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

Artur Jeż

Wrocław, Poland

7 September 2011

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).

Much smaller, than the word.
Word a^n: grammar size $\mathcal{O}(\log n)$

$$A_1 \to a \quad A_2 \to A_1 A_1 \; \dots \; A_{\ell+1} \to A_\ell A_\ell \qquad A \to A_{\ell_0} A_{\ell_1} \dots$$

Straight Line Programms SLPs

Definition (Straight Line Programms (SLP))

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

Much smaller, than the word.
Word a^n: grammar size $\mathcal{O}(\log n)$

$$A_1 \to a \quad A_2 \to A_1 A_1 \ \dots \ A_{\ell+1} \to A_\ell A_\ell \qquad A \to A_{\ell_0} A_{\ell_1} \dots$$

SLPs as a compression model

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

Usage and work on SLP

Theory

- word equations (Plandowski: satisfiability in PSPACE)

Usage and work on SLP

Theory

- word equations (Plandowski: satisfiability in PSPACE)

String algorithms

- equality
- pattern matching

Usage and work on SLP

Theory

- word equations (Plandowski: satisfiability in PSPACE)

String algorithms

- equality
- pattern matching

LZW/LZ dealing algorithms

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

String algorithms

- equality
- pattern matching

LZW/LZ dealing algorithms

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

Independent interest

- indexing structure for SLP

Compressed membership

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

Compressed membership

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

Compressed membership [Plandowski & Rytter; *Jewels are forever* 1999]

In membership problems, words are given as SLPs.

Compressed membership

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

Compressed membership [Plandowski & Rytter; *Jewels are forever* 1999]

In membership problems, words are given as SLPs.

Known results

RE, CFG, Conjunctive grammars . . .

Compressed membership

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

Compressed membership [Plandowski & Rytter; *Jewels are forever* 1999]

In membership problems, words are given as SLPs.

Known results

RE, CFG, Conjunctive grammars . . .

Open questions

- Posted in *Jewels are forever*
- some solved
- 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 $\{(p, q) \mid \delta(p, \text{word}(X_i), q)\}$

Compressed membership for NFA

Input: SLP, NFA N
Output: Yes/No

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

Compressed membership for NFA

Input: SLP, NFA N
Output: Yes/No

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

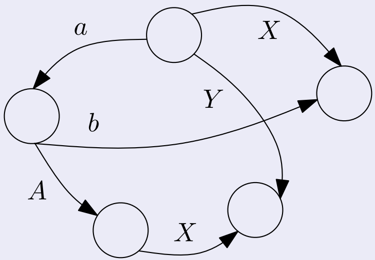
Input: SLP, NFA N
Output: Yes/No

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

New results

Theorem

Fully compressed membership for NFA is in NP.

Theorem

Fully compressed membership for DFA is in P.

New technique

New, interesting technique.

Convention

Convention

- SLPs given as a single grammar
- $X_i \to X_j X_k$, implies $i > j$, $i > k$
- input word: X_n
- word(X_i)

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 non-extendible 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 non-extendible a^n by a single symbol.

$a_2 \quad c \quad a_3$

Problems

- easy for text, what about grammar?
- what to do with the NFA?

Local decompression

Re-compression

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

Local decompression

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 $|\text{word}(X_n) > n|$ **do**
  $L_\Sigma \leftarrow$ list of letters, $L_P \leftarrow$ list of pairs
  for $ab \in L_P$ **do**
    compress pair ab, modify N accordingly
  for $a \in L_\Sigma$ **do**
    compress a non-extendible appearances, modify N accordingly

Decompress the word and solve the problem naively.

Outline

Outline of the algorithm

while $|\text{word}(X_n) > n|$ **do**
$L_\Sigma \leftarrow$ list of letters, $L_P \leftarrow$ list of pairs
for $ab \in L_P$ **do**
compress pair ab, modify N accordingly
for $a \in L_\Sigma$ **do**
compress a non-extendible appearances, modify N accordingly
Decompress the word and solve the problem naively.

Theorem

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

Proof.

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

Formalisation

New symbols: letters.

Formalisation

New symbols: letters.

Grammar invariants

- the set of nonterminals is $\{X_n, \ldots, X_1\}$ (the input ones)
- the productions are of the form

$$X_i \to uX_jvX_k \quad \text{or} \quad X_i \to uX_jv \quad \text{or} \quad X_i \to u,$$

- for $X_i \to uX_jvX_k$, the input had a production $X_i \to X_jX_k$

Formalisation

New symbols: letters.

Grammar invariants

- the set of nonterminals is $\{X_n, \ldots, X_1\}$ (the input ones)
- the productions are of the form
$$X_i \to uX_jvX_k \quad \text{or} \quad X_i \to uX_jv \quad \text{or} \quad X_i \to u,$$
- for $X_i \to uX_jvX_k$, the input had a production $X_i \to X_jX_k$

In this way the grammar does not blow up (the skeleton is the same).

NFA invariants

NFA cannot blow up either.

NFA invariants

NFA cannot blow up either.

NFA invariants

- transitions of N:

  letter transitions labelled by a single letter

  nonterminal transition labelled by nonterminal
- nonterminal transition have counterparts in the input NFA
  - the same nonterminal
  - corresponding start and end
  - the same multiplicity

NFA invariants

NFA cannot blow up either.

NFA invariants

- transitions of N:

  letter transitions labelled by a single letter

  nonterminal transition labelled by nonterminal

- nonterminal transition have counterparts in the input NFA
  - the same nonterminal
  - corresponding start and end
  - the same multiplicity

The nonterminal part is 'the same' (of size n).

What is hard, what is easy

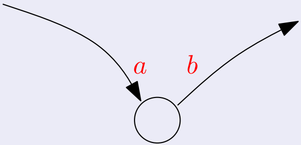
What is hard to compress, what easy?

What is hard, what is easy

What is hard to compress, what easy?

Hard

- a letter a is outer, if it is the first or last letter of some word(X_i)
- a pair ab is crossing if
  - $X_i \to uaX_jvX_k$, where word$(X_j) = b \dots$
  - ab spreads over transitions
    at least one nonterminal

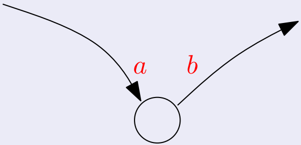


What is hard, what is easy

What is hard to compress, what easy?

Hard

- a letter a is outer, if it is the first or last letter of some word(X_i)
- a pair ab is crossing if
  - $X_i \to uaX_j vX_k$, where word$(X_j) = b\ldots$
  - ab spreads over transitions
    at least one nonterminal



Easy

- a letter a is inner otherwise
- a pair ab is non-crossing otherwise

A little detailed outline

Detailed outline

while $|\text{word}(X_n) > n|$ **do**
  while possible **do**
    for a: inner letter **do**
      compress appearances of a
    for non-crossing pair ab in $\text{word}(X_n)$ **do**
      compress ab

A little detailed outline

Detailed outline

while $|\text{word}(X_n) > n|$ **do**
  while possible **do**
    for a: inner letter **do**
      compress appearances of a
    for non-crossing pair ab in $\text{word}(X_n)$ **do**
      compress ab
  $L \leftarrow$ list of all outer letters
  for $a \in L$ **do**
    compress appearances of a
    for each $a_k b$ in $\text{word}(X_n)$ **do**
      compress $a_k b$

A little detailed outline

Detailed outline

while $|\text{word}(X_n) > n|$ **do**
- **while** possible **do**
  - **for** a: inner letter **do**
    - compress appearances of a
  - **for** non-crossing pair ab in $\text{word}(X_n)$ **do**
    - compress ab
- $L \leftarrow$ list of all outer letters
- **for** $a \in L$ **do**
  - compress appearances of a
  - **for** each $a_k b$ in $\text{word}(X_n)$ **do**
    - compress $a_k b$

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

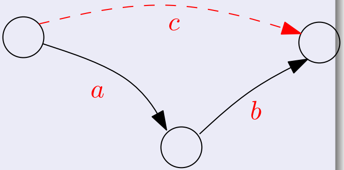
for each production $X_i \to u X_j v X_k$ **do**
  replace each ab in u, v by c

for states p, q **do**
  if $\delta_N(p, ab, q)$ **then**
    put a transition $\delta_N(p, c, q)$

Non-crossing pair compression

Non-crossing pair compression

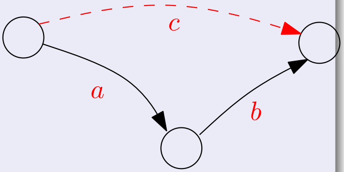
for each production $X_i \to uX_jvX_k$ **do**
  replace each ab in u, v by c
for states p, q **do**
  if $\delta_N(p, ab, q)$ **then**
    put a transition $\delta_N(p, c, q)$



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
for states p, q **do**
  if $\delta_N(p, ab, q)$ **then**
    put a transition $\delta_N(p, c, q)$



Lemma

It works.

Proof.

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

Inner letter compression

Appearance compression for and inner letter a

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

Inner letter compression

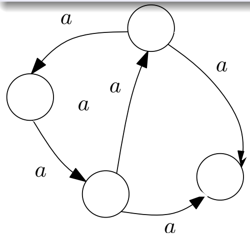
Appearance compression for and inner letter a

compute the lengths $\ell_1, \ldots, \ell_k$ of a's non-extendible appearance
for each a^{ℓ_m} **do**
  for each production $X_i \to uX_jvX_k$ **do**
    replace non-extendible a^{ℓ_m} in in u, v by a_{ℓ_m}
  for states p, q in N **do**
    guess, whether $\delta_N(p, a^{\ell_m}, q)$
    if guess is positive **then**
      verify the guess
      put a transition $\delta_{N'}(p, a_{\ell_m}, q)$

Inner letter compression

Appearance compression for and inner letter a

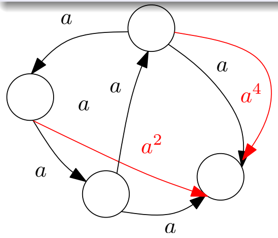
compute the lengths $\ell_1, \ldots, \ell_k$ of a's non-extendible appearance
for each a^{ℓ_m} **do**
  for each production $X_i \to uX_j vX_k$ **do**
    replace non-extendible a^{ℓ_m} in in u, v by a_{ℓ_m}
  for states p, q in N **do**
    guess, whether $\delta_N(p, a^{\ell_m}, q)$
    if guess is positive **then**
      verify the guess
      put a transition $\delta_{N'}(p, a_{\ell_m}, q)$



Inner letter compression

Appearance compression for and inner letter a

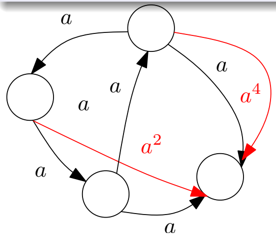
compute the lengths $\ell_1, \ldots, \ell_k$ of a's non-extendible appearance
for each a^{ℓ_m} **do**
  for each production $X_i \to uX_jvX_k$ **do**
    replace non-extendible a^{ℓ_m} in in u, v by a_{ℓ_m}
  for states p, q in N **do**
    guess, whether $\delta_N(p, a^{\ell_m}, q)$
    if guess is positive **then**
      verify the guess
      put a transition $\delta_{N'}(p, a_{\ell_m}, q)$



Inner letter compression

Appearance compression for and inner letter a

compute the lengths $\ell_1, \ldots, \ell_k$ of a's non-extendible appearance
for each a^{ℓ_m} **do**
  for each production $X_i \to uX_jvX_k$ **do**
    replace non-extendible a^{ℓ_m} in in u, v by a_{ℓ_m}
  for states p, q in N **do**
    guess, whether $\delta_N(p, a^{\ell_m}, q)$
    if guess is positive **then**
      verify the guess
      put a transition $\delta_{N'}(p, a_{\ell_m}, q)$



Lemma

It works.

- how many lengths: inner letter
- verification: Plandowski & Rytter result

Crossing pairs and outer letters

Lemma

$\mathcal{O}(n)$ outer letters and $\mathcal{O}(n^2)$ crossing pairs appearing in $\text{word}(X_n)$

Crossing pairs and outer letters

Lemma

$\mathcal{O}(n)$ outer letters and $\mathcal{O}(n^2)$ crossing pairs appearing in $\text{word}(X_n)$

Proof.

- charge outer letter to the non-terminal

Crossing pairs and outer letters

Lemma

$\mathcal{O}(n)$ outer letters and $\mathcal{O}(n^2)$ crossing pairs appearing in $\text{word}(X_n)$

Proof.

- charge outer letter to the non-terminal
- pairs in $\text{word}(X_n)$:
  - appears is some explicit word $uX_j vX_k$, total size $\mathcal{O}(n^2)$.
  - spreads over $u\,\text{word}(X_j)$,
    charged to a production $X_i \to uX_j vX_k$, total size $\mathcal{O}(n)$. □

Convert hard to easy

Convert outer letters to inner, crossing pairs to non-crossing (Sequentially).

Convert hard to easy

Convert outer letters to inner, crossing pairs to non-crossing (Sequentially).

fix an outer letter a
convert it to inner
compress a appearances
make each pair of the form $a_\ell b$ non-crossing
compress each such pair

Turning a into an inner letter

- Cut a-prefix and a-suffix from each nonterminal.
- Represent word(X_i) as $a^{\ell_i} w a^{r_i}$, turn it into w.
  - $X_i \to uX_j vX_k$
  - $\text{word}(X_i) = u\,\text{word}(X_j)\,v\,\text{word}(X_k)$

Turning a into an inner letter

- Cut a-prefix and a-suffix from each nonterminal.
- Represent word(X_i) as $a^{\ell_i} w a^{r_i}$, turn it into w.
  - $X_i \to u X_j v X_k$
  - $\text{word}(X_i) = u\,\text{word}(X_j)v\,\text{word}(X_k)$

Changing an outer letter a to an inner one

for $i = 1 \,..\, n$ **do**
  let $X_i \to u X_j v X_k$
  replace $X_i \to u a^{\ell_j} X'_j a^{r_j} v a^{\ell_k} X'_k a^{r_k}$
  calculate the a-prefix a^{ℓ_i} and a-suffix a^{r_i}, remove them

Turning a into an inner letter

- Cut a-prefix and a-suffix from each nonterminal.
- Represent $\text{word}(X_i)$ as $a^{\ell_i} w a^{r_i}$, turn it into w.
  - $X_i \to uX_j vX_k$
  - $\text{word}(X_i) = u\,\text{word}(X_j) v\,\text{word}(X_k)$

Changing an outer letter a to an inner one

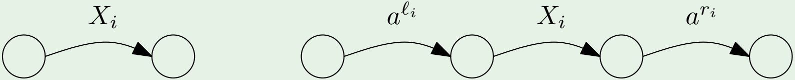
for $i = 1 \,.\,.\, n$ **do**
  let $X_i \to uX_j vX_k$
  replace $X_i \to u a^{\ell_j} X'_j a^{r_j} v a^{\ell_k} X'_k a^{r_k}$
  calculate the a-prefix a^{ℓ_i} and a-suffix a^{r_i}, remove them

Lemma

The algorithm makes a an inner letter.

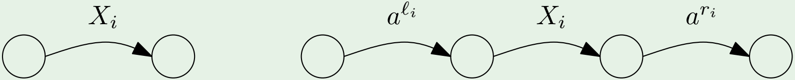
Adjusting the NFA

Adjust the transitions $\delta(p, X_i, q)$



Adjusting the NFA

Adjust the transitions $\delta(p, X_i, q)$

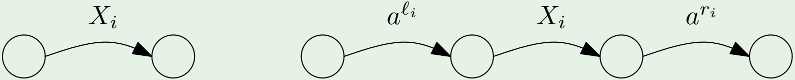


Lemma

This adjusts the NFA properly.

Adjusting the NFA

Adjust the transitions $\delta(p, X_i, q)$



Lemma

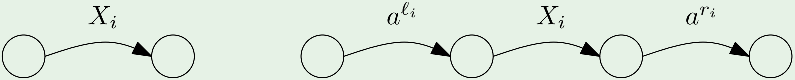
This adjusts the NFA properly.

a^ℓ

a^ℓ in NFA and grammar.

Adjusting the NFA

Adjust the transitions $\delta(p, X_i, q)$



Lemma

This adjusts the NFA properly.

a^ℓ

a^ℓ in NFA and grammar.

Appearance compression

- guess and verify, whether $\delta_N(p, a^\ell, q)$
- by Plandowski & Rytter result (unary case in NP)

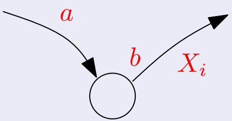
Crossing to non-crossing

Turn pairs $a_\ell b$ into non-crossing.

Crossing to non-crossing

Turn pairs $a_\ell b$ into non-crossing.

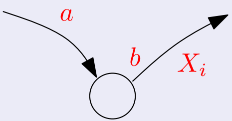
- a_ℓ is inner
- 'pop' first letter of each non-terminal
- replace: $\text{word}(X_i) = bu \mapsto \text{word}(X_i) = u$



Crossing to non-crossing

Turn pairs $a_\ell b$ into non-crossing.

- a_ℓ is inner
- 'pop' first letter of each non-terminal
- replace: $\text{word}(X_i) = bu \mapsto \text{word}(X_i) = u$



Lemma

After popping letters, no pair $a_\ell b$ is crossing.

Proof.

Easy, but goes into details. □

Sizes and running time

Running time

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

Sizes and running time

Running time

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

Size of G

$$abbbcceaX_j addfeaaf X_k$$

In each iteration

Sizes and running time

Running time

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

Size of G

$$abbbccea\color{red}{bha}\color{black}X_j\color{red}{ab}\color{black}addfeaaf\,\color{red}{cda}\color{black}X_k$$

In each iteration

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

Sizes and running time

Running time

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

Size of G

$$abbbcceabhaX_jabaddfeaafcdaX_k$$

In each iteration

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

Sizes and running time

Running time

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

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 $npoly(n, |G|, |\Sigma|, |N|)$.

Size of G

$$uvbhaX_jabxyzcdaX_k$$

In each iteration

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

New letters ($|\Sigma|$)

- noncrossing pairs, inner letters appearance compression (shrinks $|G|$)
- outer letters and crossing pairs:
  there are $\mathcal{O}(n)$ outer letters and $\mathcal{O}(n^2)$ pairs in word(X_n)

Proof end

NFA size

- size $\mathcal{O}(|Q| \cdot |\Sigma| + n)$
- new states: replacing nonterminal transitions by chains, (for outer letters, $poly(n)$)

Proof end

NFA size

- size $\mathcal{O}(|Q| \cdot |\Sigma| + n)$
- new states: replacing nonterminal transitions by chains, (for outer letters, $poly(n)$)

And this is it.

DFA

DFA vs NFA

Non-determinism: guessing and verifying transitions $\delta(p, a^\ell, q)$.

- for DFA: easy
- operations preserve determinism of DFA