Compressed Membership for NFA (DFA) with Compressed Labels is in NP (P)

Artur Jeż
University of Wrocław

What this talk is about

- Fully compressed membership problem for automata

What this talk is about

- Fully compressed membership problem for automata
- no automata in this talk

What this talk is about

- Fully compressed membership problem for automata
- no automata in this talk
- SPLs and a technique for them

What this talk is about

- Fully compressed membership problem for automata
- no automata in this talk
- SPLs and a technique for them
- more general (word equations)

What this talk is about

- Fully compressed membership problem for automata
- no automata in this talk
- SPLs and a technique for them
- more general (word equations)

Results

What this talk is about

- Fully compressed membership problem for automata
- no automata in this talk
- SPLs and a technique for them
- more general (word equations)

Results

- Fully compressed membership problem for NFA (in NP)

What this talk is about

- Fully compressed membership problem for automata
- no automata in this talk
- SPLs and a technique for them
- more general (word equations)

Results

- Fully compressed membership problem for NFA (in NP)
- Fully compressed membership problem for DFA (in P)

What this talk is about

- Fully compressed membership problem for automata
- no automata in this talk
- SPLs and a technique for them
- more general (word equations)

Results

- Fully compressed membership problem for NFA (in NP)
- Fully compressed membership problem for DFA (in P)
- (SLP) fully compressed pattern matching (in $\mathcal{O}(n^2)$)

What this talk is about

- Fully compressed membership problem for automata
- no automata in this talk
- SPLs and a technique for them
- more general (word equations)

Results

- Fully compressed membership problem for NFA (in NP)
- Fully compressed membership problem for DFA (in P)
- (SLP) fully compressed pattern matching (in $\mathcal{O}(n^2)$)
- word equations: simple, unified proof for everything that is known

Straight Line Programms SLPs

Definition (Straight Line Programms (SLP))

Context free grammar defining a single word. (Chomsky normal form).

Straight Line Programms SLPs

Definition (Straight Line Programms (SLP))

Context free grammar defining a single word. (Chomsky normal form).

Up to exponential compression.

Straight Line Programms SLPs

Definition (Straight Line Programms (SLP))

Context free grammar defining a single word. (Chomsky normal form).

Up to exponential compression.

SLPs as a compression model

- application (LZ, logarithmic transformation)
- theory (formal languages)
- preserves/captures word properties

Straight Line Programms SLPs

Definition (Straight Line Programms (SLP))

Context free grammar defining a single word. (Chomsky normal form).

Up to exponential compression.

SLPs as a compression model

- application (LZ, logarithmic transformation)
- theory (formal languages)
- preserves/captures word properties

Applied in many proofs and constructions.

Usage and work on SLP

Theory

- word equations (Plandowski: satisfiability in PSPACE)

Usage and work on SLP

Theory

- word equations (Plandowski: satisfiability in PSPACE)

LZW/LZ dealing algorithms

- $\mathcal{O}(n \log(N/n))$ pattern matching for LZ compressed text
- $\mathcal{O}(n)$ pattern matching for fully LZW compressed text

Usage and work on SLP

Theory

- word equations (Plandowski: satisfiability in PSPACE)

LZW/LZ dealing algorithms

- $\mathcal{O}(n \log(N/n))$ pattern matching for LZ compressed text
- $\mathcal{O}(n)$ pattern matching for fully LZW compressed text

String algorithms

- equality
- pattern matching

Usage and work on SLP

Theory

- word equations (Plandowski: satisfiability in PSPACE)

LZW/LZ dealing algorithms

- $\mathcal{O}(n \log(N/n))$ pattern matching for LZ compressed text
- $\mathcal{O}(n)$ pattern matching for fully LZW compressed text

String algorithms

- equality
- pattern matching

Independent interest

- indexing structure for SLP

Compressed membership

- SLPs are used
- develop tools/gain understanding
- membership problem

Compressed membership

- SLPs are used
- develop tools/gain understanding
- membership problem

Compressed membership [Plandowski & Rytter 1999]

In membership problems, words are given as SLPs.

Compressed membership

- SLPs are used
- develop tools/gain understanding
- membership problem

Compressed membership [Plandowski & Rytter 1999]

In membership problems, words are given as SLPs.

Known results

RE, CFG, Conjunctive grammars. . .

Compressed membership

- SLPs are used
- develop tools/gain understanding
- membership problem

Compressed membership [Plandowski & Rytter 1999]

In membership problems, words are given as SLPs.

Known results

RE, CFG, Conjunctive grammars. . .

Open questions

- Compressed membership for NFA

Compressed membership for NFA

Input: SLP, NFA N
Output: Yes/No

Compressed membership for NFA

Input: SLP, NFA N
Output: Yes/No

Simple dynamic algorithm: for X_i calculate $\{(p, q) \mid \delta(p, \text{val}(X_i), q)\}$

Compressed membership for NFA

Input: SLP, NFA N
Output: Yes/No

Simple dynamic algorithm: for X_i calculate $\{(p, q) \mid \delta(p, \text{val}(X_i), q)\}$
Where is the hardness?

Compressed membership for NFA

Input: SLP, NFA N
Output: Yes/No

Simple dynamic algorithm: for X_i calculate $\{(p, q) \mid \delta(p, \text{val}(X_i), q)\}$
Where is the hardness?

Compress N as well: allow transition by words.

Compressed membership for NFA

Input: SLP, NFA N
Output: Yes/No

Simple dynamic algorithm: for X_i calculate $\{(p, q) \mid \delta(p, \text{val}(X_i), q)\}$
Where is the hardness?

Compress N as well: allow transition by words.

Fully compressed NFA membership

- SLP for w
- NFA N, compressed transitions

Compressed membership for NFA

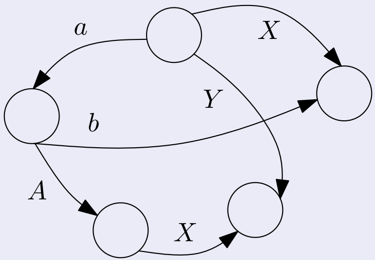
Input: SLP, NFA N
Output: Yes/No

Simple dynamic algorithm: for X_i calculate $\{(p,q) \mid \delta(p, \text{val}(X_i), q)\}$
Where is the hardness?

Compress N as well: allow transition by words.

Fully compressed NFA membership

- SLP for w
- NFA N, compressed transitions



Compressed membership for NFA: complexity

Complexity

- NP-hardness (subsum), already for
  - acyclic NFA
  - unary alphabet
- in PSPACE: enough to store positions inside decompressed words

Compressed membership for NFA: complexity

Complexity

- NP-hardness (subsum), already for
  - acyclic NFA
  - unary alphabet
- in PSPACE: enough to store positions inside decompressed words

Conjecture

In NP.

Partial results

- Plandowski & Rytter (unary in NP)
- Lohrey & Mathissen (highly periodic in NP, highly aperiodic in P)

New results

Theorem

Fully compressed membership for NFA is in NP.

Theorem

Fully compressed membership for DFA is in P.

Idea: Recompression

Difficulty: the words are long. Shorten them.

Idea: Recompression

Difficulty: the words are long. Shorten them.

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

Idea: Recompression

Difficulty: the words are long. Shorten them.

$d \quad c \quad a \quad d$

Idea: Recompression

Difficulty: the words are long. Shorten them.

$d \quad c \quad a \quad d$

Deeper understanding

New production: $d \to ab$. Building new SLP (recompression).
SLP problems: hard, as SLP are different.
Building canonical SLP for the instance.

Idea: Recompression

Difficulty: the words are long. Shorten them.

$d \quad c \quad a \quad d$

Deeper understanding

New production: $d \to ab$. Building new SLP (recompression).
SLP problems: hard, as SLP are different.
Building canonical SLP for the instance.

What to do with a^n?

$a \quad a \quad c \quad a \quad a \quad a$

Idea: Recompression

Difficulty: the words are long. Shorten them.

$d \quad c \quad a \quad d$

Deeper understanding

New production: $d \to ab$. Building new SLP (recompression).
SLP problems: hard, as SLP are different.
Building canonical SLP for the instance.

What to do with a^n? Replace each maximal a^n by a single symbol.

$a_2 \quad c \quad a_3$

Idea: Recompression

Difficulty: the words are long. Shorten them.

$d \quad c \quad a \quad d$

Deeper understanding

New production: $d \to ab$. Building new SLP (recompression).
SLP problems: hard, as SLP are different.
Building canonical SLP for the instance.

What to do with a^n? Replace each maximal a^n by a single symbol.

$a_2 \quad c \quad a_3$

Problems

Easy for text, what about grammar?

Local recompression

Re-compression

- decompressed text: easy; size: large,
- compressed text: hard; size: small.

Local recompression

Re-compression

- decompressed text: easy; size: large,
- compressed text: hard; size: small.

Local decompression

Decompress locally the SLP:

$$X \to uYvZ$$

- u, v: blocks of letters, linear size
- Y, Z: nonterminals
- recompression inside u, v

Outline

Outline of the algorithm

while $|\text{val}(X_n) > n|$ **do**

$L_\Sigma \leftarrow$ list of letters, $L_P \leftarrow$ list of pairs

for $ab \in L_P$ **do**

compress pair ab

for $a \in L_\Sigma$ **do**

compress a maximal blocks

Decompress the word and solve the problem naively.

Outline

Outline of the algorithm

while $|\text{val}(X_n) > n|$ **do**
  $L_\Sigma \leftarrow$ list of letters, $L_P \leftarrow$ list of pairs
  for $ab \in L_P$ **do**
    compress pair ab
  for $a \in L_\Sigma$ **do**
    compress a maximal blocks
Decompress the word and solve the problem naively.

Theorem

There are $\mathcal{O}(\log|\text{val}(X_n)|)$ *iterations.*

Proof.

Consider two consecutive letters ab. One of them is compressed. So word shortens by a constant factor. □

What is hard, what is easy

What is hard to compress, what easy?

What is hard, what is easy

What is hard to compress, what easy?

Hard

- a pair ab is crossing if $X_i \to uaX_jvX_k$, where $\text{val}(X_j) = b \dots$
- a letter a has crossing appearances if aa is a crossing pair

What is hard, what is easy

What is hard to compress, what easy?

Hard

- a pair ab is crossing if $X_i \to uaX_jvX_k$, where $\text{val}(X_j) = b \dots$
- a letter a has crossing appearances if aa is a crossing pair

Easy

- a pair ab is non-crossing otherwise
- a letter a has no crossing appearances otherwise

A little detailed outline

Detailed outline

while $|\text{val}(X_n) > n|$ **do**
  while possible **do**
    for non-crossing pair ab in $\text{val}(X_n)$ **do**
      compress ab
    for a: without crossing blocks **do**
      compress appearances of a

A little detailed outline

Detailed outline

while $|\text{val}(X_n) > n|$ **do**
  while possible **do**
    for non-crossing pair ab in $\text{val}(X_n)$ **do**
      compress ab
    for a: without crossing blocks **do**
      compress appearances of a
  $L \leftarrow$ list of letters with crossing blocks
  $P \leftarrow$ list of crossing pairs
  for each ab in P **do**
    compress ab
  for $a \in L$ **do**
    compress appearances of a
Decompress X_n and solve the problem naively.

Non-crossing pair compression

Non-crossing pair compression

for each production $X_i \to uX_jvX_k$ **do**
    replace each ab in u, v by c

Non-crossing pair compression

Non-crossing pair compression

for each production $X_i \to uX_jvX_k$ **do**
  replace each ab in u, v by c

Appearance compression for a without crossing blocks

compute the lengths $\ell_1, \ldots, \ell_k$ of a's maximal blocks
for each a^{ℓ_m} **do**
  for each production $X_i \to uX_jvX_k$ **do**
    replace maximal a^{ℓ_m} in in u, v by a_{ℓ_m}

Non-crossing pair compression

Non-crossing pair compression

for each production $X_i \to uX_jvX_k$ **do**
  replace each ab in u, v by c

Appearance compression for a without crossing blocks

compute the lengths $\ell_1, \ldots, \ell_k$ of a's maximal blocks
for each a^{ℓ_m} **do**
  for each production $X_i \to uX_jvX_k$ **do**
    replace maximal a^{ℓ_m} in in u, v by a_{ℓ_m}

Lemma

It works.

Proof.

The pair is non-crossing: it always appears inside production. □

Convert hard to easy

Convert crossing pairs to noncrossing and letters with crossing blocks to letters without crossing blocks (Sequentially).

Convert hard to easy

Convert crossing pairs to noncrossing and letters with crossing blocks to letters without crossing blocks (Sequentially).

- aX_i and X_i begins with b
- $X_j b$ and X_j ends with a

Convert hard to easy

Convert crossing pairs to noncrossing and letters with crossing blocks to letters without crossing blocks (Sequentially).

- aX_i and X_i begins with b
- $X_j b$ and X_j ends with a
- 'pop' first letter of X_i
- replace: $\text{val}(X_i) = bu \mapsto \text{val}(X_i) = u$
  grammar: remove leading b from rule for X_i, replace X_i by bX_i

Convert hard to easy

Convert crossing pairs to noncrossing and letters with crossing blocks to letters without crossing blocks (Sequentially).

- aX_i and X_i begins with b
- $X_j b$ and X_j ends with a
- 'pop' first letter of X_i
- replace: $\text{val}(X_i) = bu \mapsto \text{val}(X_i) = u$
  grammar: remove leading b from rule for X_i, replace X_i by bX_i
- 'pop' last letter of X_i

Convert hard to easy

Convert crossing pairs to noncrossing and letters with crossing blocks to letters without crossing blocks (Sequentially).

- aX_i and X_i begins with b
- $X_j b$ and X_j ends with a
- 'pop' first letter of X_i
- replace: $\text{val}(X_i) = bu \mapsto \text{val}(X_i) = u$
  grammar: remove leading b from rule for X_i, replace X_i by bX_i
- 'pop' last letter of X_i

Lemma

After popping letters, ab is noncrossing.

Proof.

Easy, some simple cases. □

Removing crossing blocks of a

- aa is a crossing pair: pop a
- can be insufficient
- cut a-prefix or a-suffix
- Represent $\text{val}(X_i)$ as $a^{\ell_i} w a^{r_i}$, turn it into w.

Removing crossing blocks of a

- aa is a crossing pair: pop a
- can be insufficient
- cut a-prefix or a-suffix
- Represent $\text{val}(X_i)$ as $a^{\ell_i} w a^{r_i}$, turn it into w.

Changing a letter a with crossing blocks to one without

for $i = 1 \,..\, n$ **do**
  let $X_i \to uX_jvX_k$
  calculate the a-prefix a^{ℓ_i} and a-suffix a^{r_i}, remove them
  replace X_i in rules bodies by $a^{\ell_i} X_i a^{r_i}$

Removing crossing blocks of a

- aa is a crossing pair: pop a
- can be insufficient
- cut a-prefix or a-suffix
- Represent $\text{val}(X_i)$ as $a^{\ell_i} w a^{r_i}$, turn it into w.

Changing a letter a with crossing blocks to one without

for $i = 1 \,..\, n$ **do**
  let $X_i \to u X_j v X_k$
  calculate the a-prefix a^{ℓ_i} and a-suffix a^{r_i}, remove them
  replace X_i in rules bodies by $a^{\ell_i} X_i a^{r_i}$

Lemma

After the algorithm a has no crossing block.

Removing crossing blocks of a

- aa is a crossing pair: pop a
- can be insufficient
- cut a-prefix or a-suffix
- Represent $\text{val}(X_i)$ as $a^{\ell_i} w a^{r_i}$, turn it into w.

Changing a letter a with crossing blocks to one without

for $i = 1 \mathinner{\ldotp\ldotp} n$ **do**
  let $X_i \to uX_j vX_k$
  calculate the a-prefix a^{ℓ_i} and a-suffix a^{r_i}, remove them
  replace X_i in rules bodies by $a^{\ell_i} X_i a^{r_i}$

Lemma

After the algorithm a has no crossing block.

Represent a^{ℓ} succinctly, using $\mathcal{O}(\log \ell)$ bits.

Sizes and running time

Running time

All algorithms run in time $poly(n, |G|, |\Sigma|)$.

Sizes and running time

Running time

All algorithms run in time $poly(n, |G|, |\Sigma|)$.

Size of G

$$abbbcceaX_j addfeaaf X_k$$

In each iteration

Sizes and running time

Running time

All algorithms run in time $poly(n, |G|, |\Sigma|)$.

Size of G

$$abbbccea\,bha\,X_j\,ab\,addfeaaf\,cda\,X_k$$

In each iteration

- $\mathcal{O}(n)$ new letters

Sizes and running time

Running time

All algorithms run in time $poly(n, |G|, |\Sigma|)$.

Size of G

$$abbbccea\,bhaX_j\,ab\,addfeaaf\,cdaX_k$$

In each iteration

- $\mathcal{O}(n)$ new letters
- shrinking by a constant factor

Sizes and running time

Running time

All algorithms run in time $poly(n, |G|, |\Sigma|)$.

Size of G

$$uv\,bhaX_j\,ab\,xyz\,cdaX_k$$

In each iteration

- $\mathcal{O}(n)$ new letters
- shrinking by a constant factor

Sizes and running time

Running time

All algorithms run in time $poly(n, |G|, |\Sigma|)$.

Size of G

$$uvbhaX_jabxyzcdaX_k$$

In each iteration

- $\mathcal{O}(n)$ new letters
- shrinking by a constant factor

New letters ($|\Sigma|$)

- noncrossing pairs, noncrossing blocks compression (shrinks $|G|$)
- letters with crossing blocks and crossing pairs:
  there are $\mathcal{O}(n)$ such letters and $\mathcal{O}(n^2)$ pairs in $\mathrm{val}(X_n)$

Modifications

- compressed membership: how to modify automaton?
  - for NFA: nondeterminism
  - for DFA: deterministically

Modifications

- compressed membership: how to modify automaton?
  - for NFA: nondeterminism
  - for DFA: deterministically
- compressed pattern matching
  - better analysis
  - careful implementation
  - details (ends of the pattern)

Modifications

- compressed membership: how to modify automaton?
  - for NFA: nondeterminism
  - for DFA: deterministically
- compressed pattern matching
  - better analysis
  - careful implementation
  - details (ends of the pattern)
- word equations: some further understanding

Modifications

- compressed membership: how to modify automaton?
  - for NFA: nondeterminism
  - for DFA: deterministically
- compressed pattern matching
  - better analysis
  - careful implementation
  - details (ends of the pattern)
- word equations: some further understanding

Questions

- Any further results?
- How efficient for DFA?
- Are word equations in NP?