systems of equations
- constraints (on $S(X)$, on $S(U)$)
- ...

Word Equations

Extensions

- systems of equations
- constraints (on $S(X)$, on $S(U)$)
- ...

Interest came from different sources.

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

Word Equations

Extensions

- systems of equations
- constraints (on $S(X)$, on $S(U)$)
- ...

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

Extensions

- systems of equations
- constraints (on $S(X)$, on $S(U)$)
- ...

Interest came from different sources.

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

Natural questions

- satisfiability Is there a solution?
- independence of the system
- nontriviality of solutions
- relations definability

Satisfiability

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

Satisfiability

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

Wrong

Decidable [Makanin '77]. Very complex.

Compression and word equations

Plandowski & Rytter '98

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

Compression and word equations

Plandowski & Rytter '98

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

Straight Line Program (SLP)

A context free grammar generating exactly one word.

Compression and word equations

Plandowski & Rytter '98

A length-minimal solution of size N a word equation of size n has a Straight Line Program 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 '98

A length-minimal solution of size N a word equation of size n has a Straight Line Program 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.

SLP, Composition system

SLP

Context free grammar generating a single word
Usually: several natural restrictions

- rules $A \to BC$ or $A \to a$ (Chomsky normal form)
- no chain rules ($A \to B$)
- no ϵ-rules ($A \to \epsilon$)
- only useful rules (used in the derivation)
- well-defined (no cycle-definition)

SLP, Composition system

SLP

Context free grammar generating a single word
Usually: several natural restrictions

- rules $A \to BC$ or $A \to a$ (Chomsky normal form)
- no chain rules ($A \to B$)
- no ϵ-rules ($A \to \epsilon$)
- only useful rules (used in the derivation)
- well-defined (no cycle-definition)

Generalized SLP: Composition systems

Productions of the form

$$X \to Y[i\,..\,j]Z[i'\,..\,j']$$

A natural meaning: $Y[i\,..\,j]$ is a substring of word generated by Y.
Can be transformed to SLP with a quadratic blow-up.

Plandowski & Rytter: Idea

Cut

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

Plandowski & Rytter: Idea

Cut

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

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

Plandowski & Rytter: Idea

Cut

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

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

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

Plandowski & Rytter: Idea

Cut

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

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

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

Main observation

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

Plandowski & Rytter: Idea

Cut

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

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

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

Main observation

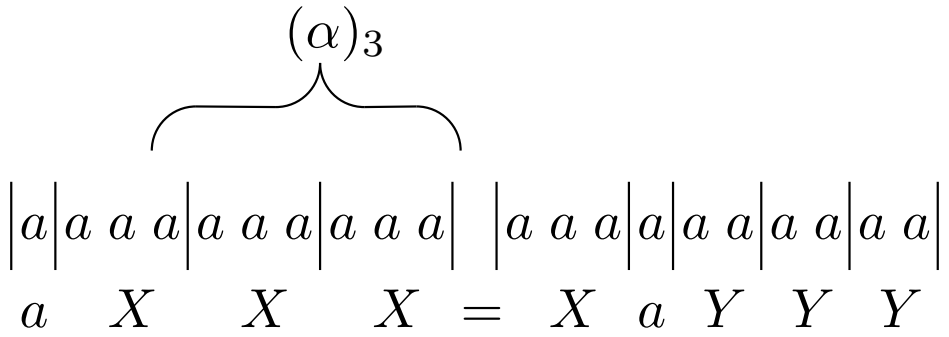
If S is length-minimal and w occurs in $S(u)$, then it "crosses" a cut.
(i.e. the cut is inside or at one of the ends)
Otherwise all occurrences could be removed.

Plandowski & Rytter proof

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

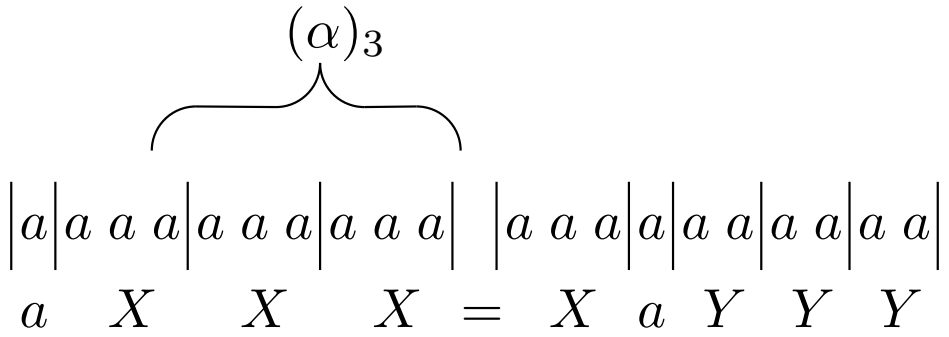
Plandowski & Rytter proof

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



Plandowski & Rytter proof

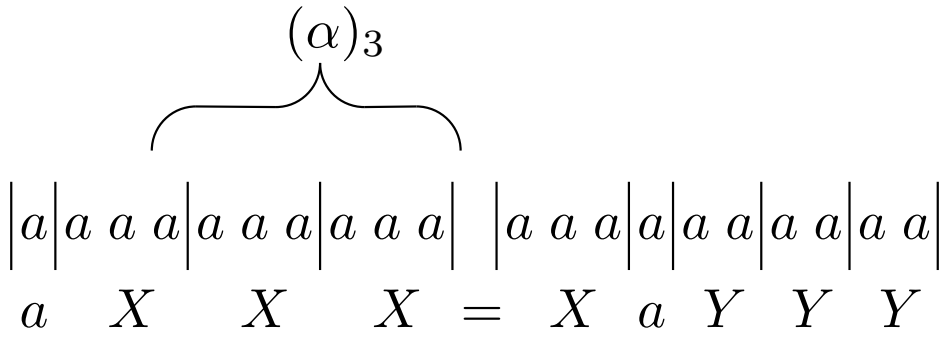
For a cut α let $(\alpha)_k$: the word 2^{k-1} to the left and right of the cut.



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

Plandowski & Rytter proof

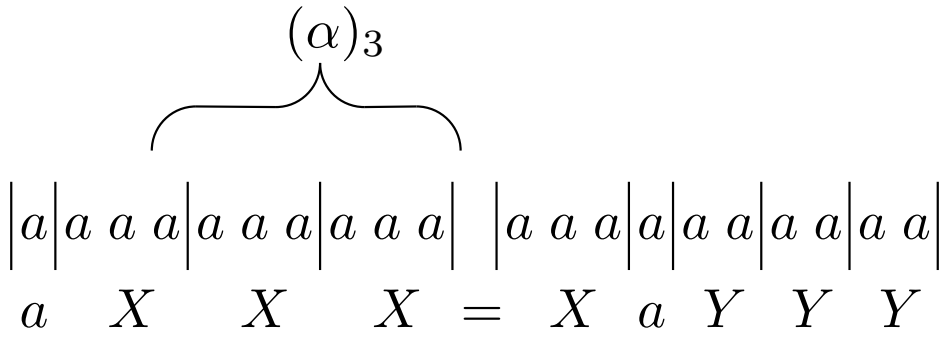
For a cut α let $(\alpha)_k$: the word 2^{k-1} to the left and right of the cut.



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

Plandowski & Rytter proof

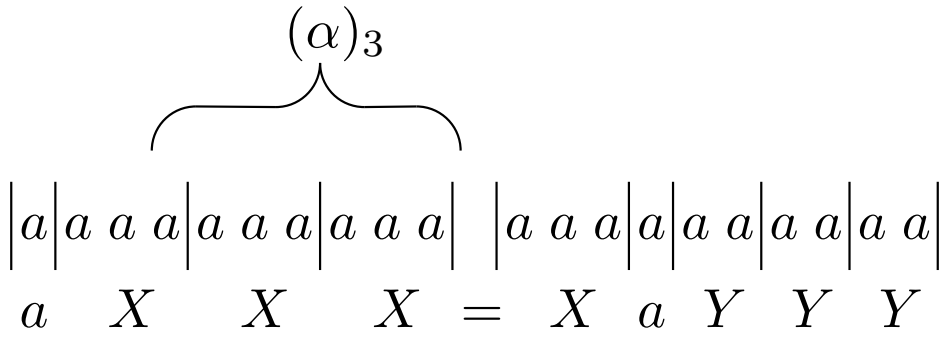
For a cut α let $(\alpha)_k$: the word 2^{k-1} to the left and right of the cut.



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

Plandowski & Rytter proof

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

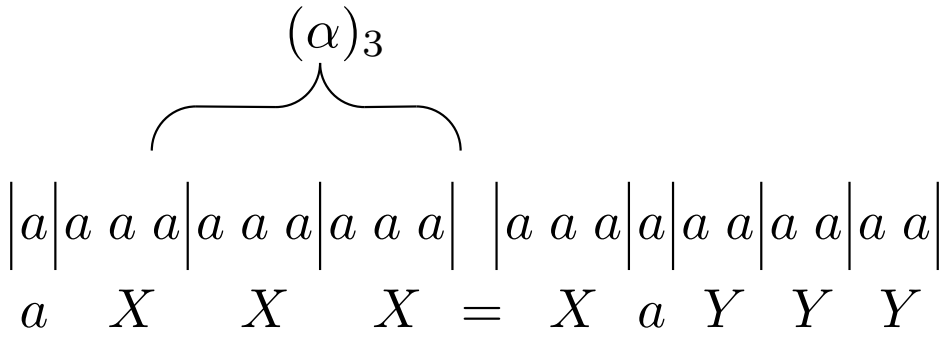


$$
\begin{aligned}
(\alpha)_{k+1} &= w(\alpha_k)w', \quad |w|, |w'| \leq 2^{k-1} \\
w &= (\beta)_k[i\,..\,j] \\
w' &= (\beta')_k[i'\,..\,j'] \\
\Longrightarrow\ (\alpha)_{k+1} &= (\beta)_k[i\,..\,j](\alpha_k)(\beta')_k[i'\,..\,j']
\end{aligned}
$$

Composition system.

Plandowski & Rytter proof

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



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

Composition system.
SLP: size $poly((|U| + |V|)\log N)$.

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.

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.

In the retrospect

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

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.

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.

Equality and Compression of Words

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

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

Equality and Compression of Words

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

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

Equality and Compression of Words

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

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

Equality and Compression of Words

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

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

Equality and Compression of Words

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

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

Equality and Compression of Words

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

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

Equality and Compression of Words

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

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

Intuition: recompression

- New letters: nonterminals of a grammar
- We build a grammar for both words, bottom-up.
- Everything is compressed in the same way!

Equality and Compression of Words

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

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

Intuition: recompression

- New letters: nonterminals of a grammar
- We build a grammar for both words, bottom-up.
- Everything is compressed in the same way!

Comparison with Plandowski's approach

Plandowski & Rytter: Top-down construction; different SLPs for sides
Plandowski: exploits structural properties, many technical problems.

Algorithm: idea

Choose a pair (or letter) from $S(U)$ and compress it.

Algorithm: idea

Choose a pair (or letter) from $S(U)$ and compress it.

while $U \notin \Sigma$ and $V \notin \Sigma$ **do**
    $L \leftarrow$ letters from $S(U) = S(V)$
    for choose $ab \in L^2$ or $a \in L$ **do**
        replace all occurrences of ab in $S(U)$ and $S(V)$
        (or replace all occurrences of blocks of a)

Algorithm: idea

Choose a pair (or letter) from $S(U)$ and compress it.

while $U \notin \Sigma$ and $V \notin \Sigma$ **do**
    $L \leftarrow$ letters from $S(U) = S(V)$
    for choose $ab \in L^2$ or $a \in L$ **do**
        replace all occurrences of ab in $S(U)$ and $S(V)$
        (or replace all occurrences of blocks of a)

How to do this for equations?

Idea at work

Working example

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

Idea at work

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

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

Idea at work

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

$$\begin{aligned} XbaYb &= baaababbbab \quad \text{for } S(X) = baaa\ S(Y) = bba \\ XcYb &= caacbcb \quad \text{for } S'(X) = caa\ S'(Y) = bc \end{aligned}$$

Idea at work

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

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

And what about replacing ab by d?

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

Idea at work

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

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

And what about replacing ab by d?

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

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

Border cases

$$U = V$$

Border cases

$$U = V$$

- there are no letters in U, V: there is a trivial solution: map each letter to a, only length matters
  reduces to linear Diophantine equations

Border cases

$$U = V$$

- there are no letters in U, V: there is a trivial solution:
  map each letter to a, only length matters
  reduces to linear Diophantine equations
- Let L: letters in $U = V$:
  there is a solution $\iff$ there is a solution using only letters in L:
  map each letter $\notin L$ to a fixed letter in L
  If $|L| = 1 \implies$ linear Diophantine equations

Border cases

$$U = V$$

- there are no letters in U, V: there is a trivial solution:
  map each letter to a, only length matters
  reduces to linear Diophantine equations
- Let L: letters in $U = V$:
  there is a solution $\iff$ there is a solution using only letters in L:
  map each letter $\notin L$ to a fixed letter in L
  If $|L| = 1 \implies$ linear Diophantine equations

So we can assume that:

- there are (≥ 2 different) letters in the equation
- the solution is over the letters in the equation

Pair types

Definition (Pair types)

Occurrence of ab in $S(U)$, $S(V)$ (for a fixed substitution S) 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 occurrence, non-crossing otherwise.

Pair types

Definition (Pair types)

Occurrence of ab in $S(U)$, $S(V)$ (for a fixed substitution S) 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 occurrence, non-crossing otherwise.

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

Pair types

Definition (Pair types)

Occurrence of ab in $S(U)$, $S(V)$ (for a fixed substitution S) 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 occurrence, non-crossing otherwise.

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

$baaa\ baa\ bba\ b = baaabaabbbab$ explicit

$baaa\ baa\ bba\ b = baaabaabbbab$ implicit

$baaa\ baa\ bba\ b = baaabaabbbab$ crossing

Compression of non-crossing pairs

PairComp(a, b)

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(a, b)

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

Lemma

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

Correctness of nondeterministic procedures

Definition (Soundness, completeness)

A nondeterministic procedure is
sound: for all nondeterministic choices: if the returned equation is satisfiable $\implies$ the original one is satisfiable
complete: given a satisfiable equation for some nondeterministic choices it returns a satisfiable equation.

Correctness of nondeterministic procedures

Definition (Soundness, completeness)

A nondeterministic procedure is
sound: for all nondeterministic choices: if the returned equation is satisfiable $\implies$ the original one is satisfiable
complete: given a satisfiable equation for some nondeterministic choices it returns a satisfiable equation.

We need a bit more for completeness: solutions "correspond".

Correctness of nondeterministic procedures

Definition (Soundness, completeness)

A nondeterministic procedure is
sound: for all nondeterministic choices: if the returned equation is satisfiable $\implies$ the original one is satisfiable
complete: given a satisfiable equation for some nondeterministic choices it returns a satisfiable equation.

We need a bit more for completeness: solutions "correspond".

In this case

If $U = V$ has a solution S such that ab is non-crossing then the returned equation $U' = V'$ has a solution S' such that $S'(U')$ is $S(U)$ with each ab replaced with c.

Correctness

Complete

$S'(X) \leftarrow S(X)$ with each ab replaced with c
$S'(U')$ 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

$S'(X) \leftarrow S(X)$ with each ab replaced with c
$S'(U')$ 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

$S'(X) \leftarrow S(X)$ with each ab replaced with c
$S'(U')$ 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

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

Correctness

Complete

$S'(X) \leftarrow S(X)$ with each ab replaced with c
$S'(U')$ 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

$S'(X) \leftarrow S(X)$ with each ab replaced with c
$S'(U')$ 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.

$$X\ baa\ Y\ b = baaabaabbbab$$

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

Correctness

Complete

$S'(X) \leftarrow S(X)$ with each ab replaced with c
$S'(U')$ 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.

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

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

Correctness

Complete

$S'(X) \leftarrow S(X)$ with each ab replaced with c
$S'(U')$ 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.

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

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

Uncrossing is sound and complete.
If S is a solution of $U = V$ then for appropriate nondeterministic choices the returned equation $U' = V'$ has a solution S' such that $S(U) = S'(U')$ and ab is non-crossing.

Uncrossing: example

Uncrossing *ab*

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

Uncrossing: example

Uncrossing ab

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

Uncrossing: example

Uncrossing ab

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

$$bX\,a\,baab\,Y\,ab{=}baaabaabbbab \qquad S'(X) = aa\ S'(Y) = b$$

Uncrossing: example

Uncrossing ab

$$X\ baa\ Y\ b{=}baaabaabbbab \quad S(X) = baaa\ S(Y) = bba$$
$$baaabaa\ bba\ b{=}baaabaabbbab$$
$$baaabaab\ ba\ b{=}baaabaabbbab$$
$$bX\,a\,baab\,Y\,ab{=}baaabaabbbab \qquad S'(X) = aa\ S'(Y) = b$$

Proof

Let $a_X \in \{\varepsilon, a\}, b_X \in \{b, \varepsilon\}$, such that we replaced X with $b_X X a_X$ (or $b_X a_X$).
Sound: if S' is a solution, define $S(X) = b_X S'(X) a_X$.
Then $S(U) = S'(U')$.

Proof

Let $a_X \in \{\varepsilon, a\}, b_X \in \{b, \varepsilon\}$, such that we replaced X with $b_X X a_X$ (or $b_X a_X$).

Sound: if S' is a solution, define $S(X) = b_X S'(X) a_X$.

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

Complete: We make the choices according to S:

- if a is last letter of $S(X)$ then replace X with Xa ($a_X = a \iff S(X)$ ends with a)
- if b is first letter of $S(X)$ then replace X with bX ($b_X = b \iff S(X)$ begins with b)
- if $S(X) = \varepsilon$ then remove it

Define $S'(X)$ such that $S(X) = b_X S'(X) a_X$.

Hence $S(U) = S'(U')$.

Proof

Let $a_X \in \{\varepsilon, a\}, b_X \in \{b, \varepsilon\}$, such that we replaced X with $b_X X a_X$ (or $b_X a_X$).

Sound: if S' is a solution, define $S(X) = b_X S'(X) a_X$.

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

Complete: We make the choices according to S:

- if a is last letter of $S(X)$ then replace X with Xa ($a_X = a \iff S(X)$ ends with a)
- if b is first letter of $S(X)$ then replace X with bX ($b_X = b \iff S(X)$ begins with b)
- if $S(X) = \varepsilon$ then remove it

Define $S'(X)$ such that $S(X) = b_X S'(X) a_X$.

Hence $S(U) = S'(U')$.

Can ab be crossing?

If $b_X = \varepsilon$ then first letter of $S'(X)$ is not b: cannot create crossing pair.

If $b_X = b$ then the letter to the left of X is $b \neq a$: cannot create a crossing pair.

Maximal blocks

Definition (maximal block of a)

When a^ℓ occurs in $S(U) = S(V)$ and cannot be extended.

Maximal blocks

Definition (maximal block of a)

When a^ℓ occurs in $S(U) = S(V)$ and cannot be extended.

Equivalents of pairs.

Maximal blocks

Definition (maximal block of a)

When a^ℓ occurs in $S(U) = S(V)$ and cannot be extended.

Equivalents of pairs.

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

Maximal blocks

Definition (maximal block of a)

When a^ℓ occurs in $S(U) = S(V)$ and cannot be extended.

Equivalents of pairs.

- Block occurrence can be explicit, implicit or crossing (wrt. S).
- 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

Definition (maximal block of a)

When a^ℓ occurs in $S(U) = S(V)$ and cannot be extended.

Equivalents of pairs.

- Block occurrence can be explicit, implicit or crossing (wrt. S).
- 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 (Exponent of periodicity: simple case)

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

Blocks compression

When a has no crossing block

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

Lemma

The ChainNCr(a) *is sound and complete.*
If $U = V$ has a solution S such that a is non-crossing then the returned equation $U' = V'$ has a solution S' such that $S'(U')$ is $S(U)$ with each maximal a-block a^ℓ is replaced with a_ℓ.

Blocks compression

When a has no crossing block

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

Lemma

The ChainNCr(a) *is sound and complete.*
If $U = V$ has a solution S such that a is non-crossing then the returned equation $U' = V'$ has a solution S' such that $S'(U')$ is $S(U)$ with each maximal a-block a^ℓ is replaced with a_ℓ.

$X \quad baa\,Y\,baaa{=}baabbaabbbaaa \qquad S(X) = baab\ S(Y) = bb$

Blocks compression

When a has no crossing block

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

Lemma

The ChainNCr(a) *is sound and complete.*
If $U = V$ has a solution S such that a is non-crossing then the returned equation $U' = V'$ has a solution S' such that $S'(U')$ is $S(U)$ with each maximal a-block a^ℓ is replaced with a_ℓ.

X baa Y baaa=baabbaabbbaaa $S(X) =$ baab $S(Y) =$ bb
baabbaabbbaaa=baabbaabbbaaa

Blocks compression

When a has no crossing block

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

Lemma

The ChainNCr(a) *is sound and complete.*
If $U = V$ has a solution S such that a is non-crossing then the returned equation $U' = V'$ has a solution S' such that $S'(U')$ is $S(U)$ with each maximal a-block a^ℓ is replaced with a_ℓ.

$$X \quad baa\,Y\,baaa = baabbaabbbaaa \qquad S(X) = baab\ S(Y) = bb$$
$$baabbaabbbaaa = baabbaabbbaaa$$

$$X \quad ba_2\,Y\,b\,a_3 = ba_2\,bba_2\,bbb\,a_3 \qquad S'(X) = ba_2b\ S'(Y) = bb$$

Blocks compression

When a has no crossing block

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

Lemma

The ChainNCr(a) *is sound and complete.*
If $U = V$ has a solution S such that a is non-crossing then the returned equation $U' = V'$ has a solution S' such that $S'(U')$ is $S(U)$ with each maximal a-block a^ℓ is replaced with a_ℓ.

$$X \quad baa\,Y\,baaa = baabbaabbbaaa \qquad S(X) = baab\ S(Y) = bb$$
$$baabbaabbbaaa = baabbaabbbaaa$$
$$ba_2bba_2bbb\ a_3 = ba_2bba_2bbb\ a_3$$
$$X \quad ba_2\,Y\,b\ a_3 = ba_2bba_2bbb\ a_3 \qquad S'(X) = ba_2b\ S'(Y) = bb$$

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 $a^{\ell} X a^{r}$
3: ▷ a^{ℓ} and a^{r} are the a-prefix and a-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 $a^{\ell} X a^{r}$
3: ▷ a^{ℓ} and a^{r} are the a-prefix and a-suffix of $S(X)$
4: **if** $S(X) = \epsilon$ **then**
5: remove X from the equation

Lemma

a-block uncrossing is sound and complete.
If S is a solution of $U = V$ then for appropriate nondeterministic choices the returned equation $U' = V'$ has a solution S' such that $S(U) = S'(U')$ and a is non-crossing.

Algorithm

while $U \notin \Sigma$ and $V \notin \Sigma$ **do**
    $L \leftarrow$ letters from $U = V$
    choose a pair of letters or a block from L
    if it is crossing **then**
        Uncross it
    Compress it

Algorithm

while $U \notin \Sigma$ and $V \notin \Sigma$ **do**
    $L \leftarrow$ letters from $U = V$
    choose a pair of letters or a block from L
    if it is crossing **then**
        Uncross it
    Compress it

- Composition of sound (complete) procedures are sound (complete) for every choice of letter/pair!

Algorithm

while $U \notin \Sigma$ and $V \notin \Sigma$ **do**
    $L \leftarrow$ letters from $U = V$
    choose a pair of letters or a block from L
    if it is crossing **then**
        Uncross it
    Compress it

- Composition of sound (complete) procedures are sound (complete) for every choice of letter/pair!
- we have a choice of pair/letter

Equation's size

We show that

- we stay in $\mathcal{O}(n^2)$ space for appropriate choice of letter/pair
- After each operation the length-minimal solution shortens.

Equation's size

We show that

- we stay in $\mathcal{O}(n^2)$ space for appropriate choice of letter/pair
- After each operation the length-minimal solution shortens.

So we terminate on positive instances.
(we do not accept negative by soundness)

Strategy

Lemma

Each compression decreases the length of the length-minimal solution word.

Proof.

We perform the compression on the solution word. □

Strategy

Lemma

Each compression decreases the length of the length-minimal solution word.

Proof.

We perform the compression on the solution word. □

Lemma

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

Proof.

Something is compressed in the equation. □

Strategy

Lemma

Each compression decreases the length of the length-minimal solution word.

Proof.

We perform the compression on the solution word. □

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.

Comments

- for a given equation we can “choose” an arbitrary solution and perform compression according to it
- the new equation has a corresponding (compressed) solution; it may have other, shorter and longer

Comments

- for a given equation we can "choose" an arbitrary solution and perform compression according to it
- the new equation has a corresponding (compressed) solution; it may have other, shorter and longer
- if we "choose" a length-minimal one and then we can guarantee some compression rate
- for any choice of solution we guarantee the equation size (for appropriate choice of pair/letter)

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

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

Open questions

Open questions

Are word equations in NP?

Open questions

Open questions

Are word equations in NP?

- Plandowski & Rytter work in nondeterministic $poly(n, \log N)$.

Open questions

Open questions

Are word equations in NP?

- Plandowski & Rytter work in nondeterministic $poly(n, \log N)$.
- one can deduce doubly-exponential upper bound on solution size

Open questions

Open questions

Are word equations in NP?

- Plandowski & Rytter work in nondeterministic $poly(n, \log N)$.
- one can deduce doubly-exponential upper bound on solution size
- only exponential lower bound

Open questions

Open questions

Are word equations in NP?

- Plandowski & Rytter work in nondeterministic $poly(n, \log N)$.
- one can deduce doubly-exponential upper bound on solution size
- only exponential lower bound
- exponential upper bound $\Longrightarrow$ NP-algorithm

Open questions

Open questions

Are word equations in NP?

- Plandowski & Rytter work in nondeterministic $poly(n, \log N)$.
- one can deduce doubly-exponential upper bound on solution size
- only exponential lower bound
- exponential upper bound $\implies$ NP-algorithm
- many believe in exponential upper bound

Extensions

Main advantage: robust.

- many things can be changed/altered
- many things can be added

Extensions

Main advantage: robust.

- many things can be changed/altered
- many things can be added

Extensions

- regular constraints
- involution
- (free) groups
- representation of all solutions
- partial commutation (not so easy)
- term equations (not so easy)

Regular constraints

Regular constraints

We may add regular constraints (for variables and whole equations).

Regular constraints

Regular constraints

We may add regular constraints (for variables and whole equations).

- for a regular langauge we define the (semigroup of) transition matrices (Boolean operations) of the NFA
- for many NFAs we take vector of such transition functions
- constraint: we specify for a variable X the transition ρ_X and require that $\rho(S(X)) = \rho_X$

Regular constraints

Regular constraints

We may add regular constraints (for variables and whole equations).

- for a regular langauge we define the (semigroup of) transition matrices (Boolean operations) of the NFA
- for many NFAs we take vector of such transition functions
- constraint: we specify for a variable X the transition ρ_X and require that $\rho(S(X)) = \rho_X$

Changes in the algorithm

- when popping, we need to guess new transition functions, i.e. when we pop a we need ρ'_X such that

$$\rho_X = \rho'_X \rho(a)$$

- everything more or less generalizes (bound on exponent of periodicity, correctness, etc.)

But not everything

- bound on the alphabet (removal of letters)
- if there a non-crossing pair then it occurs in the equation

But not everything

- bound on the alphabet (removal of letters)
- if there a non-crossing pair then it occurs in the equation

No easy way out.

But not everything

- bound on the alphabet (removal of letters)
- if there a non-crossing pair then it occurs in the equation

No easy way out.

Almost easy way out

Extend the alphabet: add a letter for each possible transition (oracle access!).
This does not change satisfiability.

But not everything

- bound on the alphabet (removal of letters)
- if there a non-crossing pair then it occurs in the equation

No easy way out.

Almost easy way out

Extend the alphabet: add a letter for each possible transition (oracle access!).
This does not change satisfiability.

- The bound on the alphabet holds
- if there a non-crossing pair then it occurs in the equation

A difficult solution

- Every letter corresponds to some string over the input alphabet.

A difficult solution

- Every letter corresponds to some string over the input alphabet.
- If we want to remove the letter: replace it back
  Intuition: its compression was a mistake, we decompress it.

A difficult solution

- Every letter corresponds to some string over the input alphabet.
- If we want to remove the letter: replace it back
  Intuition: its compression was a mistake, we decompress it.
- We keep (as a tool of a proof) a mapping from current alphabet to strings over original alphabet.

A difficult solution

- Every letter corresponds to some string over the input alphabet.
- If we want to remove the letter: replace it back
  Intuition: its compression was a mistake, we decompress it.
- We keep (as a tool of a proof) a mapping from current alphabet to strings over original alphabet.
- so the solution is over the current alphabet $\cup$ original alphabet.

Involution

Definition (Involution)

Think of it as the inverse in a free group.

Bijection $\bar{\cdot}$ on Σ^* and on $\mathcal{X}$ such that

- $\bar{\bar{a}} = a$
- $\overline{a_1 a_2 \cdots a_k} = \overline{a_k} \cdots \overline{a_2}\,\overline{a_1}$

Involution

Definition (Involution)

Think of it as the inverse in a free group.
Bijection $\bar{\cdot}$ on Σ^* and on $\mathcal{X}$ such that

- $\overline{\overline{a}} = a$
- $\overline{a_1 a_2 \cdots a_k} = \overline{a_k} \cdots \overline{a_2}\,\overline{a_1}$

Used for

- groups (formal inverse)
- biology computation
- string operation (reversal)
- ...

How to deal with involution

Remark

When we compress w to c we compress $\overline{w}$ to $\overline{c}$.

How to deal with involution

Remark

When we compress w to c we compress $\overline{w}$ to $\overline{c}$.
What if they overlap?

How to deal with involution

Remark

When we compress w to c we compress $\overline{w}$ to $\overline{c}$.
What if they overlap?

- some simple special cases ($ab\overline{a}$, etc.)
- $a = \overline{a}$, $b = \overline{b}$: we treat alternating a, b-strings as blocks

From groups to semigroups

Diekert, Gutiérrez & Hagenah 2001

Equations in free groups with regular constraints

Reduce to

Equations in free semigroups with regular constraints and involution

From groups to semigroups

Diekert, Gutiérrez & Hagenah 2001

Equations in free groups with regular constraints

Reduce to

Equations in free semigroups with regular constraints and involution

Properties

- purely syntactical
- bijection on the solutions (in case of free groups: reduced)

From groups to semigroups

Diekert, Gutiérrez & Hagenah 2001

Equations in free groups with regular constraints

Reduce to

Equations in free semigroups with regular constraints and involution

Properties

- purely syntactical
- bijection on the solutions (in case of free groups: reduced)

The same holds for other groups, but the obtained semigroups differ.

Reduction

- Triangulate the equation (equations $XY = Z$ and $X = a$)
- Remove cancellation: $XY = Z \quad \rightarrow$
  $X = X'R,\ Y = R^{-1}Y',\ X'Y' = Z$
- Look only at reduced solutions

Reduction

- Triangulate the equation (equations $XY = Z$ and $X = a$)
- Remove cancellation: $XY = Z \quad \rightarrow$
  $X = X'R,\ Y = R^{-1}Y',\ X'Y' = Z$
- Look only at reduced solutions

Reduction

- Triangulate the equation (equations $XY = Z$ and $X = a$)
- Remove cancellation: $XY = Z \quad \rightarrow$
  $X = X'R,\ Y = \overline{R}Y',\ X'Y' = Z$
- Look only at reduced solutions
- Bijection on sets of solutions.

Reduction

- Triangulate the equation (equations $XY = Z$ and $X = a$)
- Remove cancellation: $XY = Z \quad \rightarrow$
  $X = X'R,\ Y = \overline{R}Y',\ X'Y' = Z$
- Look only at reduced solutions
- Bijection on sets of solutions.
- We need involution and regular constraints

Representation of all solutions

Transforming the solutions

The algorithm transforms the solutions:

Representation of all solutions

Transforming the solutions

The algorithm transforms the solutions:

- given an equation $U = V$ and a solution S, for some nondeterministic choices we obtain an equation $U' = V'$ with solution S' that "corresponds" to $S(U)$
  (it is the same or some compression applied)

Representation of all solutions

Transforming the solutions

The algorithm transforms the solutions:

- given an equation $U = V$ and a solution S, for some nondeterministic choices we obtain an equation $U' = V'$ with solution S' that "corresponds" to $S(U)$
  (it is the same or some compression applied)
- if an equation $U = V$ is transformed to $U' = V$ which has a solution S' then we can construct a corresponding solution S of $U = V$.
  (appending/prepending letters, $c \to ab$)

Representation of all solutions

Transforming the solutions

The algorithm transforms the solutions:

- given an equation $U = V$ and a solution S, for some nondeterministic choices we obtain an equation $U' = V'$ with solution S' that "corresponds" to $S(U)$
  (it is the same or some compression applied)
- if an equation $U = V$ is transformed to $U' = V$ which has a solution S' then we can construct a corresponding solution S of $U = V$.
  (appending/prepending letters, $c \to ab$)

Problem

No bound on the exponent in block compression.

Representation of all solutions

Transforming the solutions

The algorithm transforms the solutions:

- given an equation $U = V$ and a solution S, for some nondeterministic choices we obtain an equation $U' = V'$ with solution S' that "corresponds" to $S(U)$
  (it is the same or some compression applied)
- if an equation $U = V$ is transformed to $U' = V$ which has a solution S' then we can construct a corresponding solution S of $U = V$.
  (appending/prepending letters, $c \to ab$)

Problem

No bound on the exponent in block compression.

$$aXXXX = XaYY \implies S(X) = a^{2m}, S(Y) = a^{3m}$$

Representation of all solutions

Transforming the solutions

The algorithm transforms the solutions:

- given an equation $U = V$ and a solution S, for some nondeterministic choices we obtain an equation $U' = V'$ with solution S' that "corresponds" to $S(U)$
  (it is the same or some compression applied)
- if an equation $U = V$ is transformed to $U' = V$ which has a solution S' then we can construct a corresponding solution S of $U = V$.
  (appending/prepending letters, $c \to ab$)

Problem

No bound on the exponent in block compression.

$$aXXXX = XaYY \implies S(X) = a^{2m}, S(Y) = a^{3m}$$

This boils down to solutions of systems of integer equations.
More about it later on.

Graph representation of all solutions

Representation of all solutions

- Graph (exponential size)
- Nodes: equations of size $\mathcal{O}(n^2)$, at most n occurrences of variables
- Edges between nodes, labelled with (families of) morphisms

Graph representation of all solutions

Representation of all solutions

- Graph (exponential size)
- Nodes: equations of size $\mathcal{O}(n^2)$, at most n occurrences of variables
- Edges between nodes, labelled with (families of) morphisms

Preserving solutions

If S is a solution of $U = V$ then

- the equtaion is trivial or
- there is ϕ-labelled edge to an equation $U' = V'$ with a solution S' such that $S = \phi(S')$

Graph representation of all solutions

Representation of all solutions

- Graph (exponential size)
- Nodes: equations of size $\mathcal{O}(n^2)$, at most n occurrences of variables
- Edges between nodes, labelled with (families of) morphisms

Preserving solutions

If S is a solution of $U = V$ then

- the equtaion is trivial or
- there is ϕ-labelled edge to an equation $U' = V'$ with a solution S' such that $S = \phi(S')$

If $U' = V'$ has a solution S' and there is ϕ-labelled edge from $U = V$ then $\phi(S')$ is a solution of $U = V$.

Graph representation of all solutions

Representation of all solutions

- Graph (exponential size)
- Nodes: equations of size $\mathcal{O}(n^2)$, at most n occurrences of variables
- Edges between nodes, labelled with (families of) morphisms

Preserving solutions

If S is a solution of $U = V$ then

- the equtaion is trivial or
- there is ϕ-labelled edge to an equation $U' = V'$ with a solution S' such that $S = \phi(S')$

If $U' = V'$ has a solution S' and there is ϕ-labelled edge from $U = V$ then $\phi(S')$ is a solution of $U = V$.

Each solution is obtained on a path from original equation to a trivial one.