Pairs of Complementary Unary Languages with "Balanced" Nondeterministic Automata

Viliam Geffert[1] Giovanni Pighizzini[2]

[1]Department of Computer Science
P. J. Šafárik University
Košice, Slovakia

[2]Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano
Milano, Italy

LATIN 2010 – Oaxaca, Mexico – April 20th, 2010

A general problem

Compare the number of the states of *complementary automata*, i.e, automata accepting a regular language and its complement:

Given an n-state automaton accepting L, how many states are necessary and sufficient to accept L^c?

- Deterministic automata (DFAs): trivial.
- Nondeterministic automata (NFAs):
  - trivial upper bound: 2^n, optimal [Birget 1993]
  - differences between the general case and the case of *unary languages*. [Mera&Pighizzini 2005]

A general problem

Compare the number of the states of *complementary automata*, i.e, automata accepting a regular language and its complement:

Given an n-state automaton accepting L, how many states are necessary and sufficient to accept L^{c}?

- Deterministic automata (DFAs): trivial.
- Nondeterministic automata (NFAs):
  - trivial upper bound: 2^n, optimal [Birget 1993]
  - differences between the general case and the case of *unary languages*. [Mera&Pighizzini 2005]

A general problem

Compare the number of the states of *complementary automata*, i.e, automata accepting a regular language and its complement:

Given an n-state automaton accepting L, how many states are necessary and sufficient to accept L^{c}?

- Deterministic automata (DFAs): trivial.
- Nondeterministic automata (NFAs):
  - trivial upper bound: 2^n, optimal [Birget 1993]
  - differences between the general case and the case of *unary languages*. [Mera&Pighizzini 2005]

A general problem

Compare the number of the states of *complementary automata*, i.e, automata accepting a regular language and its complement:

Given an n-state automaton accepting L, how many states are necessary and sufficient to accept L^c?

- Deterministic automata (DFAs): trivial.
- Nondeterministic automata (NFAs):
  - trivial upper bound: 2^n, optimal [Birget 1993]
  - differences between the general case and the case of *unary languages*. [Mera&Pighizzini 2005]

Size of complementary NFAs: general vs. unary case

There are languages having "small" complementary NFAs:

For each integer n there exists a regular language L such that:

- L is accepted by an n-state NFA,
- L^c is accepted by an NFA with at most $n+1$ states,
- the minimum DFA accepting L requires 2^n states.

Hence:

- L is a witness of the maximal state gap between NFAs and DFAs,
- the gap between the total size of smallest NFAs accepting L and L^c and corresponding DFAs is exponential.

The language L is defined over a two letter alphabet.

The unary case looks completely different:

If a unary language L has a "small" NFA then each NFA accepting L^c must be "large".

There are languages having "small" complementary NFAs:

For each integer n there exists a regular language L such that:

- L is accepted by an n-state NFA,
- L^c is accepted by an NFA with at most $n+1$ states,
- the minimum DFA accepting L requires 2^n states.

Hence:

- L is a witness of the maximal state gap between NFAs and DFAs,
- the gap between the total size of smallest NFAs accepting L and L^c and corresponding DFAs is exponential.

The language L is defined over a two letter alphabet.

The unary case looks completely different:

If a unary language L has a "small" NFA then
each NFA accepting L^c must be "large".

Size of complementary NFAs: general vs. unary case

There are languages having "small" complementary NFAs:

For each integer n there exists a regular language L such that:

- L is accepted by an n-state NFA,
- L^c is accepted by an NFA with at most $n + 1$ states,
- the minimum DFA accepting L requires 2^n states.

Hence:

- L is a witness of the maximal state gap between NFAs and DFAs,
- the gap between the total size of smallest NFAs accepting L and L^c and corresponding DFAs is exponential.

The language L is defined over a two letter alphabet.

The unary case looks completely different:

If a unary language L has a "small" NFA then each NFA accepting L^c must be "large".

There are languages having "small" complementary NFAs:

For each integer n there exists a regular language L such that:

- L is accepted by an n-state NFA,
- L^c is accepted by an NFA with at most $n+1$ states,
- the minimum DFA accepting L requires 2^n states.

Hence:

- L is a witness of the maximal state gap between NFAs and DFAs,
- the gap between the total size of smallest NFAs accepting L and L^c and corresponding DFAs is exponential.

The language L is defined over a two letter alphabet.

The unary case looks completely different:

If a unary language L has a "small" NFA then each NFA accepting L^c must be "large".

Size of complementary NFAs: general vs. unary case

There are languages having "small" complementary NFAs:

For each integer n there exists a regular language L such that:

- L is accepted by an n-state NFA,
- L^{c} is accepted by an NFA with at most $n+1$ states,
- the minimum DFA accepting L requires 2^n states.

Hence:

- L is a witness of the maximal state gap between NFAs and DFAs,
- the gap between the total size of smallest NFAs accepting L and L^{c} and corresponding DFAs is exponential.

The language L is defined over a two letter alphabet.

The unary case looks completely different:

If a unary language L has a "small" NFA then
each NFA accepting L^{c} must be "large".

Unary case $\Sigma = \{a\}$

The cost of the optimal simulation of n-state NFAs by DFAs in the unary case reduces from 2^n to the function $F(n) = e^{\Theta(\sqrt{n \cdot \ln n})}$, which is subexponential but superpolynomial. [Chrobak 1986]

However:

If L is a unary language accepted by an n-state NFA s.t. the minimum equivalent DFA requires $F(n)$ states, then also *each* NFA *accepting* L^c *requires at least* $F(n)$ *states.* [Mera&Pighizzini 2005]

In other words:

- If L is a witness of the maximal state gap between unary NFAs and equivalent DFAs then each NFA for L^c must have as many states as the minimum DFA.
- Hence, taking into account the total number of states of smallest NFAs accepting L and L^c, the superpolynomial gap with the size of DFAs disappears.

Unary case $\Sigma = \{a\}$

The cost of the optimal simulation of n-state NFAs by DFAs in the unary case reduces from 2^n to the function $F(n) = e^{\Theta(\sqrt{n \cdot \ln n})}$, which is subexponential but superpolynomial. [Chrobak 1986]

However:

If L is a unary language accepted by an n-state NFA s.t. the minimum equivalent DFA requires $F(n)$ states, then also *each* NFA *accepting* L^c *requires at least* $F(n)$ *states.* [Mera&Pighizzini 2005]

In other words:

- If L is a witness of the maximal state gap between unary NFAs and equivalent DFAs then each NFA for L^c must have as many states as the minimum DFA.
- Hence, taking into account the total number of states of smallest NFAs accepting L and L^c, the superpolynomial gap with the size of DFAs disappears.

Unary case $\Sigma = \{a\}$

The cost of the optimal simulation of n-state NFAs by DFAs in the unary case reduces from 2^n to the function $F(n) = e^{\Theta(\sqrt{n \cdot \ln n})}$, which is subexponential but superpolynomial. [Chrobak 1986]

However:

If L is a unary language accepted by an n-state NFA s.t. the minimum equivalent DFA requires $F(n)$ states, then also *each* NFA *accepting L^c requires at least $F(n)$ states.* [Mera&Pighizzini 2005]

In other words:

- If L is a witness of the maximal state gap between unary NFAs and equivalent DFAs then each NFA for L^c must have as many states as the minimum DFA.
- Hence, taking into account the total number of states of smallest NFAs accepting L and L^c, the superpolynomial gap with the size of DFAs disappears.

Unary case $\Sigma = \{a\}$

The cost of the optimal simulation of n-state NFAs by DFAs in the unary case reduces from 2^n to the function $F(n) = e^{\Theta(\sqrt{n \cdot \ln n})}$, which is subexponential but superpolynomial. [Chrobak 1986]

However:

If L is a unary language accepted by an n-state NFA s.t. the minimum equivalent DFA requires $F(n)$ states, then also *each* NFA *accepting* L^c *requires at least* $F(n)$ *states.* [Mera&Pighizzini 2005]

In other words:

- If L is a witness of the maximal state gap between unary NFAs and equivalent DFAs then each NFA for L^c must have as many states as the minimum DFA.
- Hence, taking into account the total number of states of smallest NFAs accepting L and L^c, the superpolynomial gap with the size of DFAs disappears.

Paper contributions

How large can be the gap between the total size of pairs of NFAs accepting a unary language and its complement and the minimum DFA ?

We prove that this gap is *superpolynomial*, not too far from $F(n)$:

There are infinitely many unary languages L such that:

- the state gap between NFAs and DFAs accepting L is a little bit smaller than $F(n)$, but it is still superpolynomial,
- the same gap is achieved in the case of L^c.

We also prove superpolynomial gaps between:

- the sizes of unary *unambiguous automata* and of DFAs,
- the sizes of unary *self-verifying automata* and of DFAs.

Paper contributions

How large can be the gap between the total size of pairs of NFAs accepting a unary language and its complement and the minimum DFA ?

We prove that this gap is *superpolynomial*, not too far from $F(n)$:

There are infinitely many unary languages L such that:

- the state gap between NFAs and DFAs accepting L is a little bit smaller than $F(n)$, but it is still superpolynomial,
- the same gap is achieved in the case of L^c.

We also prove superpolynomial gaps between:

- the sizes of unary *unambiguous automata* and of DFAs,
- the sizes of unary *self-verifying automata* and of DFAs.

Paper contributions

How large can be the gap between the total size of pairs of NFAs accepting a unary language and its complement and the minimum DFA ?

We prove that this gap is *superpolynomial*, not too far from $F(n)$:

There are infinitely many unary languages L such that:

- the state gap between NFAs and DFAs accepting L is a little bit smaller than $F(n)$, but it is still superpolynomial,
- the same gap is achieved in the case of L^{c}.

We also prove superpolynomial gaps between:

- the sizes of unary *unambiguous automata* and of DFAs,
- the sizes of unary *self-verifying automata* and of DFAs.

Paper contributions

How large can be the gap between the total size of pairs of NFAs accepting a unary language and its complement and the minimum DFA ?

We prove that this gap is *superpolynomial*, not too far from $F(n)$:

There are infinitely many unary languages L such that:

- the state gap between NFAs and DFAs accepting L is a little bit smaller than $F(n)$, but it is still superpolynomial,
- the same gap is achieved in the case of L^{c}.

We also prove superpolynomial gaps between:

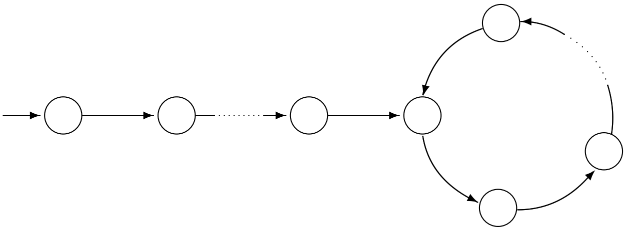
- the sizes of unary *unambiguous automata* and of DFAs,
- the sizes of unary *self-verifying automata* and of DFAs.

Input alphabet $\Sigma = \{a\}$

As a special case of unary regular languages are *cyclic languages*:

$L \subseteq \{a\}^*$ is said to be *cyclic* iff it is accepted by a DFA whose transition graph is just one loop.

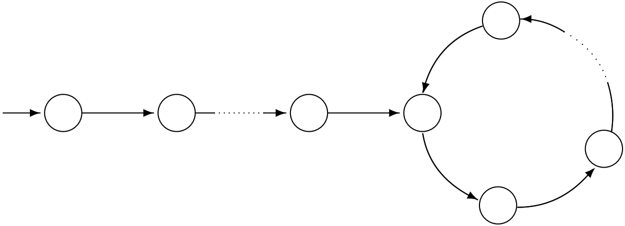
Input alphabet $\Sigma = \{a\}$



As a special case of unary regular languages are *cyclic languages*:

$L \subseteq \{a\}^*$ is said to be *cyclic* iff it is accepted by a DFA whose transition graph is just one loop.

Input alphabet $\Sigma = \{a\}$



As a special case of unary regular languages are *cyclic languages*:

$L \subseteq \{a\}^*$ is said to be *cyclic* iff it is accepted by a DFA whose transition graph is just one loop.

Let us consider:

- $\lambda_0, \lambda_1, \ldots, \lambda_{s-1}$ a sequence of $s \geq 1$ powers of different primes,
- the smallest integer $\hat{s} \geq s$ dividing one λ_ℓ, $\ell \in \{0, \ldots, s-1\}$.

We define the language:

$$L = \bigcup_{i=0}^{s-1} \{a^k \mid k \bmod \hat{s} = i = k \bmod \lambda_i\}.$$

How to recognize L and L^c?

The witness languages

Let us consider:

- $\lambda_0, \lambda_1, \ldots, \lambda_{s-1}$ a sequence of $s \geq 1$ powers of different primes,
- the smallest integer $\hat{s} \geq s$ dividing one λ_ℓ, $\ell \in \{0, \ldots, s-1\}$.

We define the language:

$$L = \bigcup_{i=0}^{s-1} \{a^k \mid k \bmod \hat{s} = i = k \bmod \lambda_i\}.$$

How to recognize L and L^c?

Let us consider:

- $\lambda_0, \lambda_1, \ldots, \lambda_{s-1}$ a sequence of $s \geq 1$ powers of different primes,
- the smallest integer $\hat{s} \geq s$ dividing one λ_ℓ, $\ell \in \{0, \ldots, s-1\}$.

We define the language:

$$L = \bigcup_{i=0}^{s-1} \{a^k \mid k \bmod \hat{s} = i = k \bmod \lambda_i\}.$$

How to recognize L and L^c?

Let us consider:

- $\lambda_0, \lambda_1, \ldots, \lambda_{s-1}$ a sequence of $s \geq 1$ powers of different primes,
- the smallest integer $\hat{s} \geq s$ dividing one λ_ℓ, $\ell \in \{0, \ldots, s-1\}$.

We define the language:

$$L = \bigcup_{i=0}^{s-1} \{a^k \mid k \bmod \hat{s} = i = k \bmod \lambda_i\}.$$

How to recognize L and L^c?

The witness languages

Let us consider:

- $\lambda_0, \lambda_1, \ldots, \lambda_{s-1}$ a sequence of $s \geq 1$ powers of different primes,
- the smallest integer $\hat{s} \geq s$ dividing one λ_ℓ, $\ell \in \{0, \ldots, s-1\}$.

We define the language:

$$L = \bigcup_{i=0}^{s-1} \{a^k \mid k \bmod \hat{s} = i = k \bmod \lambda_i\}.$$

How to recognize L and L^c?

To decide if an input string a^k belongs to L we can use the following procedure:

- $i \leftarrow k \bmod \hat{s}$
- if $i = 0$ then accept iff $k \bmod \lambda_0 = 0$
- if $i = 1$ then accept iff $k \bmod \lambda_1 = 1$
- ...
- if $i = s - 1$ then accept iff $k \bmod \lambda_{s-1} = s - 1$
- if $i \geq s$ then reject

Nondeterministic version:

- guess i, with $0 \leq i < s$
- accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i = i$

The nondeterministic version can be implemented by an NFA A^+.

To decide if an input string a^k belongs to L we can use the following procedure:

- $i \leftarrow k \bmod \hat{s}$
- if $i = 0$ then accept iff $k \bmod \lambda_0 = 0$
- if $i = 1$ then accept iff $k \bmod \lambda_1 = 1$
- ...
- if $i = s - 1$ then accept iff $k \bmod \lambda_{s-1} = s - 1$
- if $i \geq s$ then reject

Nondeterministic version:

- guess i, with $0 \leq i < s$
- accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i = i$

The nondeterministic version can be implemented by an NFA A^+.

To decide if an input string a^k belongs to L we can use the following procedure:

- $i \leftarrow k \bmod \hat{s}$
- if $i = 0$ then accept iff $k \bmod \lambda_0 = 0$
- if $i = 1$ then accept iff $k \bmod \lambda_1 = 1$
- ...
- if $i = s - 1$ then accept iff $k \bmod \lambda_{s-1} = s - 1$
- if $i \geq s$ then reject

Nondeterministic version:

- guess i, with $0 \leq i < s$
- accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i = i$

The nondeterministic version can be implemented by an NFA A^+.

How to recognize $L = \bigcup_{i=0}^{s-1} \{a^k \mid k \bmod \hat{s} = i = k \bmod \lambda_i\}$

To decide if an input string a^k belongs to L we can use the following procedure:

- $i \leftarrow k \bmod \hat{s}$
- if $i = 0$ then accept iff $k \bmod \lambda_0 = 0$
- if $i = 1$ then accept iff $k \bmod \lambda_1 = 1$
- ...
- if $i = s - 1$ then accept iff $k \bmod \lambda_{s-1} = s - 1$
- if $i \geq s$ then reject

Nondeterministic version:

- guess i, with $0 \leq i < s$
- accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i = i$

The nondeterministic version can be implemented by an NFA A^+.

To decide if an input string a^k belongs to L we can use the following procedure:

- $i \leftarrow k \bmod \hat{s}$
- if $i = 0$ then accept iff $k \bmod \lambda_0 = 0$
- if $i = 1$ then accept iff $k \bmod \lambda_1 = 1$
- ...
- if $i = s - 1$ then accept iff $k \bmod \lambda_{s-1} = s - 1$
- if $i \geq s$ then reject

Nondeterministic version:

- guess i, with $0 \leq i < s$
- accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i = i$

The nondeterministic version can be implemented by an NFA A^+.

How to recognize $L = \bigcup_{i=0}^{s-1}\{a^k \mid k \bmod \hat{s} = i = k \bmod \lambda_i\}$

To decide if an input string a^k belongs to L we can use the following procedure:

- $i \leftarrow k \bmod \hat{s}$
- if $i = 0$ then accept iff $k \bmod \lambda_0 = 0$
- if $i = 1$ then accept iff $k \bmod \lambda_1 = 1$
- ...
- if $i = s - 1$ then accept iff $k \bmod \lambda_{s-1} = s - 1$
- if $i \geq s$ then reject

Nondeterministic version:

- guess i, with $0 \leq i < s$
- accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i = i$

The nondeterministic version can be implemented by an NFA A^+.

How to recognize $L = \bigcup_{i=0}^{s-1} \{a^k \mid k \bmod \hat{s} = i = k \bmod \lambda_i\}$

To decide if an input string a^k belongs to L we can use the following procedure:

- $i \leftarrow k \bmod \hat{s}$
- if $i = 0$ then accept iff $k \bmod \lambda_0 = 0$
- if $i = 1$ then accept iff $k \bmod \lambda_1 = 1$
- ...
- if $i = s - 1$ then accept iff $k \bmod \lambda_{s-1} = s - 1$
- if $i \geq s$ then reject

Nondeterministic version:

- guess i, with $0 \leq i < s$
- accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i = i$

The nondeterministic version can be implemented by an NFA A^+.

To decide if an input string a^k belongs to L we can use the following procedure:

- $i \leftarrow k \bmod \hat{s}$
- if $i = 0$ then accept iff $k \bmod \lambda_0 = 0$
- if $i = 1$ then accept iff $k \bmod \lambda_1 = 1$
- ...
- if $i = s - 1$ then accept iff $k \bmod \lambda_{s-1} = s - 1$
- if $i \geq s$ then reject

Nondeterministic version:

- guess i, with $0 \leq i < s$
- accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i = i$

The nondeterministic version can be implemented by an NFA A^+.

NFA A^+:

- one initial state and s disjoint loops
- in the initial state one among s possible transitions is chosen: guess i, with $0 \le i < s$
- transition i leads to the ith loop, which implements: accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i = i$

length of the ith loop: $\mathrm{lcm}(\hat{s}, \lambda_i)$

Size of A^+:

- Summing up: total number of states: $1 + \sum_{i=0}^{s-1} \mathrm{lcm}(\hat{s}, \lambda_i)$
- However $\mathrm{lcm}(\hat{s}, \lambda_i) = \begin{cases} \hat{s} \cdot \lambda_i, & \text{if } i \neq \ell; \\ \lambda_\ell, & \text{otherwise.} \end{cases}$
- Hence, the total number of the states is:

$$N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i\neq\ell}^{s-1} \lambda_i .$$

An NFA for $L = \bigcup_{i=0}^{s-1}\{a^k \mid k \bmod \hat{s} = i = k \bmod \lambda_i\}$

NFA A^+:

- one initial state and s disjoint loops
- in the initial state one among s possible transitions is chosen: guess i, with $0 \leq i < s$
- transition i leads to the ith loop, which implements: accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i = i$

length of the ith loop: $\text{lcm}(\hat{s}, \lambda_i)$

Size of A^+:

- Summing up: total number of states: $1 + \sum_{i=0}^{s-1} \text{lcm}(\hat{s}, \lambda_i)$
- However $\text{lcm}(\hat{s}, \lambda_i) = \begin{cases} \hat{s} \cdot \lambda_i, & \text{if } i \neq \ell; \\ \lambda_\ell, & \text{otherwise.} \end{cases}$
- Hence, the total number of the states is:

$$N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i \neq \ell}^{s-1} \lambda_i .$$

An NFA for $L = \bigcup_{i=0}^{s-1} \{a^k \mid k \bmod \hat{s} = i = k \bmod \lambda_i\}$

NFA A^+:

- one initial state and s disjoint loops
- in the initial state one among s possible transitions is chosen: guess i, with $0 \leq i < s$
- transition i leads to the ith loop, which implements: accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i = i$

length of the ith loop: $\text{lcm}(\hat{s}, \lambda_i)$

Size of A^+:

- Summing up: total number of states: $1 + \sum_{i=0}^{s-1} \text{lcm}(\hat{s}, \lambda_i)$
- However $\text{lcm}(\hat{s}, \lambda_i) = \begin{cases} \hat{s} \cdot \lambda_i, & \text{if } i \neq \ell; \\ \lambda_\ell, & \text{otherwise.} \end{cases}$
- Hence, the total number of the states is:

$$N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i\neq\ell}^{s-1} \lambda_i.$$

An NFA for $L = \bigcup_{i=0}^{s-1}\{a^k \mid k \bmod \hat{s} = i = k \bmod \lambda_i\}$

NFA A^+:

- one initial state and s disjoint loops
- in the initial state one among s possible transitions is chosen: guess i, with $0 \le i < s$
- transition i leads to the ith loop, which implements: accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i = i$

length of the ith loop: $\mathrm{lcm}(\hat{s}, \lambda_i)$

Size of A^+:

- Summing up: total number of states: $1 + \sum_{i=0}^{s-1} \mathrm{lcm}(\hat{s}, \lambda_i)$
- However $\mathrm{lcm}(\hat{s}, \lambda_i) = \begin{cases} \hat{s} \cdot \lambda_i, & \text{if } i \neq \ell; \\ \lambda_\ell, & \text{otherwise.} \end{cases}$
- Hence, the total number of the states is:

$$N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i\neq\ell}^{s-1} \lambda_i .$$

An NFA for $L = \bigcup_{i=0}^{s-1}\{a^k \mid k \bmod \hat{s} = i = k \bmod \lambda_i\}$

NFA A^+:

- one initial state and s disjoint loops
- in the initial state one among s possible transitions is chosen: guess i, with $0 \leq i < s$
- transition i leads to the ith loop, which implements: accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i = i$

length of the ith loop: $\mathrm{lcm}(\hat{s}, \lambda_i)$

Size of A^+:

- Summing up: total number of states: $1 + \sum_{i=0}^{s-1} \mathrm{lcm}(\hat{s}, \lambda_i)$
- However $\mathrm{lcm}(\hat{s}, \lambda_i) = \begin{cases} \hat{s} \cdot \lambda_i, & \text{if } i \neq \ell; \\ \lambda_\ell, & \text{otherwise.} \end{cases}$
- Hence, the total number of the states is:

$$N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i\neq\ell}^{s-1} \lambda_i .$$

An NFA for $L = \bigcup_{i=0}^{s-1}\{a^k \mid k \bmod \hat{s} = i = k \bmod \lambda_i\}$

NFA A^+:

- one initial state and s disjoint loops
- in the initial state one among s possible transitions is chosen: guess i, with $0 \leq i < s$
- transition i leads to the ith loop, which implements: accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i = i$

length of the ith loop: $\mathrm{lcm}(\hat{s}, \lambda_i)$

Size of A^+:

- Summing up: total number of states: $1 + \sum_{i=0}^{s-1} \mathrm{lcm}(\hat{s}, \lambda_i)$
- However $\mathrm{lcm}(\hat{s}, \lambda_i) = \begin{cases} \hat{s} \cdot \lambda_i, & \text{if } i \neq \ell; \\ \lambda_\ell, & \text{otherwise.} \end{cases}$
- Hence, the total number of the states is:

$$N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i\neq\ell}^{s-1} \lambda_i .$$

NFA A^+:

- one initial state and s disjoint loops
- in the initial state one among s possible transitions is chosen: guess i, with $0 \leq i < s$
- transition i leads to the ith loop, which implements: accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i = i$

length of the ith loop: $\mathrm{lcm}(\hat{s}, \lambda_i)$

Size of A^+:

- Summing up: total number of states: $1 + \sum_{i=0}^{s-1} \mathrm{lcm}(\hat{s}, \lambda_i)$
- However $\mathrm{lcm}(\hat{s}, \lambda_i) = \begin{cases} \hat{s} \cdot \lambda_i, & \text{if } i \neq \ell; \\ \lambda_\ell, & \text{otherwise.} \end{cases}$
- Hence, the total number of the states is:

$$N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i\neq\ell}^{s-1} \lambda_i .$$

An NFA for $L = \bigcup_{i=0}^{s-1}\{a^k \mid k \bmod \hat{s} = i = k \bmod \lambda_i\}$

NFA A^+:

- one initial state and s disjoint loops
- in the initial state one among s possible transitions is chosen:
  guess i, with $0 \leq i < s$
- transition i leads to the ith loop, which implements:
  accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i = i$

  length of the ith loop: $\mathrm{lcm}(\hat{s}, \lambda_i)$

Size of A^+:

- Summing up: total number of states: $1 + \sum_{i=0}^{s-1} \mathrm{lcm}(\hat{s}, \lambda_i)$
- However $\mathrm{lcm}(\hat{s}, \lambda_i) = \begin{cases} \hat{s} \cdot \lambda_i, & \text{if } i \neq \ell; \\ \lambda_\ell, & \text{otherwise.} \end{cases}$
- Hence, the total number of the states is:

$$N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i\neq\ell}^{s-1} \lambda_i .$$

An example

Let $\lambda_0 = 2^2, \lambda_1 = 5$. Then $\hat{s} = s = 2$.

$$
\begin{aligned}
L &= \{a^k \mid (k \bmod 2 = 0 = k \bmod 2^2) \vee (k \bmod 2 = 1 = k \bmod 5)\} \\
&= \{a^k \mid (k \bmod 2^2 = 0) \vee (k \bmod 2 = 1 = k \bmod 5)\} \\
&= (a^4)^* \cup a(a^{10})^*.
\end{aligned}
$$

NFA A^+ for L:

$\lambda_0 = 2^2$ states

$\hat{s} \cdot \lambda_1 = 2 \cdot 5$ states

An example

Let $\lambda_0 = 2^2, \lambda_1 = 5$. Then $\hat{s} = s = 2$.

$$\begin{aligned} L &= \{a^k \mid (k \bmod 2 = 0 = k \bmod 2^2) \vee (k \bmod 2 = 1 = k \bmod 5)\} \\ &= \{a^k \mid (k \bmod 2^2 = 0) \vee (k \bmod 2 = 1 = k \bmod 5)\} \\ &= (a^4)^* \cup a(a^{10})^*. \end{aligned}$$

NFA A^+ for L:

$\lambda_0 = 2^2$ states

$\hat{s} \cdot \lambda_1 = 2 \cdot 5$ states

Let $\lambda_0 = 2^2, \lambda_1 = 5$. Then $\hat{s} = s = 2$.

$$\begin{aligned} L &= \{a^k \mid (k \bmod 2 = 0 = k \bmod 2^2) \vee (k \bmod 2 = 1 = k \bmod 5)\} \\ &= \{a^k \mid (k \bmod 2^2 = 0) \vee (k \bmod 2 = 1 = k \bmod 5)\} \\ &= (a^4)^* \cup a(a^{10})^*. \end{aligned}$$

NFA A^+ for L:

$\lambda_0 = 2^2$ states

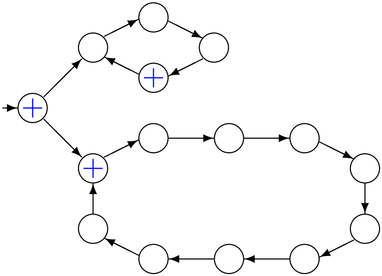
$\hat{s} \cdot \lambda_1 = 2 \cdot 5$ states

An example

Let $\lambda_0 = 2^2, \lambda_1 = 5$. Then $\hat{s} = s = 2$.

$$\begin{aligned} L &= \{a^k \mid (k \bmod 2 = 0 = k \bmod 2^2) \vee (k \bmod 2 = 1 = k \bmod 5)\} \\ &= \{a^k \mid (k \bmod 2^2 = 0) \vee (k \bmod 2 = 1 = k \bmod 5)\} \\ &= (a^4)^* \cup a(a^{10})^*. \end{aligned}$$

NFA A^+ for L:



$\lambda_0 = 2^2$ states

$\hat{s} \cdot \lambda_1 = 2 \cdot 5$ states

How to accept the complement of L

Deterministic procedure to decide whether or not $a^k \in L$:

- $i \leftarrow k \bmod \hat{s}$
- if $i = 0$ then accept iff $k \bmod \lambda_0 = 0$
- if $i = 1$ then accept iff $k \bmod \lambda_1 = 1$
- ...
- if $i = s - 1$ then accept iff $k \bmod \lambda_{s-1} = s - 1$
- if $i \geq s$ then reject

To recognize L^{c} we just need to replace “accept” with ”reject” and vice versa, in the previous procedure.

How to accept the complement of L

Deterministic procedure to decide whether or not $a^k \in L^{\mathrm{c}}$:

- $i \leftarrow k \bmod \hat{s}$
- if $i = 0$ then reject iff $k \bmod \lambda_0 = 0$
- if $i = 1$ then reject iff $k \bmod \lambda_1 = 1$
- ...
- if $i = s - 1$ then reject iff $k \bmod \lambda_{s-1} = s - 1$
- if $i \geq s$ then accept

How to accept the complement of L

Deterministic procedure to decide whether or not $a^k \in L^{\mathrm{c}}$:

- $i \leftarrow k \bmod \hat{s}$
- if $i = 0$ then accept iff $k \bmod \lambda_0 \neq 0$
- if $i = 1$ then accept iff $k \bmod \lambda_1 \neq 1$
- ...
- if $i = s - 1$ then accept iff $k \bmod \lambda_{s-1} \neq s - 1$
- if $i \geq s$ then accept

How to accept the complement of L

Deterministic procedure to decide whether or not $a^k \in L^{\mathrm{c}}$:

- $i \leftarrow k \bmod \hat{s}$
- if $i = 0$ then accept iff $k \bmod \lambda_0 \neq 0$
- if $i = 1$ then accept iff $k \bmod \lambda_1 \neq 1$
- ...
- if $i = s - 1$ then accept iff $k \bmod \lambda_{s-1} \neq s - 1$
- if $i \geq s$ then accept

Nondeterministic version:

- guess i, with $0 \leq i < s$
- if $i < s$ then accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i \neq i$
- if $i \geq s$ then accept iff $k \bmod \hat{s} = i$

Hence:

$$L^{\mathrm{c}} = \bigcup_{i=0}^{s-1} \{a^k \mid k \bmod \hat{s} = i \wedge k \bmod \lambda_i \neq i\} \cup \{a^k \mid k \bmod \hat{s} \geq s)\}$$

How to accept the complement of L

Deterministic procedure to decide whether or not $a^k \in L^c$:

- $i \leftarrow k \bmod \hat{s}$
- if $i = 0$ then accept iff $k \bmod \lambda_0 \neq 0$
- if $i = 1$ then accept iff $k \bmod \lambda_1 \neq 1$
- ...
- if $i = s - 1$ then accept iff $k \bmod \lambda_{s-1} \neq s - 1$
- if $i \geq s$ then accept

Nondeterministic version:

- guess i, with $0 \leq i < s$
- if $i < s$ then accept iff $k \bmod \hat{s} = i$ and $k \bmod \lambda_i \neq i$
- if $i \geq s$ then accept iff $k \bmod \hat{s} = i$

Hence:

$$L^c = \bigcup_{i=0}^{s-1} \{a^k \mid k \bmod \hat{s} = i \wedge k \bmod \lambda_i \neq i\} \cup \{a^k \mid k \bmod \hat{s} \geq s)\}$$

An NFA A^- for $L^{c} = \bigcup_{i=0}^{s-1}\{a^k \mid k \bmod \hat{s} = i \wedge k \bmod \lambda_i \neq i\} \cup \{a^k \mid k \bmod \hat{s} \geq s)\}$

Same transition graph as A^+:

- For $i = 0, \ldots, s-1$, the ith loop is used to accept if $k \bmod \hat{s} = i$ and $k \bmod \lambda_i \neq i$
- however, the $(s-1)$th loop accepts also if $k \bmod \hat{s} \geq s$.

A^+ and A^- have the same size $N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i\neq\ell}^{s-1} \lambda_i$.

It can be observed that A^+ and A^- are *unambiguous*.

An NFA A^- for $L^c = \bigcup_{i=0}^{s-1}\{a^k \mid k \bmod \hat{s} = i \wedge k \bmod \lambda_i \neq i\} \cup \{a^k \mid k \bmod \hat{s} \geq s)\}$

Same transition graph as A^+:

- For $i = 0, \ldots, s-1$, the ith loop is used to accept if $k \bmod \hat{s} = i$ and $k \bmod \lambda_i \neq i$
- however, the $(s-1)$th loop accepts also if $k \bmod \hat{s} \geq s$.

A^+ and A^- have the same size $N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i\neq\ell}^{s-1} \lambda_i$.

It can be observed that A^+ and A^- are *unambiguous*.

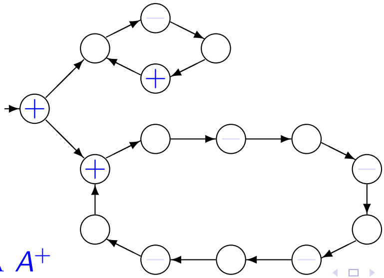
The previous example: how to accept the complement

Let $\lambda_0 = 2^2, \lambda_1 = 5$. Then $\hat{s} = s = 2$.

$$L = \{a^k \mid \text{if } k \text{ is even then } k \bmod 2^2 = 0, \text{ if } k \text{ is odd then } k \bmod 5 = 1\}.$$

Hence:

$$L^c = \{a^k \mid \text{if } k \text{ is even then } k \bmod 2^2 \neq 0, \text{ if } k \text{ is odd then } k \bmod 5 \neq 1\}$$
$$= a^2(a^4)^* \cup (a^3 + a^5 + a^7 + a^9)(a^{10})^*.$$



NFA A^+

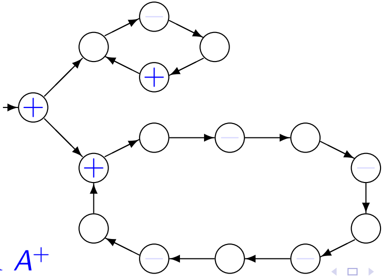
The previous example: how to accept the complement

Let $\lambda_0 = 2^2, \lambda_1 = 5$. Then $\hat{s} = s = 2$.

$$\begin{array}{rcl} L & = & \{a^k \mid \text{if } k \text{ is even then } k \bmod 2^2 = 0, \\ & & \quad \text{if } k \text{ is odd then } k \bmod 5 = 1\}. \end{array}$$

Hence:

$$\begin{array}{rcl} L^{\mathrm{c}} & = & \{a^k \mid \text{if } k \text{ is even then } k \bmod 2^2 \neq 0, \\ & & \quad \text{if } k \text{ is odd then } k \bmod 5 \neq 1\} \\ & = & a^2(a^4)^* \cup (a^3 + a^5 + a^7 + a^9)(a^{10})^*. \end{array}$$



NFA A^+

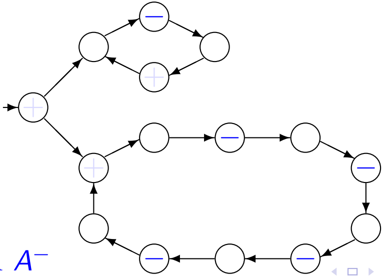
The previous example: how to accept the complement

Let $\lambda_0 = 2^2, \lambda_1 = 5$. Then $\hat{s} = s = 2$.

$$
\begin{aligned}
L &= \{a^k \mid \text{if } k \text{ is even then } k \bmod 2^2 = 0, \\
&\qquad \text{if } k \text{ is odd then } k \bmod 5 = 1\}.
\end{aligned}
$$

Hence:

$$
\begin{aligned}
L^{\mathrm{c}} &= \{a^k \mid \text{if } k \text{ is even then } k \bmod 2^2 \neq 0, \\
&\qquad \text{if } k \text{ is odd then } k \bmod 5 \neq 1\} \\
&= a^2(a^4)^* \cup (a^3 + a^5 + a^7 + a^9)(a^{10})^*.
\end{aligned}
$$



NFA A^-

By previous discussion and by investigating the structure of the minimum DFA accepting L, we proved that:

Theorem

- *The transition graph of the minimum* DFA *A accepting L is a loop of $\lambda_0 \cdot \lambda_1 \cdots \lambda_{s-1}$ states.*
- *L and L^{c} are accepted by two unambiguous* NFAs *A^+ and A^- of at most $N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i\neq\ell}^{s-1} \lambda_i$ states.*

We now study the following question:

How large is the gap between N and $\lambda_0 \cdot \lambda_1 \cdots \lambda_{s-1}$,
i.e, between the size of NFAs A^+, A^-,
and of the minimum DFA A ?

By previous discussion and by investigating the structure of the minimum DFA accepting L, we proved that:

Theorem

- *The transition graph of the minimum* DFA A *accepting* L *is a loop of* $\lambda_0 \cdot \lambda_1 \cdots \lambda_{s-1}$ *states.*
- L *and* L^c *are accepted by two unambiguous* NFAs A^+ *and* A^- *of at most* $N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i\neq\ell}^{s-1} \lambda_i$ *states.*

We now study the following question:

How large is the gap between N and $\lambda_0 \cdot \lambda_1 \cdots \lambda_{s-1}$,
i.e, between the size of NFAs A^+, A^-,
and of the minimum DFA A ?

The witness languages and their automata

By previous discussion and by investigating the structure of the minimum DFA accepting L, we proved that:

Theorem

- *The transition graph of the minimum* DFA A *accepting* L *is a loop of* $\lambda_0 \cdot \lambda_1 \cdots \lambda_{s-1}$ *states.*
- L *and* L^{c} *are accepted by two unambiguous* NFAs A^+ *and* A^- *of at most* $N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i\neq\ell}^{s-1} \lambda_i$ *states.*

We now study the following question:

How large is the gap between N and $\lambda_0 \cdot \lambda_1 \cdots \lambda_{s-1}$,
i.e, between the size of NFAs A^+, A^-,
and of the minimum DFA A ?

By previous discussion and by investigating the structure of the minimum DFA accepting L, we proved that:

Theorem

- *The transition graph of the minimum* DFA A *accepting* L *is a loop of* $\lambda_0 \cdot \lambda_1 \cdots \lambda_{s-1}$ *states.*
- L *and* L^{c} *are accepted by two unambiguous* NFAs A^+ *and* A^- *of at most* $N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i\neq\ell}^{s-1} \lambda_i$ *states.*

We now study the following question:

How large is the gap between N and $\lambda_0 \cdot \lambda_1 \cdots \lambda_{s-1}$,
i.e, between the size of NFAs A^+, A^-,
and of the minimum DFA A ?

- Initially investigated in group theory. [Landau 1903, 1909]
- Fundamental role in the analysis of simulations among various models of unary automata. [Chrobak 1986]

Definition

For a positive integer n:

$$F(n) = \max\{\mathrm{lcm}(\lambda_0, \ldots, \lambda_{s-1}) \mid \lambda_0 + \cdots + \lambda_{s-1} = n\},$$

where $\lambda_0, \ldots, \lambda_{s-1}$ denote arbitrary positive integers.

- Sharp estimation: [Szalay 1980]

$$F(n) = e^{(1+o(1)) \cdot \sqrt{n \cdot \ln n}}$$

The Landau Function $F(n)$

- Initially investigated in group theory. [Landau 1903, 1909]
- Fundamental role in the analysis of simulations among various models of unary automata. [Chrobak 1986]

Definition

For a positive integer n:

$$F(n) = \max\{\mathrm{lcm}(\lambda_0, \ldots, \lambda_{s-1}) \mid \lambda_0 + \cdots + \lambda_{s-1} = n\},$$

where $\lambda_0, \ldots, \lambda_{s-1}$ denote arbitrary positive integers.

- Sharp estimation: [Szalay 1980]

$$F(n) = e^{(1+o(1))\cdot\sqrt{n \cdot \ln n}}$$

The Landau Function $F(n)$

- Initially investigated in group theory. [Landau 1903, 1909]
- Fundamental role in the analysis of simulations among various models of unary automata. [Chrobak 1986]

Definition

For a positive integer n:

$$F(n) = \max\{\mathrm{lcm}(\lambda_0, \ldots, \lambda_{s-1}) \mid \lambda_0 + \cdots + \lambda_{s-1} = n\},$$

where $\lambda_0, \ldots, \lambda_{s-1}$ denote arbitrary positive integers.

- Sharp estimation: [Szalay 1980]

$$F(n) = e^{(1+o(1))\cdot\sqrt{n \cdot \ln n}}$$

The Landau Function $F(n)$

- Initially investigated in group theory. [Landau 1903, 1909]
- Fundamental role in the analysis of simulations among various models of unary automata. [Chrobak 1986]

Definition

For a positive integer n:

$$F(n) = \max\{\text{lcm}(\lambda_0, \ldots, \lambda_{s-1}) \mid \lambda_0 + \cdots + \lambda_{s-1} = n\},$$

where $\lambda_0, \ldots, \lambda_{s-1}$ denote arbitrary positive integers.

- Sharp estimation: [Szalay 1980]

$$F(n) = e^{(1+o(1))\cdot\sqrt{n\cdot\ln n}}$$

$$F(n) = \max\{\operatorname{lcm}(\lambda_0, \ldots, \lambda_{s-1}) \mid \lambda_0 + \cdots + \lambda_{s-1} = n\}$$

We observe that:

- $\operatorname{lcm}(\lambda_0, \ldots, \lambda_{s-1}) = \operatorname{lcm}(\lambda_0, \ldots, \lambda_{s-1}, 1, \ldots, 1)$:
  it is enough to require that $\lambda_0 + \cdots + \lambda_{s-1} \leq n$.
- If $\lambda_i = a \cdot b$ with $\gcd(a, b) = 1$ then $a + b \leq a \cdot b$:
  replacing λ_i with a and b does not increase the sum and does not change the least common multiple of λ_js.
- If λ_i divides λ_j then λ_i can be removed without changing the least common multiple.

Hence:

$$F(n) = \max\{\lambda_0 \cdots \lambda_{s-1} \mid \lambda_0 + \cdots + \lambda_{s-1} \leq n\},$$

where $\lambda_0, \ldots, \lambda_{s-1}$ denote powers of different primes, *exactly as in the definition of the witness language L.*

$$F(n) = \max\{\text{lcm}(\lambda_0, \ldots, \lambda_{s-1}) \mid \lambda_0 + \cdots + \lambda_{s-1} = n\}$$

We observe that:

- $\text{lcm}(\lambda_0, \ldots, \lambda_{s-1}) = \text{lcm}(\lambda_0, \ldots, \lambda_{s-1}, 1, \ldots, 1)$: it is enough to require that $\lambda_0 + \cdots + \lambda_{s-1} \leq n$.
- If $\lambda_i = a \cdot b$ with $\gcd(a, b) = 1$ then $a + b \leq a \cdot b$: replacing λ_i with a and b does not increase the sum and does not change the least common multiple of λ_js.
- If λ_i divides λ_j then λ_i can be removed without changing the least common multiple.

Hence:

$$F(n) = \max\{\lambda_0 \cdots \lambda_{s-1} \mid \lambda_0 + \cdots + \lambda_{s-1} \leq n\},$$

where $\lambda_0, \ldots, \lambda_{s-1}$ denote powers of different primes, *exactly as in the definition of the witness language L.*

$$F(n) = \max\{\mathrm{lcm}(\lambda_0, \ldots, \lambda_{s-1}) \mid \lambda_0 + \cdots + \lambda_{s-1} = n\}$$

We observe that:

- $\mathrm{lcm}(\lambda_0, \ldots, \lambda_{s-1}) = \mathrm{lcm}(\lambda_0, \ldots, \lambda_{s-1}, 1, \ldots, 1)$: it is enough to require that $\lambda_0 + \cdots + \lambda_{s-1} \leq n$.
- If $\lambda_i = a \cdot b$ with $\gcd(a, b) = 1$ then $a + b \leq a \cdot b$: replacing λ_i with a and b does not increase the sum and does not change the least common multiple of λ_js.
- If λ_i divides λ_j then λ_i can be removed without changing the least common multiple.

Hence:

$$F(n) = \max\{\lambda_0 \cdots \lambda_{s-1} \mid \lambda_0 + \cdots + \lambda_{s-1} \leq n\},$$

where $\lambda_0, \ldots, \lambda_{s-1}$ denote powers of different primes, *exactly as in the definition of the witness language L.*

The Landau Function $F(n)$

$$F(n) = \max\{\mathrm{lcm}(\lambda_0, \ldots, \lambda_{s-1}) \mid \lambda_0 + \cdots + \lambda_{s-1} = n\}$$

We observe that:

- $\mathrm{lcm}(\lambda_0, \ldots, \lambda_{s-1}) = \mathrm{lcm}(\lambda_0, \ldots, \lambda_{s-1}, 1, \ldots, 1)$:
  it is enough to require that $\lambda_0 + \cdots + \lambda_{s-1} \leq n$.
- If $\lambda_i = a \cdot b$ with $\gcd(a, b) = 1$ then $a + b \leq a \cdot b$:
  replacing λ_i with a and b does not increase the sum and does not change the least common multiple of λ_js.
- If λ_i divides λ_j then λ_i can be removed without changing the least common multiple.

Hence:

$$F(n) = \max\{\lambda_0 \cdots \lambda_{s-1} \mid \lambda_0 + \cdots + \lambda_{s-1} \leq n\},$$

where $\lambda_0, \ldots, \lambda_{s-1}$ denote powers of different primes, *exactly as in the definition of the witness language L.*

$$F(n) = \max\{\mathrm{lcm}(\lambda_0, \ldots, \lambda_{s-1}) \mid \lambda_0 + \cdots + \lambda_{s-1} = n\}$$

We observe that:

- $\mathrm{lcm}(\lambda_0, \ldots, \lambda_{s-1}) = \mathrm{lcm}(\lambda_0, \ldots, \lambda_{s-1}, 1, \ldots, 1)$: it is enough to require that $\lambda_0 + \cdots + \lambda_{s-1} \leq n$.
- If $\lambda_i = a \cdot b$ with $\gcd(a, b) = 1$ then $a + b \leq a \cdot b$: replacing λ_i with a and b does not increase the sum and does not change the least common multiple of λ_js.
- If λ_i divides λ_j then λ_i can be removed without changing the least common multiple.

Hence:

$$F(n) = \max\{\lambda_0 \cdots \lambda_{s-1} \mid \lambda_0 + \cdots + \lambda_{s-1} \leq n\},$$

where $\lambda_0, \ldots, \lambda_{s-1}$ denote powers of different primes, *exactly as in the definition of the witness language L.*

Witness language and Landau Function

Let us fix an integer n and some powers $\lambda_0, \ldots, \lambda_{s-1}$ of different primes such that:

$$\lambda_0 + \cdots + \lambda_{s-1} \leq n \quad \text{and} \quad F(n) = \lambda_0 \cdots \lambda_{s-1}.$$

The witness language L and its complement are both accepted:

- by NFAs with at most $N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i \neq \ell}^{s-1} \lambda_i$ states,
- by minimum DFAs with $F(n)$ states.

Using:

- some properties of $F(n)$ [Nicolas 1968, Grantham 1995],
- the Bertrand's postulate [Ramanujan 1919],

we proved that $\sqrt{n \cdot \ln n} \geq \Omega(\sqrt[3]{N \cdot \ln^2 N})$.

Hence:

$$F(n) = e^{(1+o(1)) \cdot \sqrt{n \cdot \ln n}} \geq e^{\Omega(\sqrt[3]{N \cdot \ln^2 N})}.$$

Witness language and Landau Function

Let us fix an integer n and some powers $\lambda_0, \ldots, \lambda_{s-1}$ of different primes such that:

$$\lambda_0 + \cdots + \lambda_{s-1} \leq n \quad \text{and} \quad F(n) = \lambda_0 \cdots \lambda_{s-1}.$$

The witness language L and its complement are both accepted:

- by NFAs with at most $N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i\neq\ell}^{s-1} \lambda_i$ states,
- by minimum DFAs with $F(n)$ states.

Using:

- some properties of $F(n)$ [Nicolas 1968, Grantham 1995],
- the Bertrand's postulate [Ramanujan 1919],

we proved that $\sqrt{n \cdot \ln n} \geq \Omega(\sqrt[3]{N \cdot \ln^2 N})$.

Hence:

$$F(n) = e^{(1+o(1)) \cdot \sqrt{n \cdot \ln n}} \geq e^{\Omega(\sqrt[3]{N \cdot \ln^2 N})}.$$

Let us fix an integer n and some powers $\lambda_0, \ldots, \lambda_{s-1}$ of different primes such that:

$$\lambda_0 + \cdots + \lambda_{s-1} \leq n \quad \text{and} \quad F(n) = \lambda_0 \cdots \lambda_{s-1}.$$

The witness language L and its complement are both accepted:

- by NFAs with at most $N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i\neq\ell}^{s-1} \lambda_i$ states,
- by minimum DFAs with $F(n)$ states.

Using:

- some properties of $F(n)$ [Nicolas 1968, Grantham 1995],
- the Bertrand's postulate [Ramanujan 1919],

we proved that $\sqrt{n \cdot \ln n} \geq \Omega(\sqrt[3]{N \cdot \ln^2 N})$.

Hence:

$$F(n) = e^{(1+o(1))\cdot\sqrt{n\cdot\ln n}} \geq e^{\Omega(\sqrt[3]{N\cdot\ln^2 N})}.$$

Witness language and Landau Function

Let us fix an integer n and some powers $\lambda_0, \ldots, \lambda_{s-1}$ of different primes such that:

$$\lambda_0 + \cdots + \lambda_{s-1} \le n \quad \text{and} \quad F(n) = \lambda_0 \cdots \lambda_{s-1}.$$

The witness language L and its complement are both accepted:

- by NFAs with at most $N = 1 + \lambda_\ell + \hat{s} \cdot \sum_{i=0, i\neq\ell}^{s-1} \lambda_i$ states,
- by minimum DFAs with $F(n)$ states.

Using:

- some properties of $F(n)$ [Nicolas 1968, Grantham 1995],
- the Bertrand's postulate [Ramanujan 1919],

we proved that $\sqrt{n \cdot \ln n} \ge \Omega(\sqrt[3]{N \cdot \ln^2 N})$.

Hence:

$$F(n) = e^{(1+o(1))\cdot\sqrt{n\cdot\ln n}} \ge e^{\Omega(\sqrt[3]{N\cdot\ln^2 N})}.$$

The exponential gap

Summing up, for infinitely many N we provided a language L s.t.:

- L and L^{c} are accepted by two NFAs A^{+} and A^{-} using at most N states,
- the minimum DFA accepting L (or L^{c}) must use at least $e^{\Omega(\sqrt[3]{N \cdot \ln^2 N})}$ states.

Hence:

The gap between the total number of states of the pair of complementary unary NFAs A^{+}, A^{-} and the minimum equivalent DFA is superpolynomial.

Remark: actually we can show the existence of a witness language *L for each sufficiently large N*.

Summing up, for infinitely many N we provided a language L s.t.:

- L and L^{c} are accepted by two NFAs A^+ and A^- using at most N states,
- the minimum DFA accepting L (or L^{c}) must use at least $e^{\Omega(\sqrt[3]{N \cdot \ln^2 N})}$ states.

Hence:

The gap between the total number of states of the pair of complementary unary NFAs A^+, A^- and the minimum equivalent DFA is superpolynomial.

Remark: actually we can show the existence of a witness language *L for each sufficiently large N.*

Summing up, for infinitely many N we provided a language L s.t.:

- L and L^c are accepted by two NFAs A^+ and A^- using at most N states,
- the minimum DFA accepting L (or L^c) must use at least $e^{\Omega(\sqrt[3]{N \cdot \ln^2 N})}$ states.

Hence:

The gap between the total number of states of the pair of complementary unary NFAs A^+, A^- and the minimum equivalent DFA is superpolynomial.

Remark: actually we can show the existence of a witness language *L for each sufficiently large N*.

Finite automata with a "symmetric form" of nondeterminism [Ďuriš, Hromkovič, Rolim & Schnitger 1977].

The state set is partitioned in three groups:

- *accepting* states ("yes")
- *rejecting* states ("no")
- *neutral* states ("I do not know")

It is required that:

- on each input string at least one accepting or one rejecting state is reached,
- on a same input string both accepting and rejecting states are not reachable.

Self-verifying automata (SVFAs)

Finite automata with a "symmetric form" of nondeterminism [Ďuriš, Hromkovič, Rolim & Schnitger 1977].

The state set is partitioned in three groups:

- *accepting* states ("yes")
- *rejecting* states ("no")
- *neutral* states ("I do not know")

It is required that:

- on each input string at least one accepting or one rejecting state is reached,
- on a same input string both accepting and rejecting states are not reachable.

Self-verifying automata (SVFAs)

Finite automata with a "symmetric form" of nondeterminism [Ďuriš, Hromkovič, Rolim & Schnitger 1977].

The state set is partitioned in three groups:

- *accepting* states ("yes")
- *rejecting* states ("no")
- *neutral* states ("I do not know")

It is required that:

- on each input string at least one accepting or one rejecting state is reached,
- on a same input string both accepting and rejecting states are not reachable.

Self-verifying automata (SVFAs)

SVFAs characterize the class of regular languages.

Each n-state SVFA can be converted into an equivalent DFA with $O(3^{n/3}) \approx O(1.45^n)$ states. This cost is tight, *for an input alphabet of at least two letters*. [Jirásková & Pighizzini 2009]

What about the tight cost in the unary case?

- It must be strictly smaller than $F(n)$, the cost of the conversion of unary NFAs into DFAs. [J&P 2009]
- As a consequence of the gap between complementary unary NFAs and DFAs we get that this cost is superpolynomial, not too far from $F(n)$.
- This is proved by observing how a SVFA can be obtained from two complementary NFAs.

Self-verifying automata (SVFAs)

SVFAs characterize the class of regular languages.

Each n-state SVFA can be converted into an equivalent DFA with $O(3^{n/3}) \approx O(1.45^n)$ states. This cost is tight, *for an input alphabet of at least two letters*. [Jirásková & Pighizzini 2009]

What about the tight cost in the unary case?

- It must be strictly smaller than $F(n)$, the cost of the conversion of unary NFAs into DFAs. [J&P 2009]
- As a consequence of the gap between complementary unary NFAs and DFAs we get that this cost is superpolynomial, not too far from $F(n)$.
- This is proved by observing how a SVFA can be obtained from two complementary NFAs.

Self-verifying automata (SVFAs)

SVFAs characterize the class of regular languages.

Each n-state SVFA can be converted into an equivalent DFA with $O(3^{n/3}) \approx O(1.45^n)$ states. This cost is tight, *for an input alphabet of at least two letters*. [Jirásková & Pighizzini 2009]

What about the tight cost in the unary case?

- It must be strictly smaller than $F(n)$, the cost of the conversion of unary NFAs into DFAs. [J&P 2009]
- As a consequence of the gap between complementary unary NFAs and DFAs we get that this cost is superpolynomial, not too far from $F(n)$.
- This is proved by observing how a SVFA can be obtained from two complementary NFAs.

Self-verifying automata (SVFAs)

SVFAs characterize the class of regular languages.

Each n-state SVFA can be converted into an equivalent DFA with $O(3^{n/3}) \approx O(1.45^n)$ states. This cost is tight, *for an input alphabet of at least two letters*. [Jirásková & Pighizzini 2009]

What about the tight cost in the unary case?

- It must be strictly smaller than $F(n)$, the cost of the conversion of unary NFAs into DFAs. [J&P 2009]
- As a consequence of the gap between complementary unary NFAs and DFAs we get that this cost is superpolynomial, not too far from $F(n)$.
- This is proved by observing how a SVFA can be obtained from two complementary NFAs.

Self-verifying automata (SVFAs)

SVFAs characterize the class of regular languages.

Each n-state SVFA can be converted into an equivalent DFA with $O(3^{n/3}) \approx O(1.45^n)$ states. This cost is tight, *for an input alphabet of at least two letters*. [Jirásková & Pighizzini 2009]

What about the tight cost in the unary case?

- It must be strictly smaller than $F(n)$, the cost of the conversion of unary NFAs into DFAs. [J&P 2009]
- As a consequence of the gap between complementary unary NFAs and DFAs we get that this cost is superpolynomial, not too far from $F(n)$.
- This is proved by observing how a SVFA can be obtained from two complementary NFAs.

Self-verifying automata (SVFAs)

SVFAs characterize the class of regular languages.

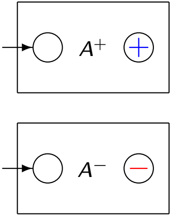
Each n-state SVFA can be converted into an equivalent DFA with $O(3^{n/3}) \approx O(1.45^n)$ states. This cost is tight, *for an input alphabet of at least two letters*. [Jirásková & Pighizzini 2009]

What about the tight cost in the unary case?

- It must be strictly smaller than $F(n)$, the cost of the conversion of unary NFAs into DFAs. [J&P 2009]
- As a consequence of the gap between complementary unary NFAs and DFAs we get that this cost is superpolynomial, not too far from $F(n)$.
- This is proved by observing how a SVFA can be obtained from two complementary NFAs.

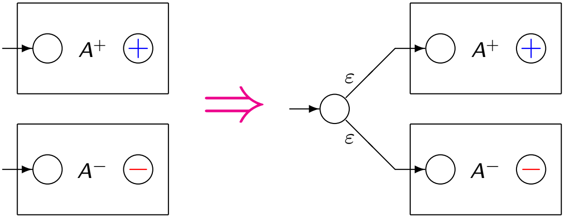
From pairs of complementary NFAs to a SVFAs

Let A^+ and A^- be two complementary NFAs.
We can build an equivalent SVFA A as the "union" of A^+ and A^-, using a new initial state:



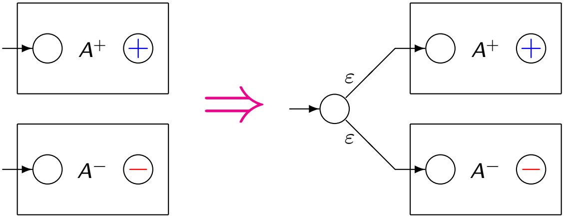
However, if A^+ and A^- have the same transition graph we can do better...

Let A^+ and A^- be two complementary NFAs.
We can build an equivalent SVFA A as the "union" of A^+ and A^-, using a new initial state:

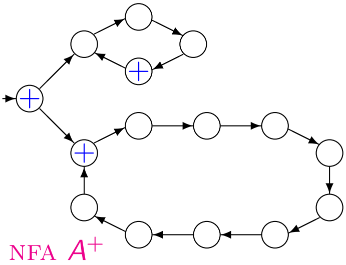


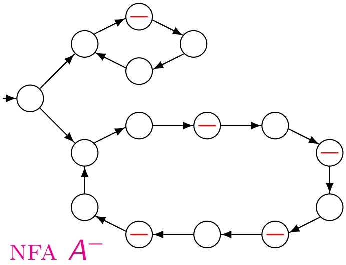
However, if A^+ and A^- have the same transition graph we can do better...

Let A^+ and A^- be two complementary NFAs.
We can build an equivalent SVFA A as the "union" of A^+ and A^-, using a new initial state:

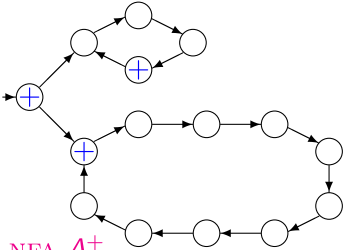


However, if A^+ and A^- have the same transition graph we can do better...

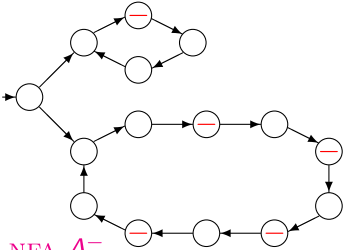

NFA A^+


NFA A^-

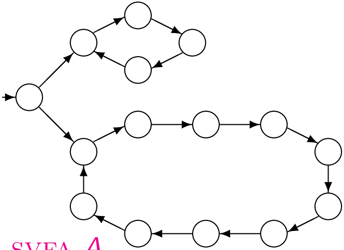
From our witness NFAs to SVFAs



NFA A^+

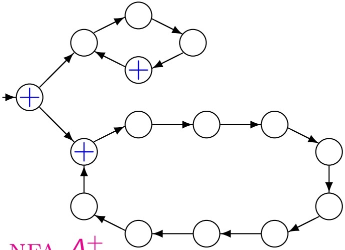


NFA A^-

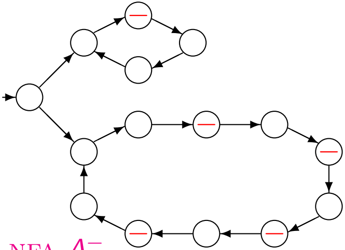


SVFA A

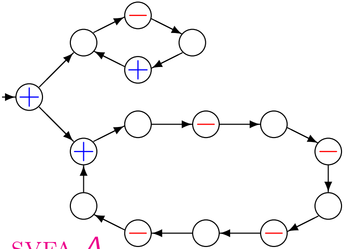
From our witness NFAs to SVFAs



NFA A^+



NFA A^-



SVFA A

Same size!

Conclusion

We proved that the following gaps are superpolynomial, *even for unary cyclic languages:*

- between the total size of two NFAs accepting one language and its complement and the size of the corresponding DFA,
- between the sizes of SVFAs and DFAs.

The witness NFAs A^+ and A^- we used are *unambiguous.*
Hence, also the following gaps are superpolynomial for unary cyclic languages:

- between the sizes of unambiguous NFAs and DFAs,
- between the sizes of unambiguous SVFAs and DFAs.

The superpolynomial function in all these gaps is $e^{\Omega(\sqrt[3]{N \cdot \ln^2 N})}$.
We strongly believe that for unary languages these gaps cannot be significantly improved.

Conclusion

We proved that the following gaps are superpolynomial, *even for unary cyclic languages:*

- between the total size of two NFAs accepting one language and its complement and the size of the corresponding DFA,
- between the sizes of SVFAs and DFAs.

The witness NFAs A^+ and A^- we used are *unambiguous.* Hence, also the following gaps are superpolynomial for unary cyclic languages:

- between the sizes of unambiguous NFAs and DFAs,
- between the sizes of unambiguous SVFAs and DFAs.

The superpolynomial function in all these gaps is $e^{\Omega(\sqrt[3]{N \cdot \ln^2 N})}$.
We strongly believe that for unary languages these gaps cannot be significantly improved.

Conclusion

We proved that the following gaps are superpolynomial, *even for unary cyclic languages:*

- between the total size of two NFAs accepting one language and its complement and the size of the corresponding DFA,
- between the sizes of SVFAs and DFAs.

The witness NFAs A^+ and A^- we used are *unambiguous.*

Hence, also the following gaps are superpolynomial for unary cyclic languages:

- between the sizes of unambiguous NFAs and DFAs,
- between the sizes of unambiguous SVFAs and DFAs.

The superpolynomial function in all these gaps is $e^{\Omega(\sqrt[3]{N \cdot \ln^2 N})}$.

We strongly believe that for unary languages these gaps cannot be significantly improved.

Conclusion

We proved that the following gaps are superpolynomial, *even for unary cyclic languages:*

- between the total size of two NFAs accepting one language and its complement and the size of the corresponding DFA,
- between the sizes of SVFAs and DFAs.

The witness NFAs A^+ and A^- we used are *unambiguous.*
Hence, also the following gaps are superpolynomial for unary cyclic languages:

- between the sizes of unambiguous NFAs and DFAs,
- between the sizes of unambiguous SVFAs and DFAs.

The superpolynomial function in all these gaps is $e^{\Omega(\sqrt[3]{N \cdot \ln^2 N})}$.
We strongly believe that for unary languages these gaps cannot be significantly improved.

Conclusion

We proved that the following gaps are superpolynomial, *even for unary cyclic languages:*

- between the total size of two NFAs accepting one language and its complement and the size of the corresponding DFA,
- between the sizes of SVFAs and DFAs.

The witness NFAs A^+ and A^- we used are *unambiguous.*
Hence, also the following gaps are superpolynomial for unary cyclic languages:

- between the sizes of unambiguous NFAs and DFAs,
- between the sizes of unambiguous SVFAs and DFAs.

The superpolynomial function in all these gaps is $e^{\Omega(\sqrt[3]{N \cdot \ln^2 N})}$.
We strongly believe that for unary languages these gaps cannot be significantly improved.