Eliminating the nondeterminism from two-way finite automata

Giovanni Pighizzini

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano

Frankfurt – December 11, 2007

Eliminating the nondeterminism from two-way automata

Outline of the talk

- Finite state automata and their descriptional complexity
- Eliminating nondeterminism using two–way motion: the problem of Sakoda and Sipser
- Unary automata, unary languages and their properties

Eliminating the nondeterminism from two-way automata

Outline of the talk

- Finite state automata and their descriptional complexity
- Eliminating nondeterminism using two–way motion: the problem of Sakoda and Sipser
- Unary automata, unary languages and their properties
- From unary one-way nondeterministic automata to two-way deterministic automata

Eliminating the nondeterminism from two-way automata

Outline of the talk

- Finite state automata and their descriptional complexity
- Eliminating nondeterminism using two–way motion: the problem of Sakoda and Sipser
- Unary automata, unary languages and their properties
- From unary *one-way* nondeterministic automata to two-way deterministic automata (optimal quadratic simulation)
- From unary *two-way* nondeterministic automata to two-way deterministic automata

Eliminating the nondeterminism from two-way automata

Outline of the talk

- Finite state automata and their descriptional complexity
- Eliminating nondeterminism using two–way motion: the problem of Sakoda and Sipser
- Unary automata, unary languages and their properties
- From unary *one-way* nondeterministic automata to two-way deterministic automata (optimal quadratic simulation)
- From unary *two-way* nondeterministic automata to two-way deterministic automata (subexponential simulation)

Eliminating the nondeterminism from two-way automata

Outline of the talk

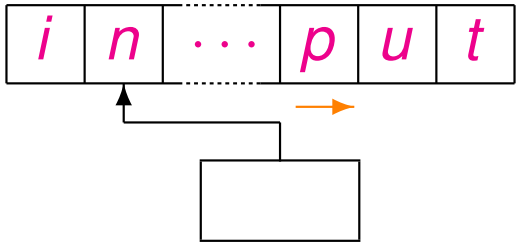
- Finite state automata and their descriptional complexity
- Eliminating nondeterminism using two–way motion: the problem of Sakoda and Sipser
- Unary automata, unary languages and their properties
- From unary *one-way* nondeterministic automata to two-way deterministic automata (optimal quadratic simulation)
- From unary *two-way* nondeterministic automata to two-way deterministic automata (subexponential simulation)

Eliminating the nondeterminism from two-way automata

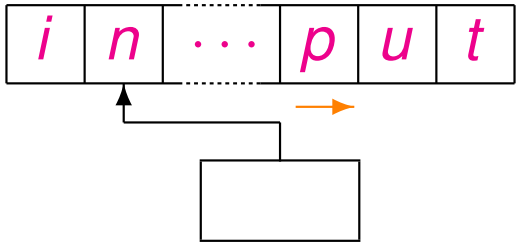
Outline of the talk

- Finite state automata and their descriptional complexity
- Eliminating nondeterminism using two–way motion: the problem of Sakoda and Sipser
- Unary automata, unary languages and their properties
- From unary *one-way* nondeterministic automata to two-way deterministic automata (optimal quadratic simulation)
- From unary *two-way* nondeterministic automata to two-way deterministic automata (subexponential simulation)

Finite state automata



Finite state automata

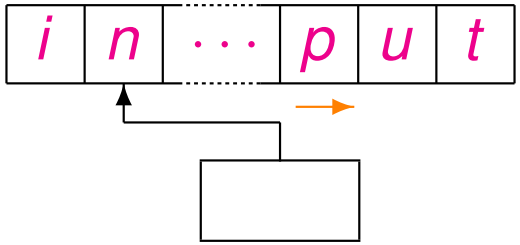


Base version:

One-way determistic finite automata (1dfa)

- one-way input tape
- deterministic

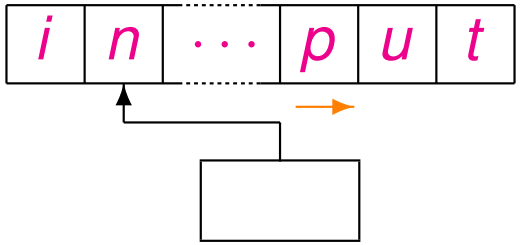
Finite state automata



Some possibile variants introducing:

- non determinism
- two-way input head motion
- alternation
- ...

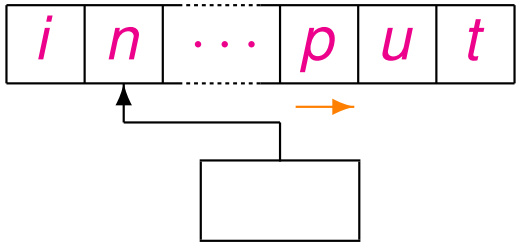
Finite state automata



Some possibile variants introducing:

- non determinism
- two-way input head motion
- alternation
- ...

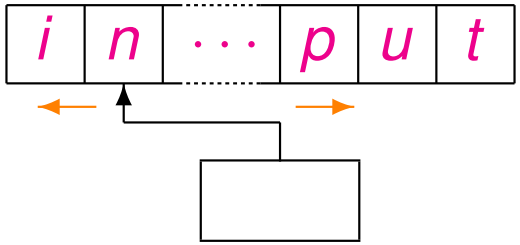
Finite state automata



Some possibile variants introducing:

- non determinism
  - one-way nondetermistic finite automata (1nfa)
- two-way input head motion
- alternation
- ...

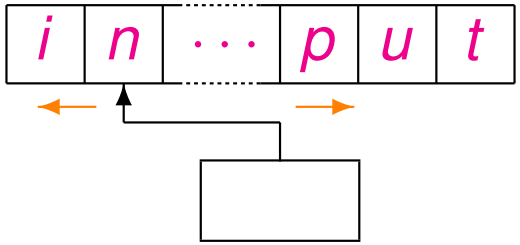
Finite state automata



Some possibile variants introducing:

- non determinism (1nfa)
- two-way input head motion
- alternation
- ...

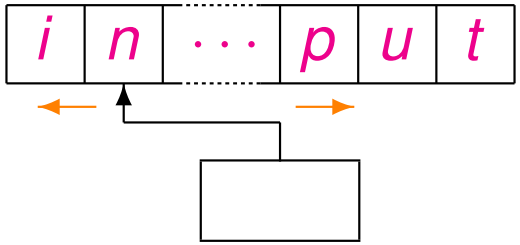
Finite state automata



Some possibile variants introducing:

- non determinism (1nfa)
- two-way input head motion
  - two-way determistic finite automata (2dfa)
  - two-way nondetermistic finite automata (2nfa)
- alternation
- ...

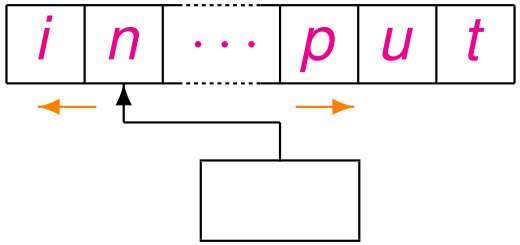
Finite state automata



Some possibile variants introducing:

- non determinism (1nfa)
- two-way input head motion
  - two-way determistic finite automata (2dfa)
  - two-way nondetermistic finite automata (2nfa)
- alternation
- ...

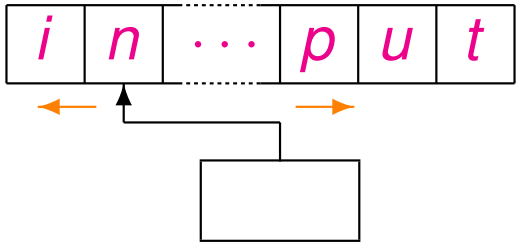
Finite state automata



Some possibile variants introducing:

- non determinism (1nfa)
- two-way input head motion (2dfa, 2nfa)
- alternation
- ...

Finite state automata



Some possibile variants introducing:

- non determinism (1nfa)
- two-way input head motion (2dfa, 2nfa)
- alternation
- ...

Finite state automata: different variants

1dfa, 1nfa, 2dfa, 2nfa, ...

What about the power of these models?

Finite state automata: different variants

1dfa, 1nfa, 2dfa, 2nfa, ...

What about the power of these models?

All these models have the same computational power, namely they characterize the class of *regular languages*,

Finite state automata: different variants

1dfa, 1nfa, 2dfa, 2nfa, ...

What about the power of these models?

All these models have the same computational power, namely they characterize the class of *regular languages*, however...

Finite state automata: different variants

1dfa, 1nfa, 2dfa, 2nfa, ...

What about the power of these models?

All these models have the same computational power, namely they characterize the class of *regular languages*, however...

...some of them are more succinct.

Finite state automata: different variants

1dfa, 1nfa, 2dfa, 2nfa, ...

What about the power of these models?

All these models have the same computational power, namely they characterize the class of *regular languages*, however...

...some of them are more succinct.

Example

- Each n-state 1nfa can be simulated by a 1dfa with 2^n states (subset construction) [Rabin and Scott '59], and:
- For each integer $n \geq 1$ there is a language which is accepted by a n-state 1nfa which requires 2^n state to be accepted by a 1dfa [Meyer and Fischer '71].

Finite state automata: different variants

1dfa, 1nfa, 2dfa, 2nfa, ...

What about the power of these models?

All these models have the same computational power, namely they characterize the class of *regular languages*, however...

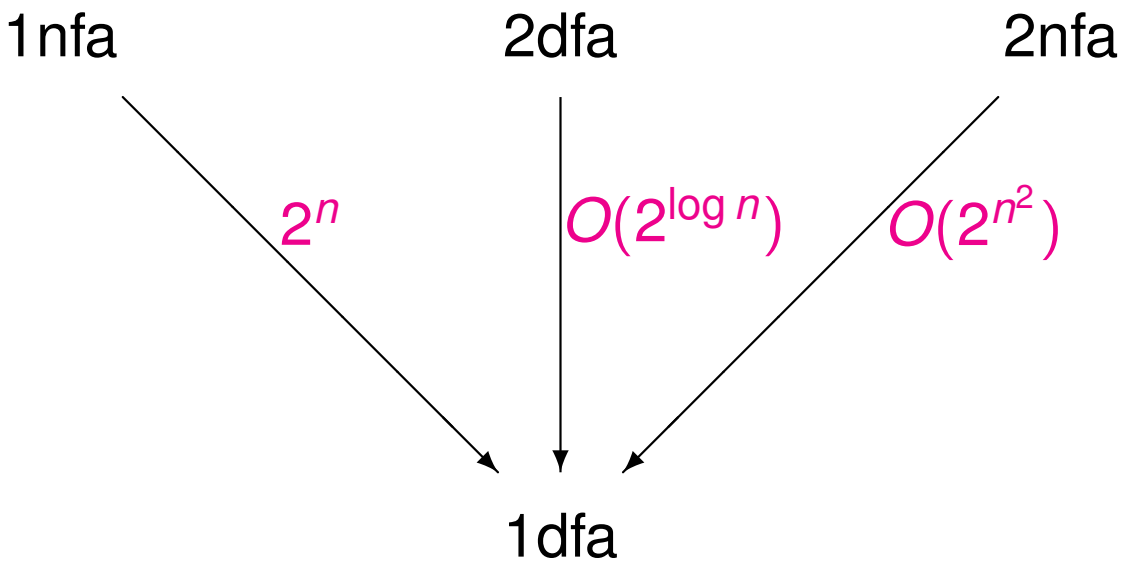
...some of them are more succinct.

Example

- Each n-state 1nfa can be simulated by a 1dfa with 2^n states (subset construction) [Rabin and Scott '59], and:
- For each integer $n \geq 1$ there is a language which is accepted by a n-state 1nfa which requires 2^n state to be accepted by a 1dfa [Meyer and Fischer '71].

Finite state automata

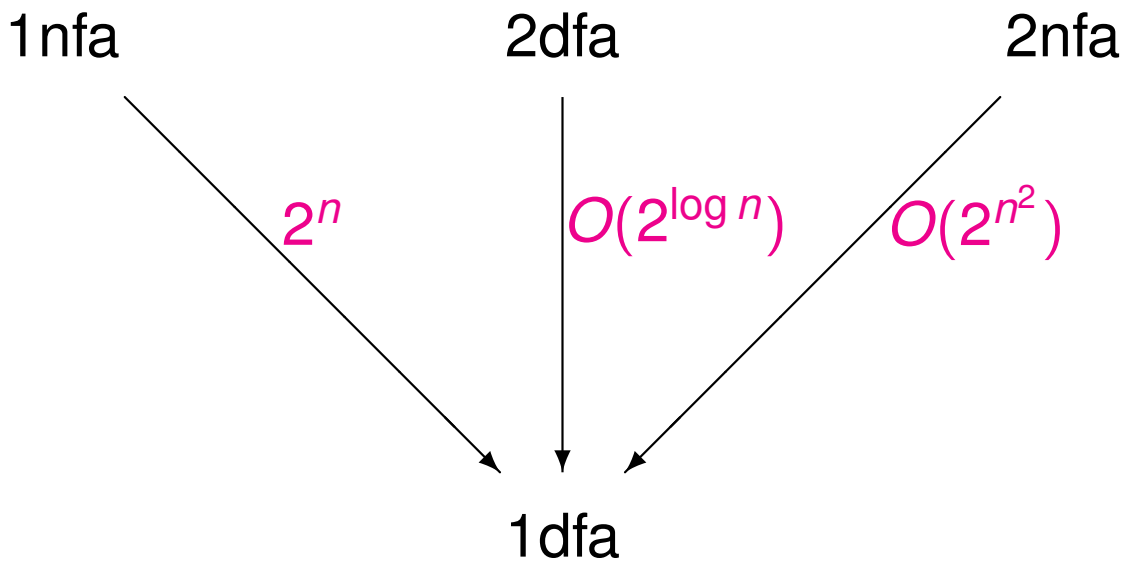
Costs of the optimal simulations by 1dfa:



[Rabin and Scott '59, Shepardson '59,Meyer and Fischer '71,...]

Finite state automata

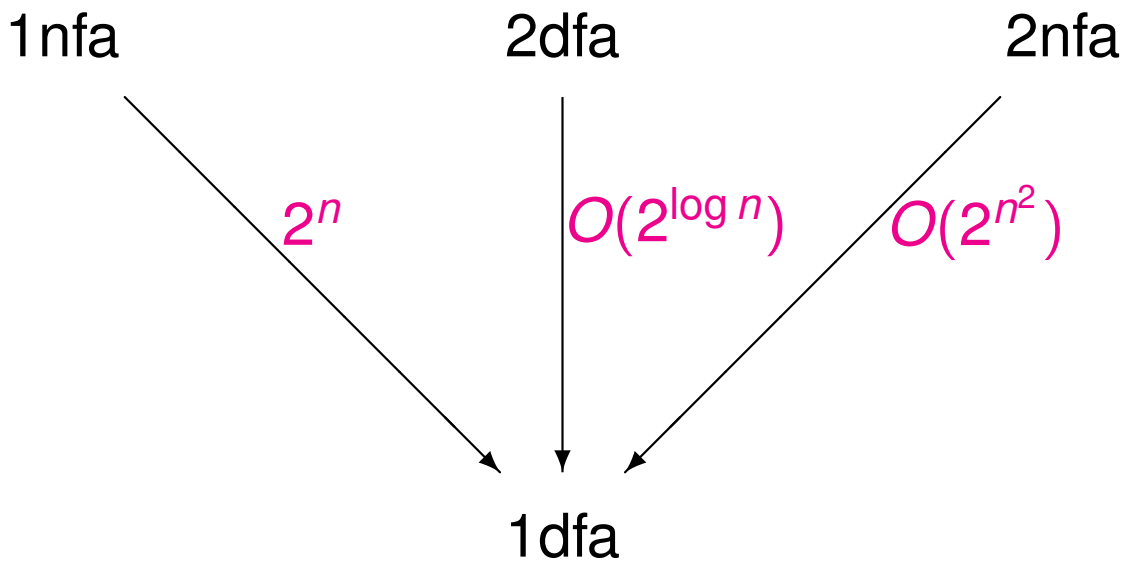
Costs of the optimal simulations by 1dfa:



How much two-way motion is useful in the elimination of the nondeterminism?

Finite state automata

Costs of the optimal simulations by 1dfa:



Problem ([Sakoda and Sipser 1978])

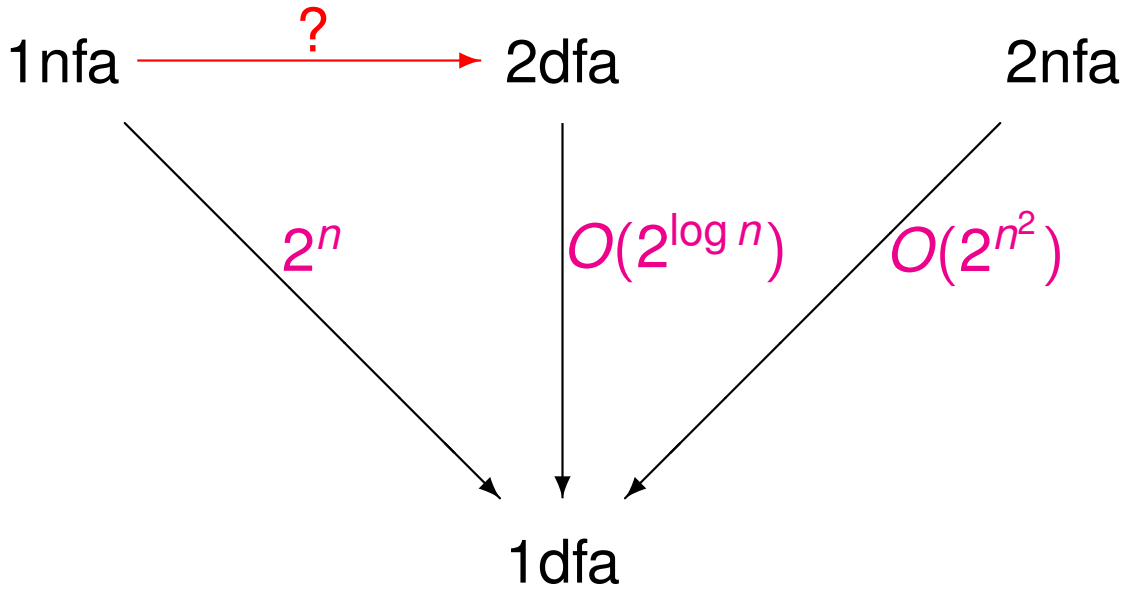
Find the costs, in terms of states, of the optimal simulations of

- *1nfa by 2dfa*
- *2nfa by 2dfa*

Conjecture: these costs are exponential

Finite state automata

Costs of the optimal simulations by 1dfa:



Problem ([Sakoda and Sipser 1978])

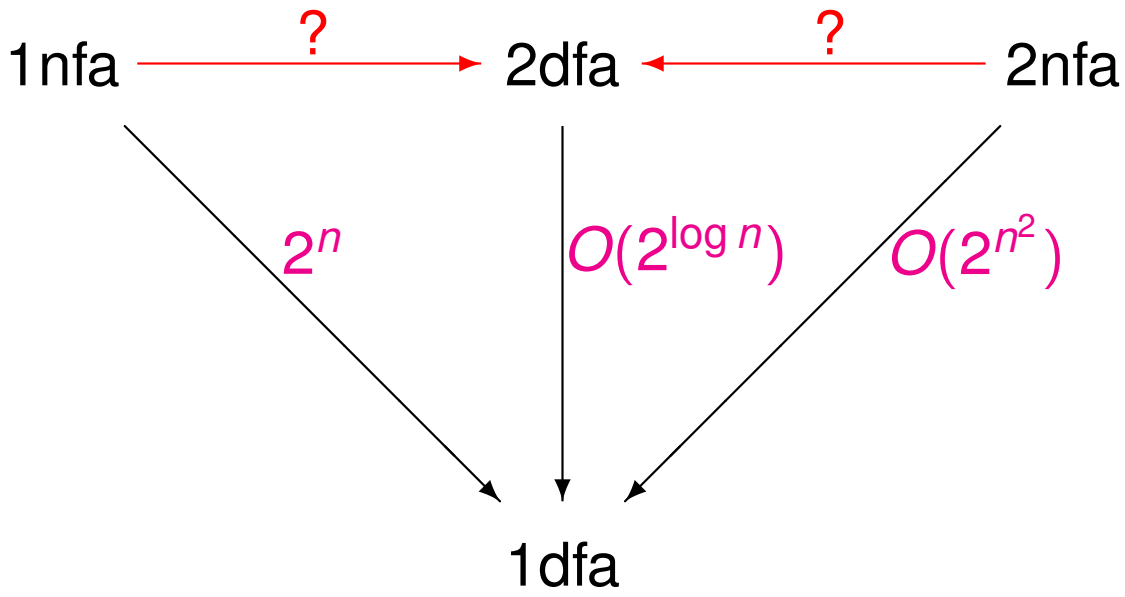
Find the costs, in terms of states, of the optimal simulations of

- *1nfa by 2dfa*
- *2nfa by 2dfa*

Conjecture: these costs are exponential

Finite state automata

Costs of the optimal simulations by 1dfa:



Problem ([Sakoda and Sipser 1978])

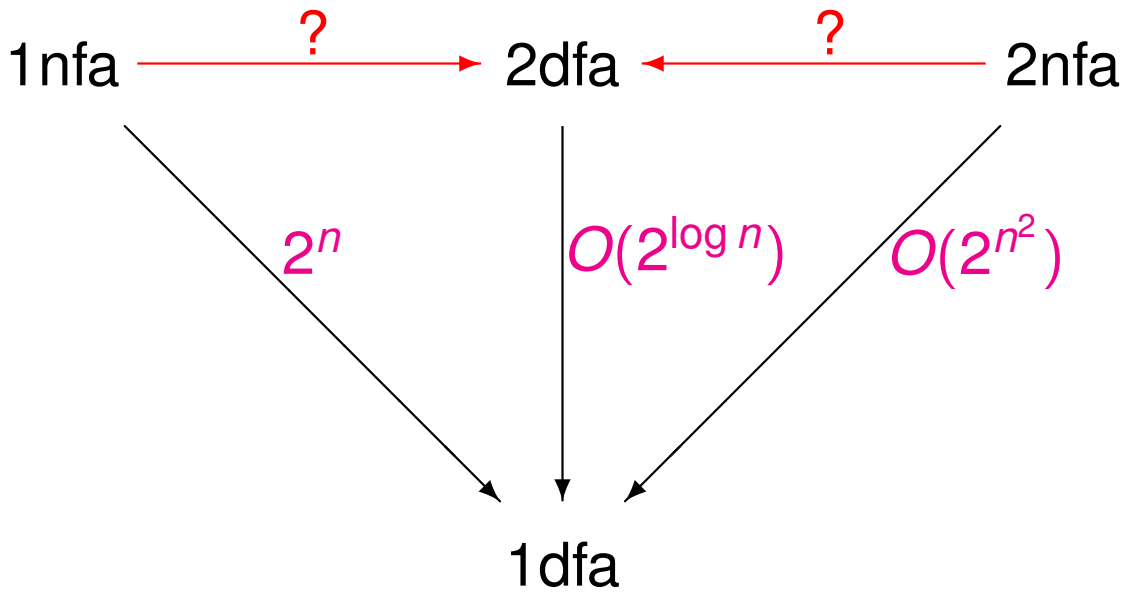
Find the costs, in terms of states, of the optimal simulations of

- *1nfa by 2dfa*
- *2nfa by 2dfa*

Conjecture: these costs are exponential

Finite state automata

Costs of the optimal simulations by 1dfa:



Problem ([Sakoda and Sipser 1978])

Find the costs, in terms of states, of the optimal simulations of

- *1nfa by 2dfa*
- *2nfa by 2dfa*

Conjecture: these costs are exponential

Sakoda and Sipser question: complete languages

Theorem (Complete languages for 1nfa vs 2dfa)

There exists a sequence of languages $< B_1, B_2, \ldots, B_n, \ldots >$ *s.t. for each integer* $n \geq 1$*:*

- B_n *is accepted by a 1nfa with n states, and*
- *among all languages accepted by n-state 1nfa,* B_n *requires the largest 2dfa.*

Remark: the second condition implies that
the simulation of 1nfa by 2dfa is polynomial iff each B_n is accepted by a 2dfa with a polynomial (in n) number of states.

In a similar way:

Theorem (Complete languages for 2nfa vs 2dfa)

There exists a sequence of languages $< C_1, C_2, \ldots, C_n, \ldots >$ *complete for the reduction of 2nfa to 2dfa.*

Sakoda and Sipser question: complete languages

Theorem (Complete languages for 1nfa vs 2dfa)

There exists a sequence of languages $< B_1, B_2, \ldots, B_n, \ldots >$ *s.t. for each integer* $n \geq 1$*:*

- B_n *is accepted by a 1nfa with n states, and*
- *among all languages accepted by n-state 1nfa,* B_n *requires the largest 2dfa.*

Remark: the second condition implies that
the simulation of 1nfa by 2dfa is polynomial iff each B_n is accepted by a 2dfa with a polynomial (in n) number of states.

In a similar way:

Theorem (Complete languages for 2nfa vs 2dfa)

There exists a sequence of languages $< C_1, C_2, \ldots, C_n, \ldots >$ *complete for the reduction of 2nfa to 2dfa.*

Sakoda and Sipser question: complete languages

Theorem (Complete languages for 1nfa vs 2dfa)

There exists a sequence of languages $< B_1, B_2, \ldots, B_n, \ldots >$ *s.t. for each integer* $n \geq 1$*:*

- B_n *is accepted by a 1nfa with n states, and*
- *among all languages accepted by n-state 1nfa,* B_n *requires the largest 2dfa.*

Remark: the second condition implies that
the simulation of 1nfa by 2dfa is polynomial iff each B_n is accepted by a 2dfa with a polynomial (in n) number of states.

In a similar way:

Theorem (Complete languages for 2nfa vs 2dfa)

There exists a sequence of languages $< C_1, C_2, \ldots, C_n, \ldots >$ *complete for the reduction of 2nfa to 2dfa.*

Sakoda and Sipser question: complete languages

Theorem (Complete languages for 1nfa vs 2dfa)

There exists a sequence of languages $< B_1, B_2, \ldots, B_n, \ldots >$ *s.t. for each integer* $n \geq 1$*:*

- B_n *is accepted by a 1nfa with n states, and*
- *among all languages accepted by n-state 1nfa,* B_n *requires the largest 2dfa.*

Remark: the second condition implies that
the simulation of 1nfa by 2dfa is polynomial iff each B_n is accepted by a 2dfa with a polynomial (in n) number of states.

In a similar way:

Theorem (Complete languages for 2nfa vs 2dfa)

There exists a sequence of languages $< C_1, C_2, \ldots, C_n, \ldots >$ *complete for the reduction of 2nfa to 2dfa.*

Sakoda and Sipser question: complete languages

Theorem (Complete languages for 1nfa vs 2dfa)

There exists a sequence of languages $< B_1, B_2, \ldots, B_n, \ldots >$ *s.t. for each integer* $n \geq 1$*:*

- B_n *is accepted by a 1nfa with n states, and*
- *among all languages accepted by n-state 1nfa,* B_n *requires the largest 2dfa.*

Remark: the second condition implies that
the simulation of 1nfa by 2dfa is polynomial iff each B_n is accepted by a 2dfa with a polynomial (in n) number of states.

In a similar way:

Theorem (Complete languages for 2nfa vs 2dfa)

There exists a sequence of languages $< C_1, C_2, \ldots, C_n, \ldots >$ *complete for the reduction of 2nfa to 2dfa.*

Sakoda and Sipser question: lower bounds

Polynomial lower bounds have been proved for the cost $c(n)$ of simulation of 1nfa by 2dfa. In particular:

- $c(n) \in \Omega(\frac{n^2}{\log n})$ [Berman and Lingas 1977]
- $c(n) \in \Omega(n^2)$ [Chrobak 1986]

Exponential lower bounds have been proved, if the resulting machines are required to satisfy some special conditions. e.g.,

- *sweeping automata* [Sipser 1980]
- *oblivious automata* [Hromkovič and Schnitger 2003]

Sakoda and Sipser question: lower bounds

Polynomial lower bounds have been proved for the cost $c(n)$ of simulation of 1nfa by 2dfa. In particular:

- $c(n) \in \Omega(\frac{n^2}{\log n})$ [Berman and Lingas 1977]
- $c(n) \in \Omega(n^2)$ [Chrobak 1986]

Exponential lower bounds have been proved, if the resulting machines are required to satisfy some special conditions. e.g.,

- *sweeping automata* [Sipser 1980]
- *oblivious automata* [Hromkovič and Schnitger 2003]

Sakoda and Sipser question: lower bounds

Polynomial lower bounds have been proved for the cost $c(n)$ of simulation of 1nfa by 2dfa. In particular:

- $c(n) \in \Omega(\frac{n^2}{\log n})$ [Berman and Lingas 1977]
- $c(n) \in \Omega(n^2)$ [Chrobak 1986]

Exponential lower bounds have been proved, if the resulting machines are required to satisfy some special conditions. e.g.,

- *sweeping automata* [Sipser 1980]
- *oblivious automata* [Hromkovič and Schnitger 2003]

Sakoda and Sipser question: lower bounds

Polynomial lower bounds have been proved for the cost $c(n)$ of simulation of 1nfa by 2dfa. In particular:

- $c(n) \in \Omega(\frac{n^2}{\log n})$ [Berman and Lingas 1977]
- $c(n) \in \Omega(n^2)$ [Chrobak 1986]

Exponential lower bounds have been proved, if the resulting machines are required to satisfy some special conditions. e.g.,

- *sweeping automata* [Sipser 1980]
- *oblivious automata* [Hromkovič and Schnitger 2003]

Sakoda and Sipser question: lower bounds

Polynomial lower bounds have been proved for the cost $c(n)$ of simulation of 1nfa by 2dfa. In particular:

- $c(n) \in \Omega(\frac{n^2}{\log n})$ [Berman and Lingas 1977]
- $c(n) \in \Omega(n^2)$ [Chrobak 1986]

Exponential lower bounds have been proved, if the resulting machines are required to satisfy some special conditions. e.g.,

- *sweeping automata* [Sipser 1980]
- *oblivious automata* [Hromkovič and Schnitger 2003]

Sakoda and Sipser question: lower bounds

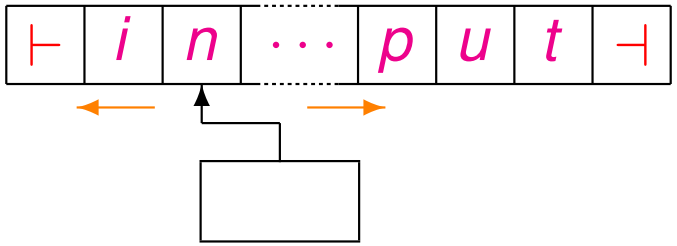
Polynomial lower bounds have been proved for the cost $c(n)$ of simulation of 1nfa by 2dfa. In particular:

- $c(n) \in \Omega(\frac{n^2}{\log n})$ [Berman and Lingas 1977]
- $c(n) \in \Omega(n^2)$ [Chrobak 1986]

Exponential lower bounds have been proved, if the resulting machines are required to satisfy some special conditions. e.g.,

- *sweeping automata* [Sipser 1980]
- *oblivious automata* [Hromkovič and Schnitger 2003]

Two-way automata



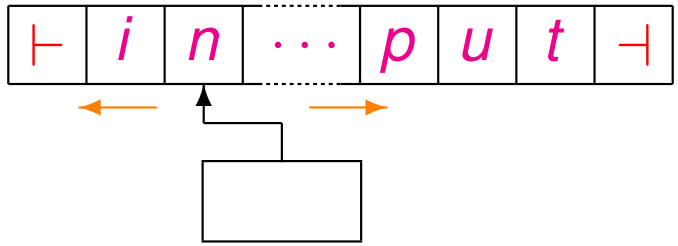
- Input string surrounded by two *endmarkers* $\vdash$ and $\dashv$
- Transition function

$$\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to 2^{Q \times \{-1,0,+1\}}$$

where $(p, d) \in \delta(q, a)$ means that the automaton

- in the state q, with the input head scanning the symbol $a \in \Sigma \cup \{\vdash, \dashv\}$
- can make a transition to the state p
- moving the input head in the direction specified by d: if $d = -1$ left; if $d = 0$ stationary; if $d = +1$ right; if $a = \vdash$ ($a = \dashv$, resp.) then $d \neq -1$ ($d \neq +1$, resp.)

Two-way automata



- Input string surrounded by two *endmarkers* $\vdash$ and $\dashv$
- Transition function

$$\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to 2^{Q \times \{-1,0,+1\}}$$

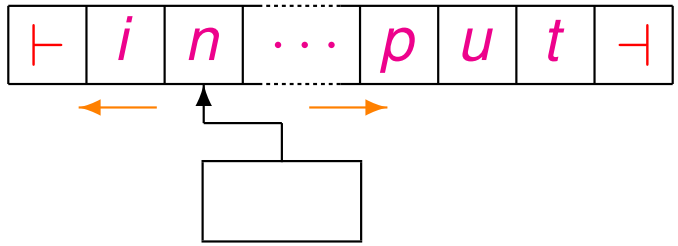
where $(p, d) \in \delta(q, a)$ means that the automaton

- in the state q, with the input head scanning the symbol $a \in \Sigma \cup \{\vdash, \dashv\}$
- can make a transition to the state p
- moving the input head in the direction specified by d:

  $d = -1$: left; $d = 0$: stationary; $d = +1$: right.

  If $a = \vdash$ ($a = \dashv$, resp.) then $d \neq -1$ ($d \neq +1$, resp.)

Two-way automata



- Input string surrounded by two *endmarkers* $\vdash$ and $\dashv$
- Transition function

$$\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to 2^{Q \times \{-1,0,+1\}}$$

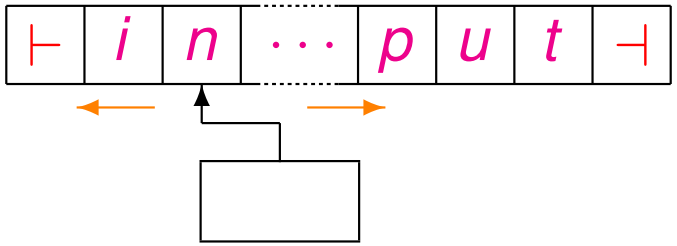
where $(p, d) \in \delta(q, a)$ means that the automaton

- in the state q, with the input head scanning the symbol $a \in \Sigma \cup \{\vdash, \dashv\}$
- can make a transition to the state p
- moving the input head in the direction specified by d:

$d = -1$: left; $d = 0$: stationary; $d = +1$: right.

If $a = \vdash$ ($a = \dashv$, resp.) then $d \neq -1$ ($d \neq +1$, resp.)

Two-way automata



- Input string surrounded by two *endmarkers* $\vdash$ and $\dashv$
- Transition function

$$\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to 2^{Q \times \{-1,0,+1\}}$$

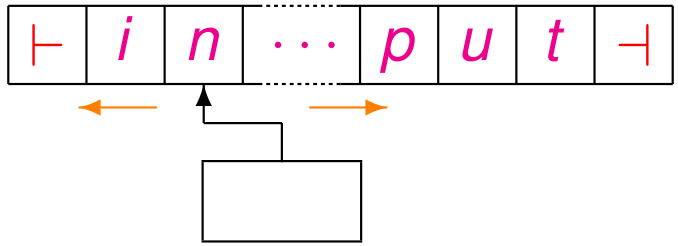
where $(p, d) \in \delta(q, a)$ means that the automaton

- in the state q, with the input head scanning the symbol $a \in \Sigma \cup \{\vdash, \dashv\}$
- can make a transition to the state p
- moving the input head in the direction specified by d:

$d = -1$: left; $d = 0$: stationary; $d = +1$: right.

If $a = \vdash$ ($a = \dashv$, resp.) then $d \neq -1$ ($d \neq +1$, resp.)

Two-way automata



- Input string surrounded by two *endmarkers* $\vdash$ and $\dashv$
- Transition function

$$\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to 2^{Q \times \{-1,0,+1\}}$$

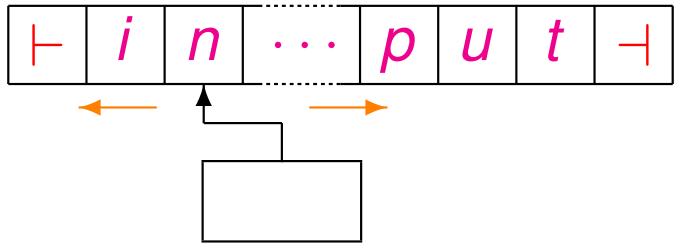
where $(p, d) \in \delta(q, a)$ means that the automaton

- in the state q, with the input head scanning the symbol $a \in \Sigma \cup \{\vdash, \dashv\}$
- can make a transition to the state p
- moving the input head in the direction specified by d:

$d = -1$: left; $d = 0$: stationary; $d = +1$: right.

If $a = \vdash$ ($a = \dashv$, resp.) then $d \neq -1$ ($d \neq +1$, resp.)

Two-way automata



- Input string surrounded by two *endmarkers* $\vdash$ and $\dashv$
- Transition function

$$\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to 2^{Q \times \{-1,0,+1\}}$$

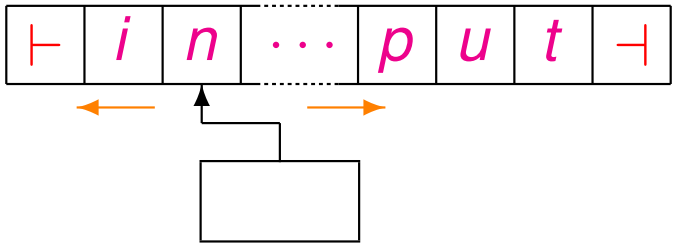
where $(p, d) \in \delta(q, a)$ means that the automaton

- in the state q, with the input head scanning the symbol $a \in \Sigma \cup \{\vdash, \dashv\}$
- can make a transition to the state p
- moving the input head in the direction specified by d:

  $d = -1$: left; $d = 0$: stationary; $d = +1$: right.

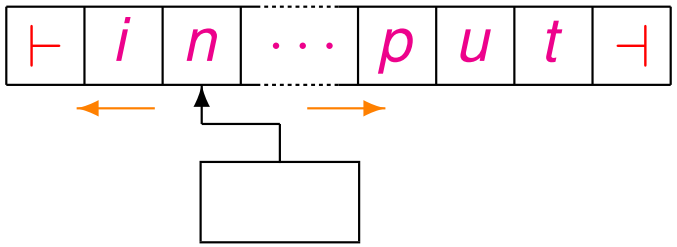
  If $a = \vdash$ ($a = \dashv$, resp.) then $d \neq -1$ ($d \neq +1$, resp.)

Two-way automata



- The automaton is *deterministic* if and only if $\#\delta(q, a) \leq 1$ for each $q \in Q$, $a \in \Sigma \cup \{\vdash, \dashv\}$

Sweeping automata



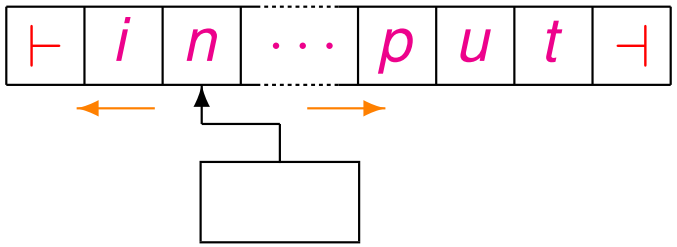
Definition

A two-way automaton A is said to be *sweeping* if and only if

- A is deterministic, and
- the input head of A can change direction only on the endmarkers

Note: the computation of a sweeping automaton is a sequence of left-to-right and right-to-left traversals of the input

Sweeping automata



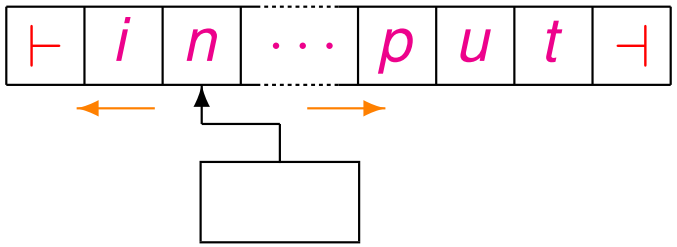
Definition

A two-way automaton A is said to be *sweeping* if and only if

- A is deterministic, and
- the input head of A can change direction only on the endmarkers

Note: the computation of a sweeping automaton is a sequence of left-to-right and right-to-left traversals of the input

Sweeping automata



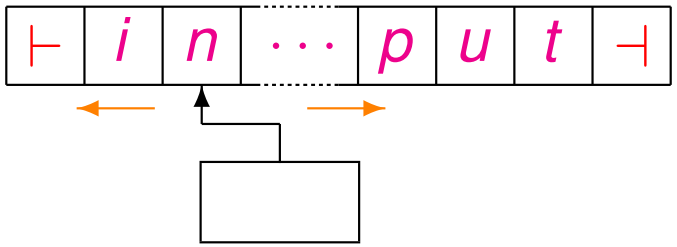
Definition

A two-way automaton A is said to be *sweeping* if and only if

- A is deterministic, and
- the input head of A can change direction only on the endmarkers

Note: the computation of a sweeping automaton is a sequence of left-to-right and right-to-left traversals of the input

Sweeping automata



Definition

A two-way automaton A is said to be *sweeping* if and only if

- A is deterministic, and
- the input head of A can change direction only on the endmarkers

Note: the computation of a sweeping automaton is a sequence of left-to-right and right-to-left traversals of the input

Sweeping automata

Theorem ([Sipser 1980])

There exists a family of languages $< B_1, B_2, \ldots, B_n, \ldots >$ s.t. for each integer $n \geq 1$:

- *B_n is accepted by a 1nfa with n states, and*
- *B_n cannot be accepted by any sweeping automaton with less than 2^n states.*

However, 2dfa can be exponentially more succinct than sweeping automata:

Theorem ([Berman 1981, Micali 1981])

There exists a family of languages $< A_1, A_2, \ldots, A_n, \ldots >$ s.t. for each integer $n \geq 1$:

- *A_n is accepted by a 2dfa with n states, and*
- *A_n cannot be accepted by any sweeping automaton with less than $2^n - 1$ states.*

Sweeping automata

Theorem ([Sipser 1980])

There exists a family of languages $< B_1, B_2, \ldots, B_n, \ldots >$ s.t. for each integer $n \geq 1$:

- *B_n is accepted by a 1nfa with n states, and*
- *B_n cannot be accepted by any sweeping automaton with less than 2^n states.*

However, 2dfa can be exponentially more succinct than sweeping automata:

Theorem ([Berman 1981, Micali 1981])

There exists a family of languages $< A_1, A_2, \ldots, A_n, \ldots >$ s.t. for each integer $n \geq 1$:

- *A_n is accepted by a 2dfa with n states, and*
- *A_n cannot be accepted by any sweeping automaton with less than $2^n - 1$ states.*

Sweeping automata

Theorem ([Sipser 1980])

There exists a family of languages $< B_1, B_2, \ldots, B_n, \ldots >$ s.t. for each integer $n \geq 1$:

- *B_n is accepted by a 1nfa with n states, and*
- *B_n cannot be accepted by any sweeping automaton with less than 2^n states.*

However, 2dfa can be exponentially more succinct than sweeping automata:

Theorem ([Berman 1981, Micali 1981])

There exists a family of languages $< A_1, A_2, \ldots, A_n, \ldots >$ s.t. for each integer $n \geq 1$:

- *A_n is accepted by a 2dfa with n states, and*
- *A_n cannot be accepted by any sweeping automaton with less than $2^n - 1$ states.*

Sweeping automata

Theorem ([Sipser 1980])

There exists a family of languages $< B_1, B_2, \ldots, B_n, \ldots >$ s.t. for each integer $n \geq 1$:

- *B_n is accepted by a 1nfa with n states, and*
- *B_n cannot be accepted by any sweeping automaton with less than 2^n states.*

However, 2dfa can be exponentially more succinct than sweeping automata:

Theorem ([Berman 1981, Micali 1981])

There exists a family of languages $< A_1, A_2, \ldots, A_n, \ldots >$ s.t. for each integer $n \geq 1$:

- *A_n is accepted by a 2dfa with n states, and*
- *A_n cannot be accepted by any sweeping automaton with less than $2^n - 1$ states.*

Sweeping automata

Theorem ([Sipser 1980])

There exists a family of languages $< B_1, B_2, \ldots, B_n, \ldots >$ *s.t. for each integer* $n \geq 1$*:*

- B_n *is accepted by a 1nfa with n states, and*
- B_n *cannot be accepted by any sweeping automaton with less than* 2^n *states.*

However, 2dfa can be exponentially more succinct than sweeping automata:

Theorem ([Berman 1981, Micali 1981])

There exists a family of languages $< A_1, A_2, \ldots, A_n, \ldots >$ *s.t. for each integer* $n \geq 1$*:*

- A_n *is accepted by a 2dfa with n states, and*
- A_n *cannot be accepted by any sweeping automaton with less than* $2^n - 1$ *states.*

Sweeping automata

Theorem ([Sipser 1980])

There exists a family of languages $< B_1, B_2, \ldots, B_n, \ldots >$ *s.t. for each integer* $n \geq 1$*:*

- B_n *is accepted by a 1nfa with n states, and*
- B_n *cannot be accepted by any sweeping automaton with less than* 2^n *states.*

However, 2dfa can be exponentially more succinct than sweeping automata:

Theorem ([Berman 1981, Micali 1981])

There exists a family of languages $< A_1, A_2, \ldots, A_n, \ldots >$ *s.t. for each integer* $n \geq 1$*:*

- A_n *is accepted by a 2dfa with n states, and*
- A_n *cannot be accepted by any sweeping automaton with less than* $2^n - 1$ *states.*

Sweeping automata

Theorem ([Sipser 1980])

There exists a family of languages $< B_1, B_2, \ldots, B_n, \ldots >$ s.t. for each integer $n \geq 1$:

- *B_n is accepted by a 1nfa with n states, and*
- *B_n cannot be accepted by any sweeping automaton with less than 2^n states.*

However, 2dfa can be exponentially more succinct than sweeping automata:

Theorem ([Berman 1981, Micali 1981])

There exists a family of languages $< A_1, A_2, \ldots, A_n, \ldots >$ s.t. for each integer $n \geq 1$:

- *A_n is accepted by a 2dfa with n states, and*
- *A_n cannot be accepted by any sweeping automaton with less than $2^n - 1$ states.*

Sakoda and Sipser question

Problem ([Sakoda and Sipser 1978])

Find the costs, in terms of states, of the optimal simulations of

- *1nfa by 2dfa*
- *2nfa by 2dfa*

- Very difficult in its general form
- Not very encouraging obtained results:

  Lower and upper bounds too far
  (Polynomial versus exponential)

- Hence:

  Try to solve a restriction of it!

- Consider the unary case $\#\Sigma = 1$

Sakoda and Sipser question

Problem ([Sakoda and Sipser 1978])

Find the costs, in terms of states, of the optimal simulations of

- *1nfa by 2dfa*
- *2nfa by 2dfa*

- Very difficult in its general form
- Not very encouraging obtained results:

  Lower and upper bounds too far
  (Polynomial versus exponential)

- Hence:

  Try to solve a restriction of it!

- Consider the unary case $\#\Sigma = 1$

Sakoda and Sipser question

Problem ([Sakoda and Sipser 1978])

Find the costs, in terms of states, of the optimal simulations of

- *1nfa by 2dfa*
- *2nfa by 2dfa*

- Very difficult in its general form
- Not very encouraging obtained results:

  Lower and upper bounds too far
  (Polynomial versus exponential)

- Hence:

  Try to solve a restriction of it!

- Consider the unary case $\#\Sigma = 1$

Sakoda and Sipser question

Problem ([Sakoda and Sipser 1978])

Find the costs, in terms of states, of the optimal simulations of

- *1nfa by 2dfa*
- *2nfa by 2dfa*

- Very difficult in its general form
- Not very encouraging obtained results:

  Lower and upper bounds too far
  (Polynomial versus exponential)

- Hence:

  Try to solve a restriction of it!

- Consider the unary case $\#\Sigma = 1$

Sakoda and Sipser question

Problem ([Sakoda and Sipser 1978])

Find the costs, in terms of states, of the optimal simulations of

- *1nfa by 2dfa*
- *2nfa by 2dfa*

- Very difficult in its general form
- Not very encouraging obtained results:

  Lower and upper bounds too far
  (Polynomial versus exponential)

- Hence:

  Try to solve a restriction of it!

- Consider the unary case $\#\Sigma = 1$

Unary automata: 1dfa

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

Theorem

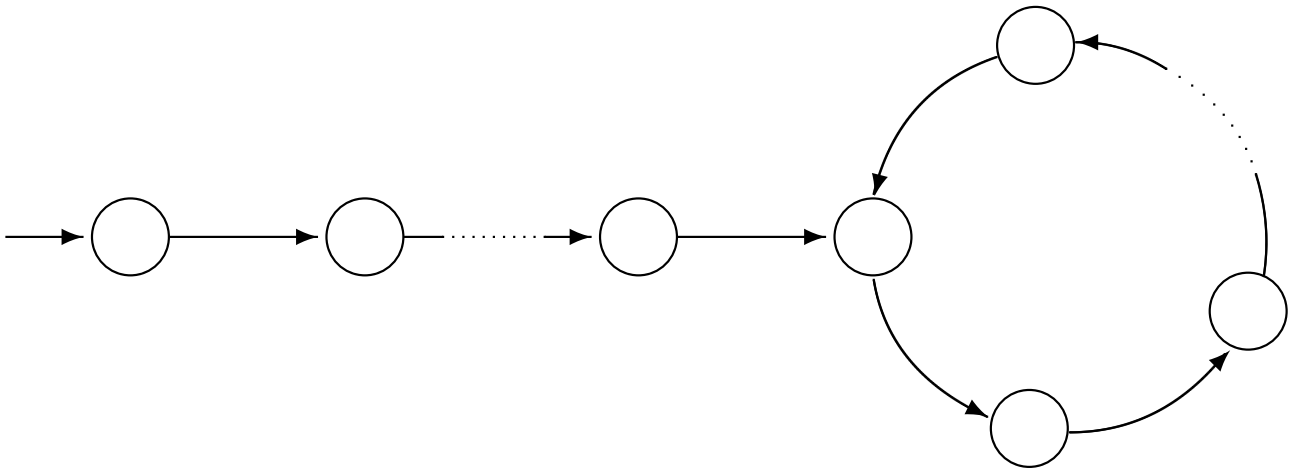
$L \subseteq \{a\}^*$ is regular iff $\exists \mu \geq 0, \lambda \geq 1$ s.t.

$$\forall n \geq \mu : a^n \in L \text{ iff } a^{n+\lambda} \in L.$$

Special case $\mu = 0$: the language is *periodic* or *cyclic*.

Unary automata: 1dfa

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



Theorem

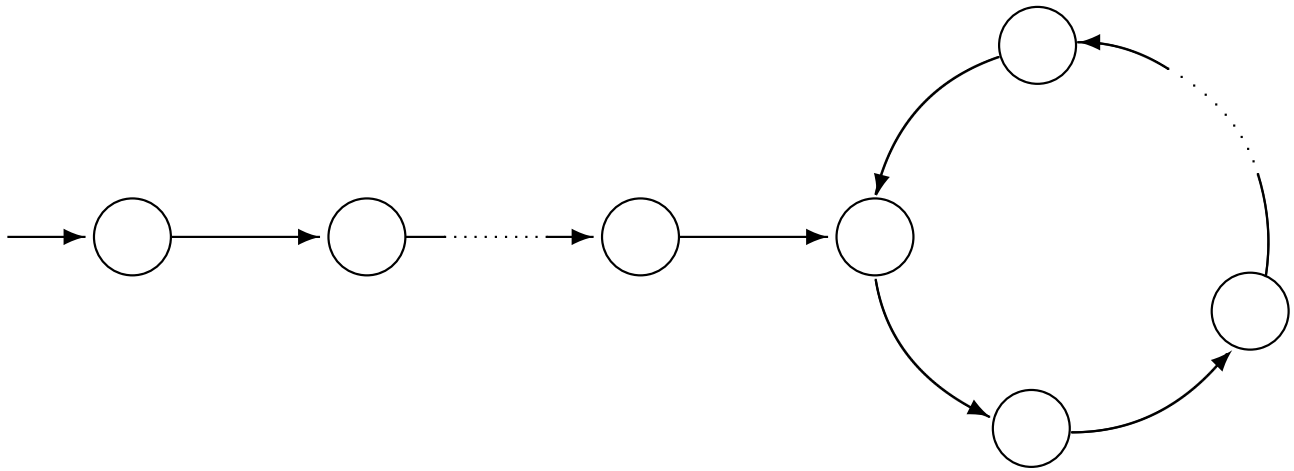
$L \subseteq \{a\}^*$ is regular iff $\exists \mu \geq 0, \lambda \geq 1$ s.t.

$$\forall n \geq \mu : a^n \in L \text{ iff } a^{n+\lambda} \in L.$$

Special case $\mu = 0$: the language is *periodic* or *cyclic*.

Unary automata: 1dfa

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



Theorem

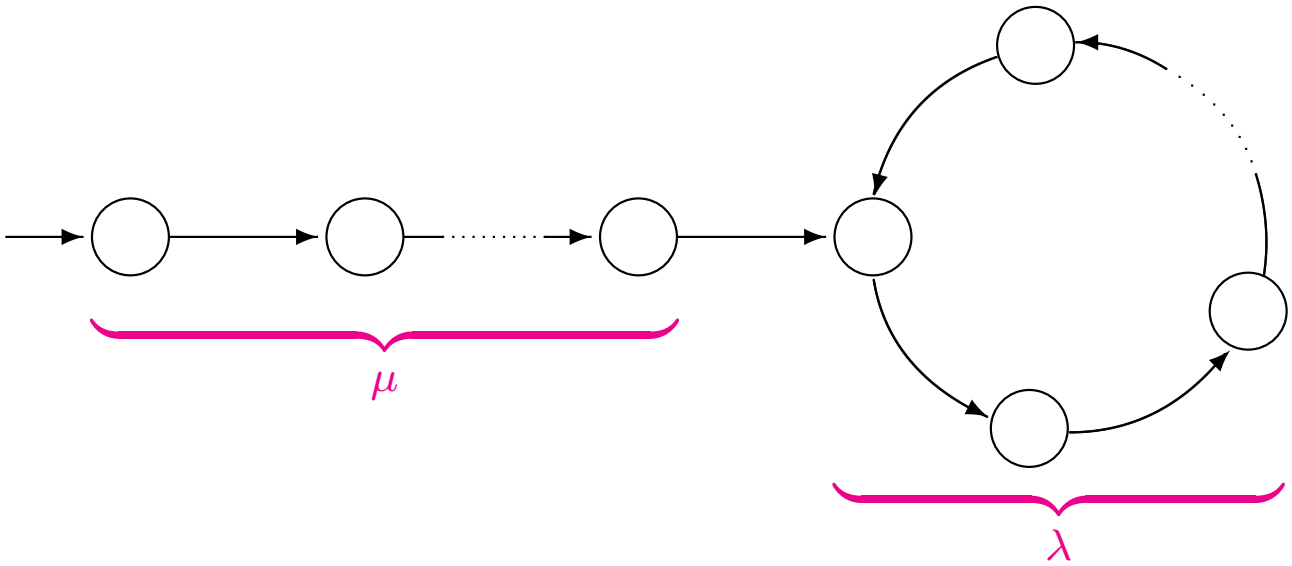
$L \subseteq \{a\}^*$ is regular iff $\exists \mu \geq 0, \lambda \geq 1$ *s.t.*

$$\forall n \geq \mu : a^n \in L \text{ iff } a^{n+\lambda} \in L.$$

Special case $\mu = 0$: the language is *periodic* or *cyclic*.

Unary automata: 1dfa

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



Theorem

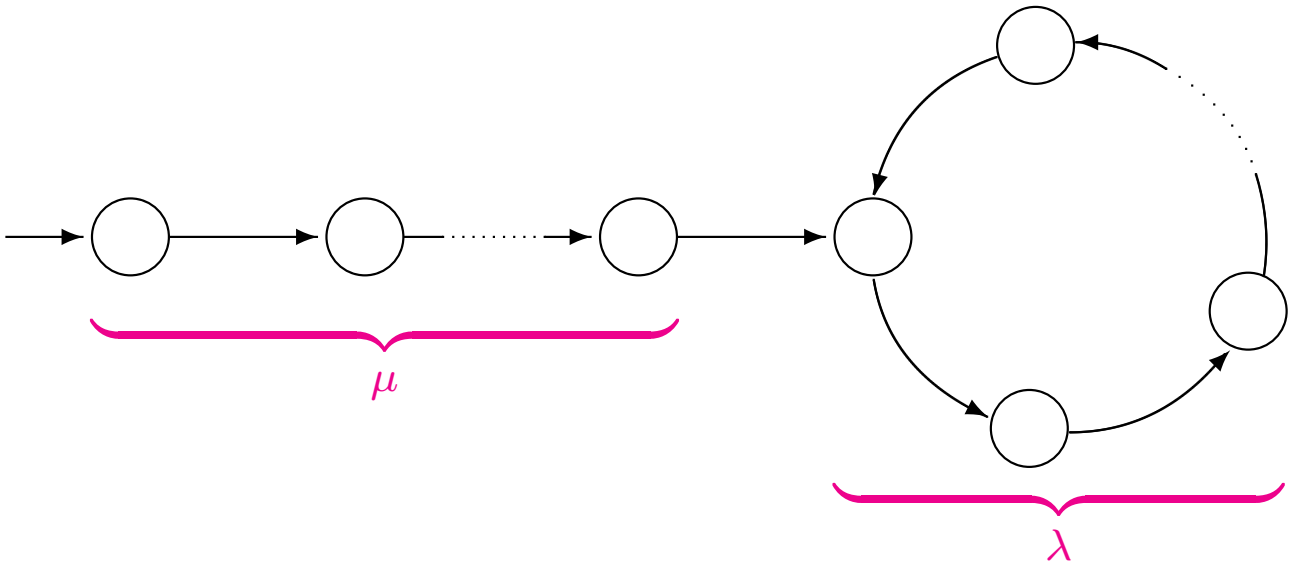
$L \subseteq \{a\}^*$ is regular iff $\exists \mu \geq 0, \lambda \geq 1$ s.t.

$$\forall n \geq \mu : a^n \in L \text{ iff } a^{n+\lambda} \in L.$$

Special case $\mu = 0$: the language is *periodic* or *cyclic*.

Unary automata: 1dfa

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



Theorem

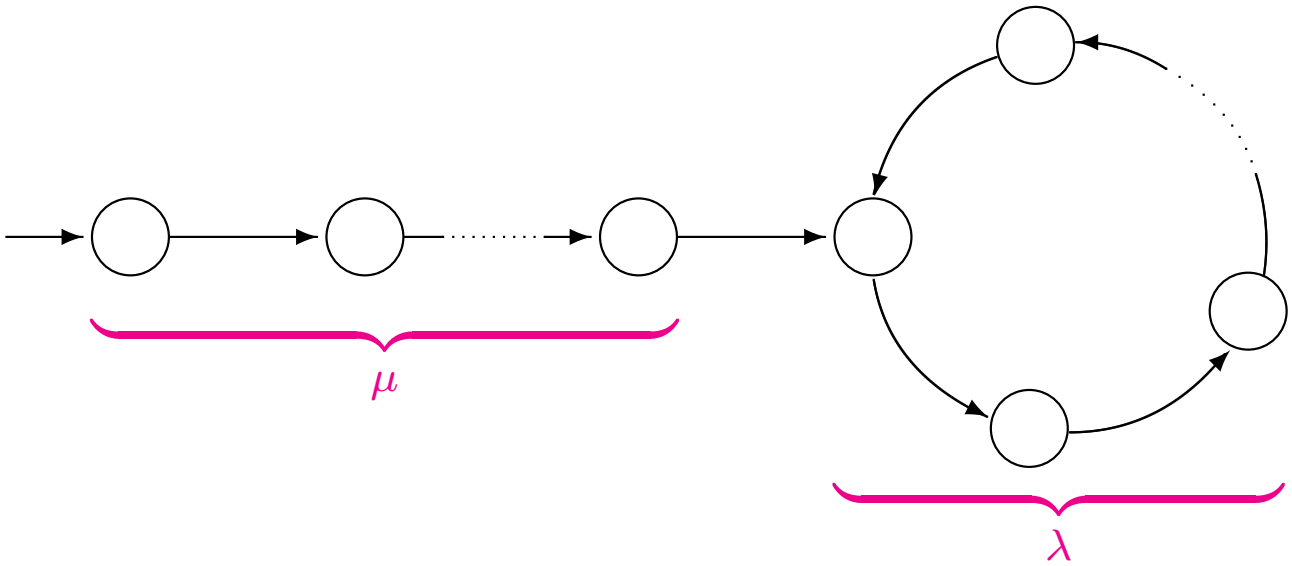
$L \subseteq \{a\}^*$ is regular iff $\exists \mu \geq 0, \lambda \geq 1$ s.t.

$$\forall n \geq \mu : a^n \in L \text{ iff } a^{n+\lambda} \in L.$$

Special case $\mu = 0$: the language is *periodic* or *cyclic*.

Unary automata: 1dfa

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



Theorem

$L \subseteq \{a\}^*$ is regular iff $\exists \mu \geq 0, \lambda \geq 1$ s.t.

$$\forall n \geq \mu : a^n \in L \text{ iff } a^{n+\lambda} \in L.$$

Special case $\mu = 0$: the language is *periodic* or *cyclic*.

Unary automata: 1nfa

The transition graph describing the automaton can have a whatever structure, however...

...we can restrict to 1nfa with the following form *(Chrobak normal form)*:

- an initial path
- a nondeterministic choice
- a set of cycles

Unary automata: 1nfa

The transition graph describing the automaton can have a whatever structure, however...

...we can restrict to 1nfa with the following form *(Chrobak normal form)*:

- an initial path
- a nondeterministic choice
- a set of cycles

Unary automata: 1nfa

The transition graph describing the automaton can have a whatever structure, however...

...we can restrict to 1nfa with the following form *(Chrobak normal form)*:

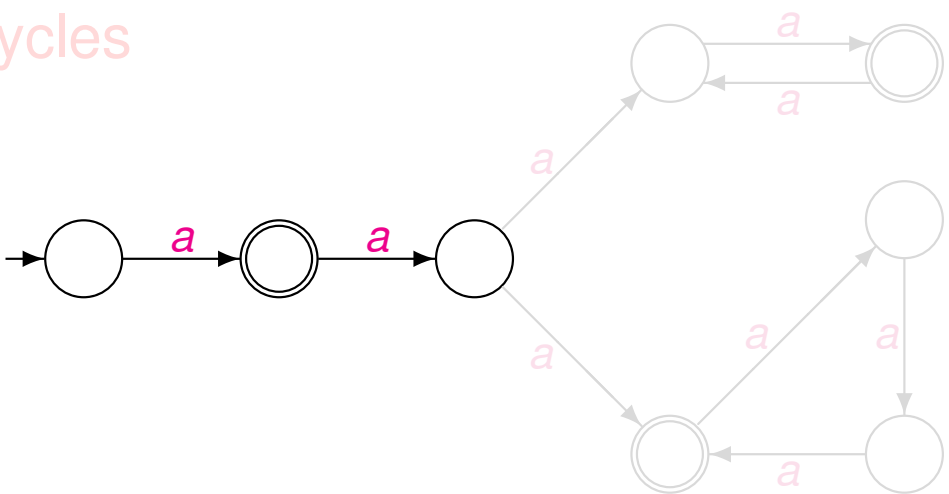
- an initial path
- a nondeterministic choice
- a set of cycles

Unary automata: 1nfa

The transition graph describing the automaton can have a whatever structure, however...

...we can restrict to 1nfa with the following form *(Chrobak normal form)*:

- an initial path
- a nondeterministic choice
- a set of cycles

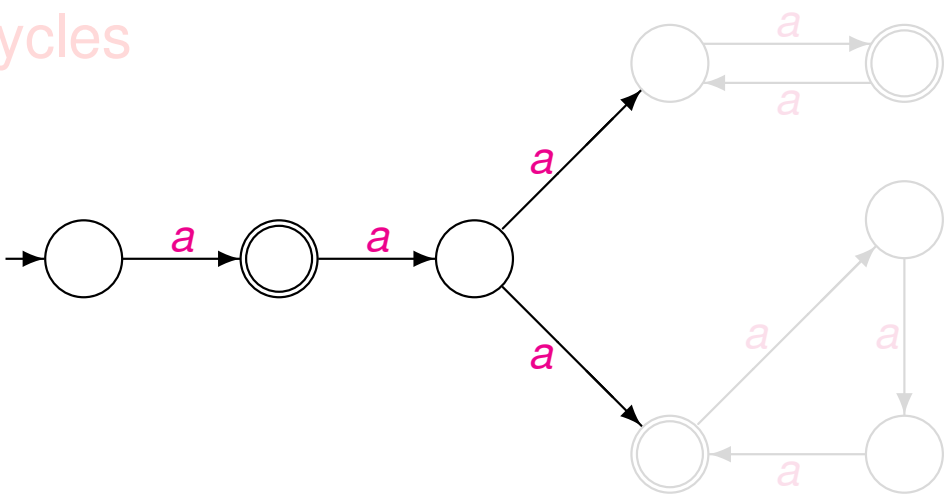


Unary automata: 1nfa

The transition graph describing the automaton can have a whatever structure, however...

...we can restrict to 1nfa with the following form *(Chrobak normal form)*:

- an initial path
- a nondeterministic choice
- a set of cycles

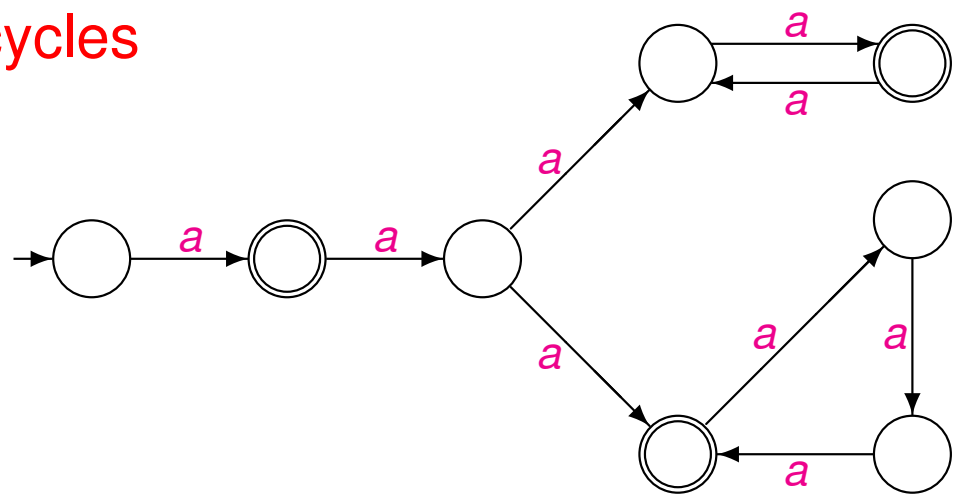


Unary automata: 1nfa

The transition graph describing the automaton can have a whatever structure, however...

...we can restrict to 1nfa with the following form *(Chrobak normal form)*:

- an initial path
- a nondeterministic choice
- a set of cycles



Unary automata: Chrobak normal form

Theorem

For any unary n-state 1nfa there exists an equivalent 1nfa in Chrobak normal form s.t.

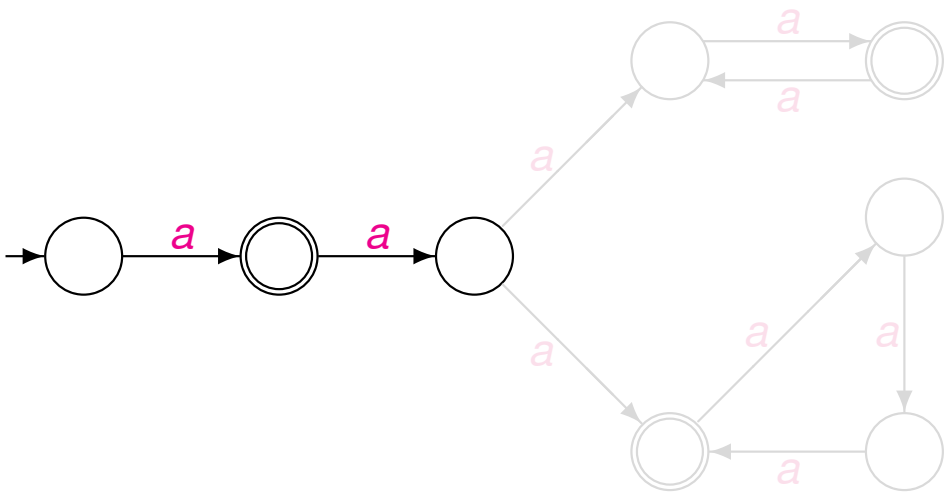
- *there are $O(n^2)$ states on the initial path*
- *the total number of states on the cycles is at most n*

Unary automata: Chrobak normal form

Theorem

For any unary n-state 1nfa there exists an equivalent 1nfa in Chrobak normal form s.t.

- *there are $O(n^2)$ states on the initial path*
- *the total number of states on the cycles is at most n*

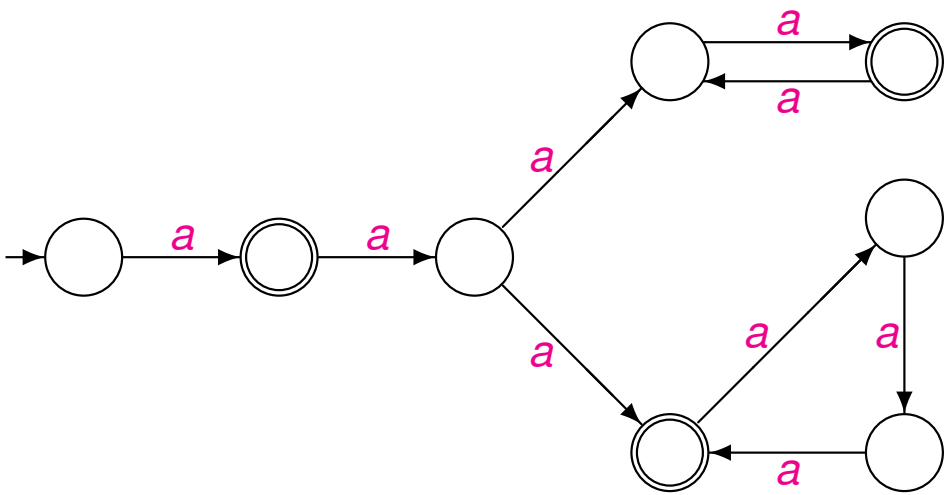


Unary automata: Chrobak normal form

Theorem

For any unary n-state 1nfa there exists an equivalent 1nfa in Chrobak normal form s.t.

- *there are $O(n^2)$ states on the initial path*
- *the total number of states on the cycles is at most n*



Unary nondeterministic automata

How to eliminate the nondeterminism from a unary 1nfa?

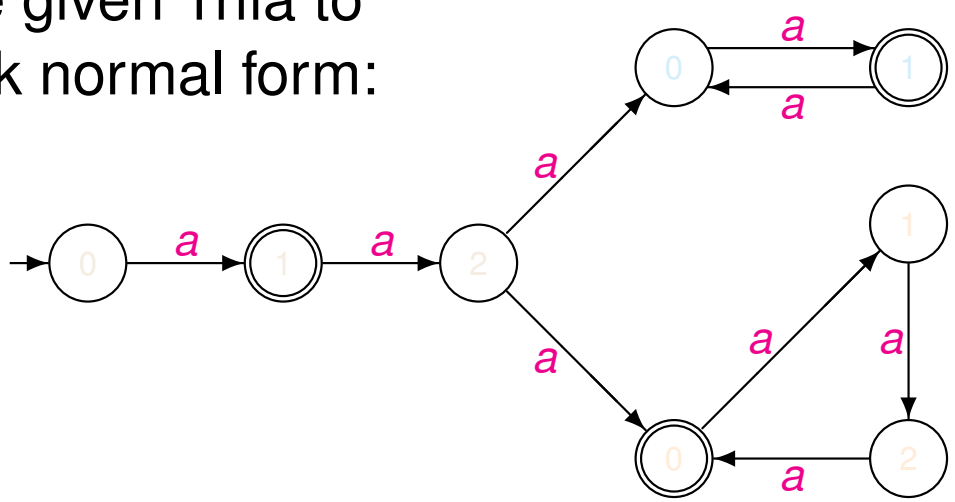
1. Convert the given 1nfa to the Chrobak normal form:

2. Copy the initial path
3. Replace the set of cycles with a unique cycle:

Unary nondeterministic automata

How to eliminate the nondeterminism from a unary 1nfa?

1. Convert the given 1nfa to the Chrobak normal form:

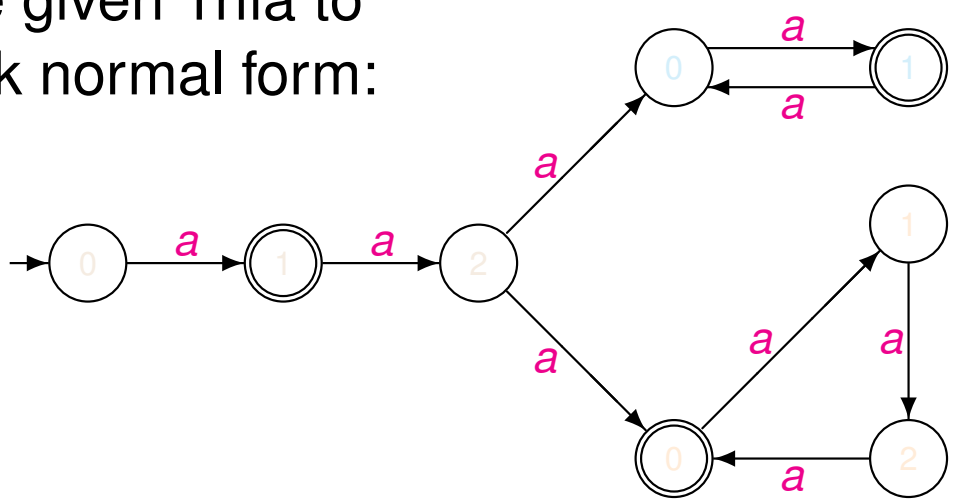


2. Copy the initial path
3. Replace the set of cycles with a unique cycle:

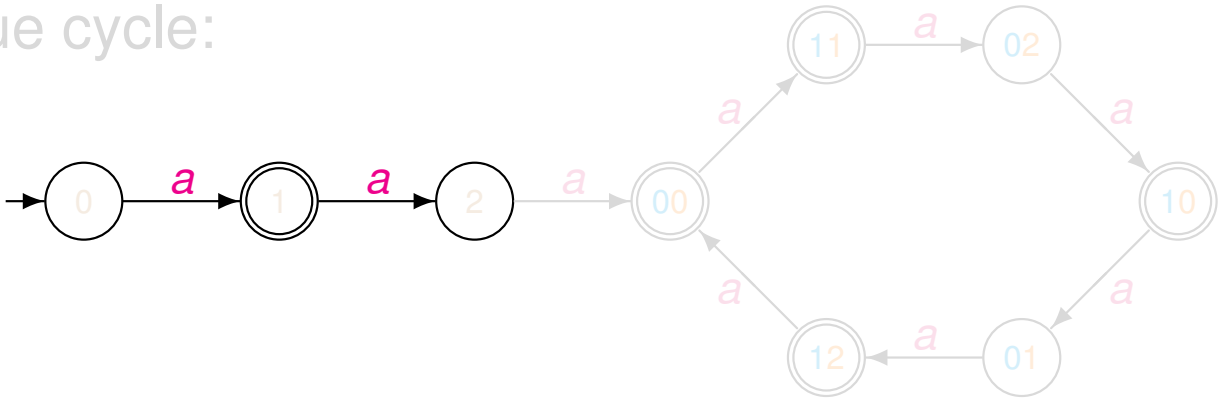
Unary nondeterministic automata

How to eliminate the nondeterminism from a unary 1nfa?

1. Convert the given 1nfa to the Chrobak normal form:



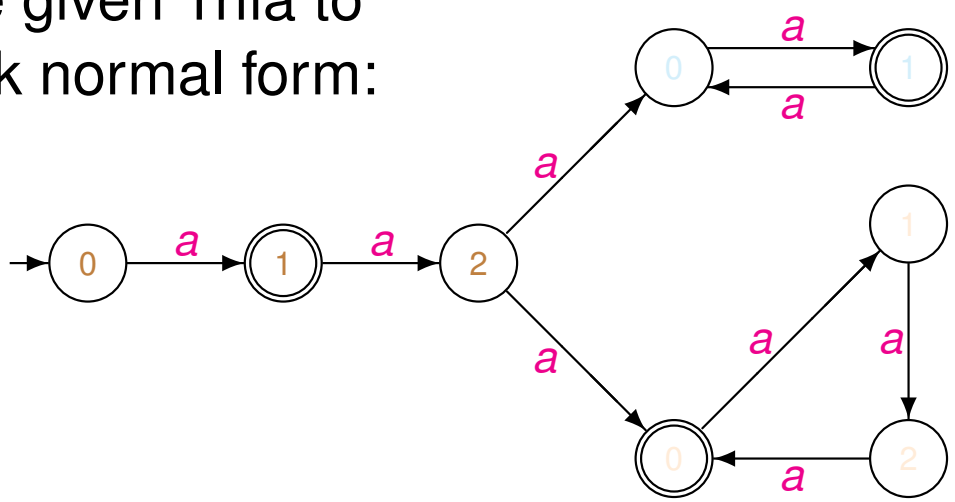
2. Copy the initial path
3. Replace the set of cycles with a unique cycle:



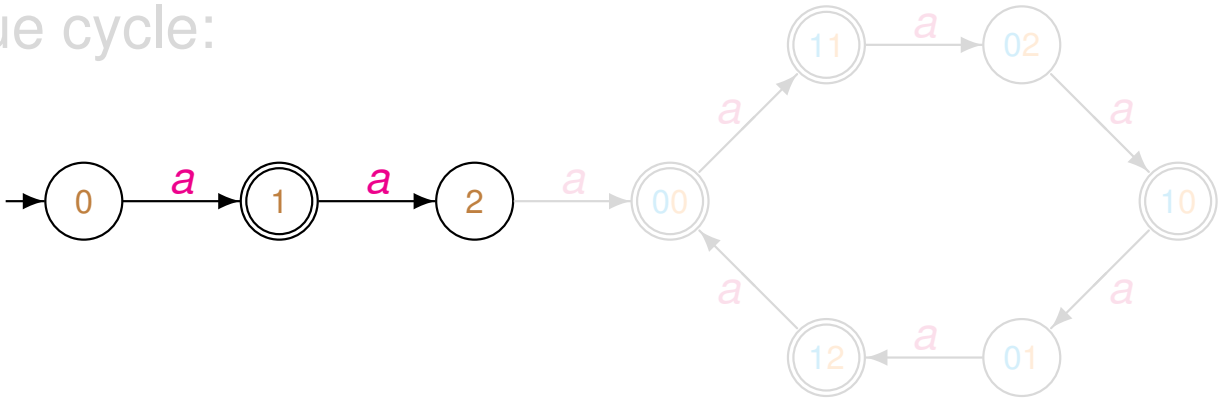
Unary nondeterministic automata

How to eliminate the nondeterminism from a unary 1nfa?

1. Convert the given 1nfa to the Chrobak normal form:



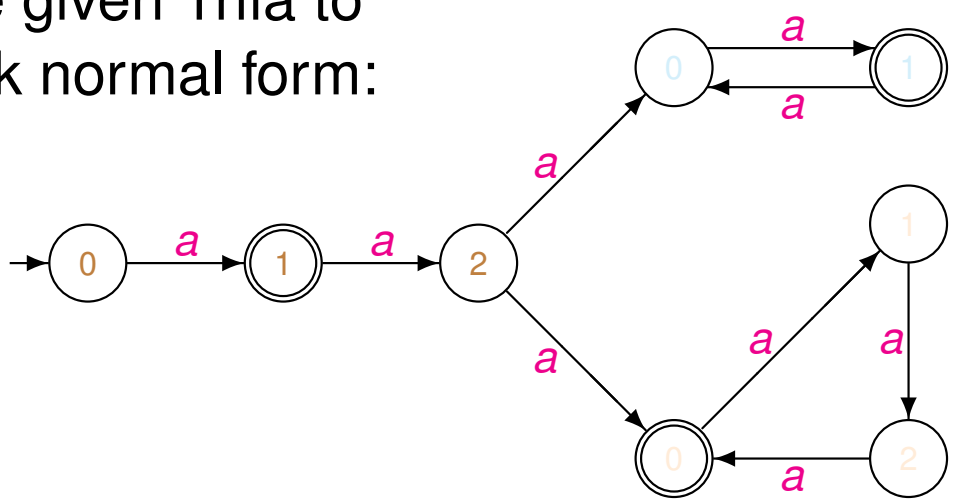
2. Copy the initial path
3. Replace the set of cycles with a unique cycle:



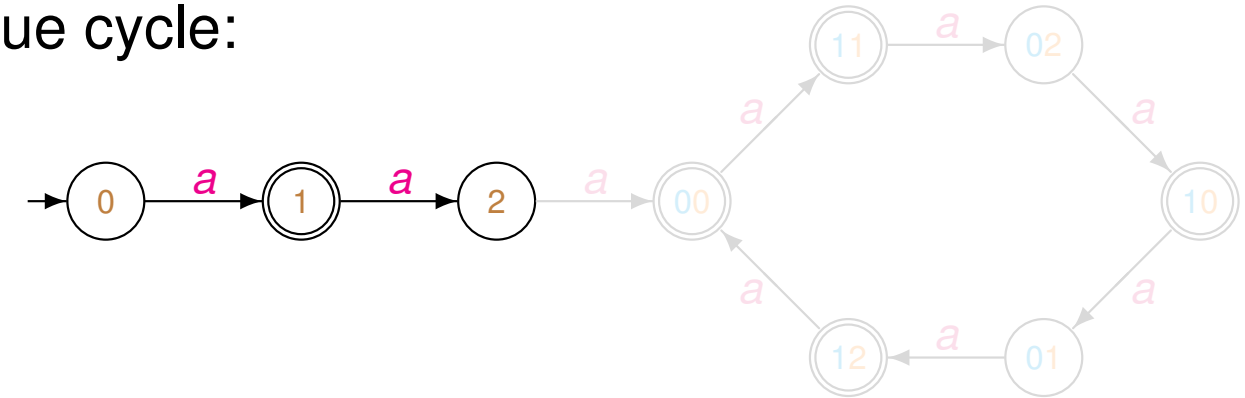
Unary nondeterministic automata

How to eliminate the nondeterminism from a unary 1nfa?

1. Convert the given 1nfa to the Chrobak normal form:



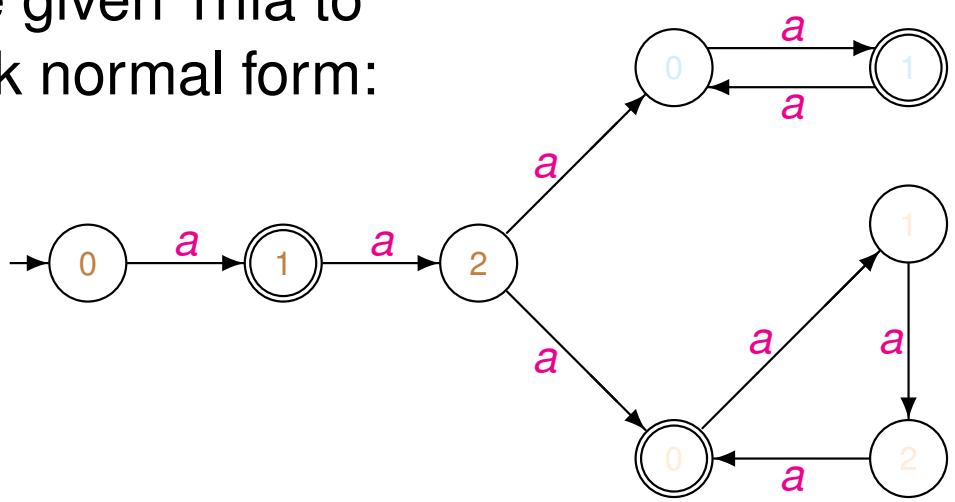
2. Copy the initial path
3. Replace the set of cycles with a unique cycle:



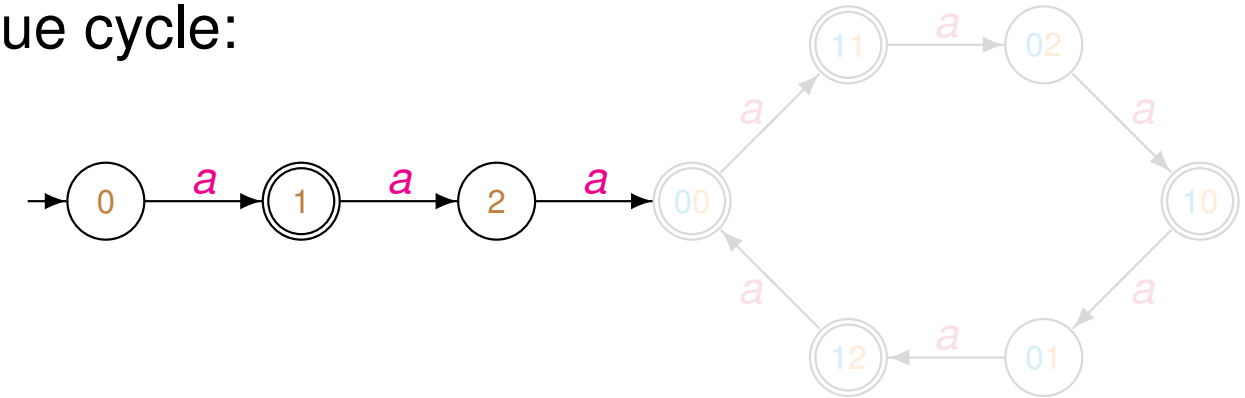
Unary nondeterministic automata

How to eliminate the nondeterminism from a unary 1nfa?

1. Convert the given 1nfa to the Chrobak normal form:



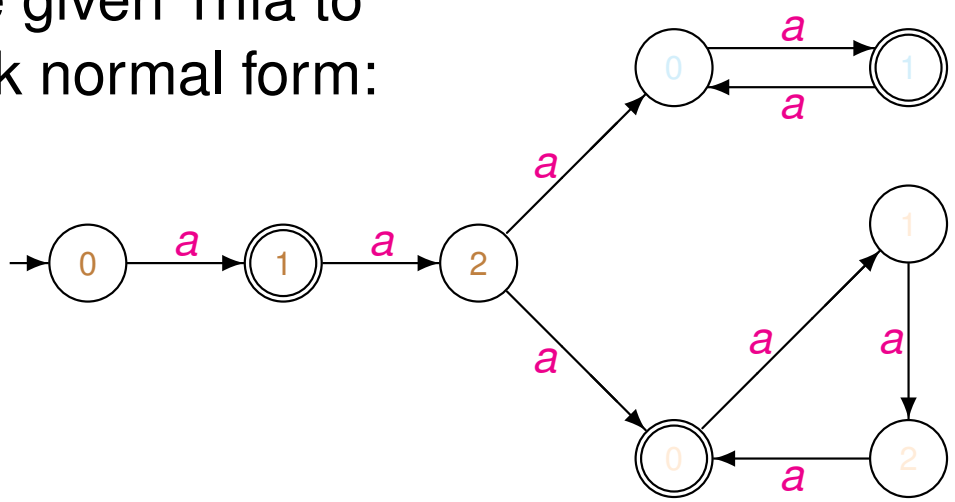
2. Copy the initial path
3. Replace the set of cycles with a unique cycle:



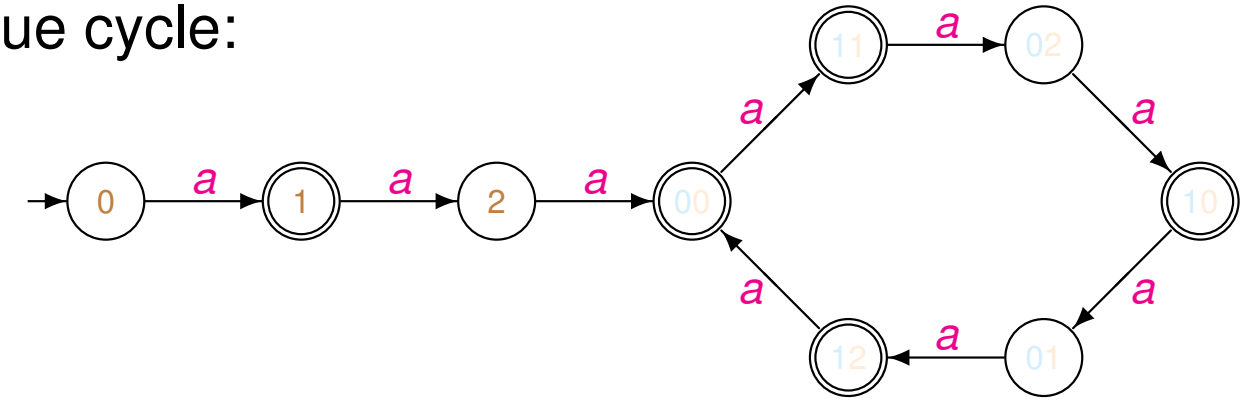
Unary nondeterministic automata

How to eliminate the nondeterminism from a unary 1nfa?

1. Convert the given 1nfa to the Chrobak normal form:



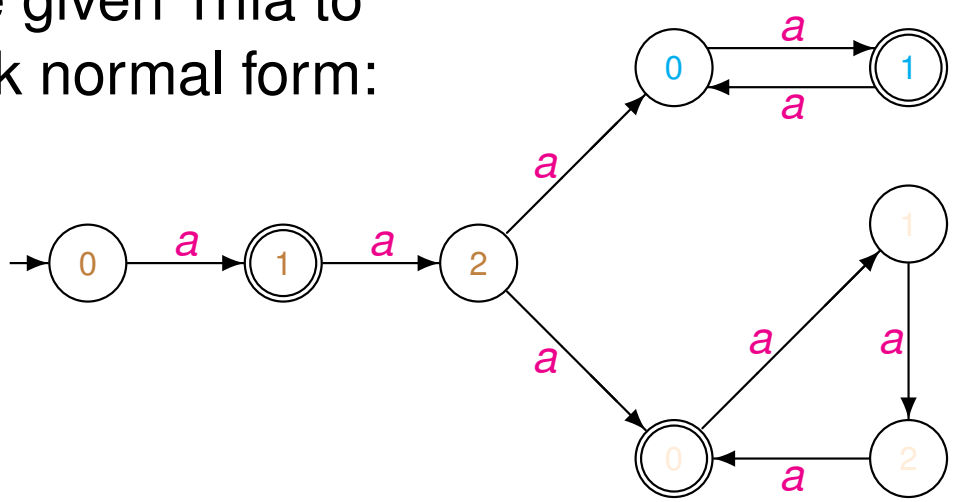
2. Copy the initial path
3. Replace the set of cycles with a unique cycle:



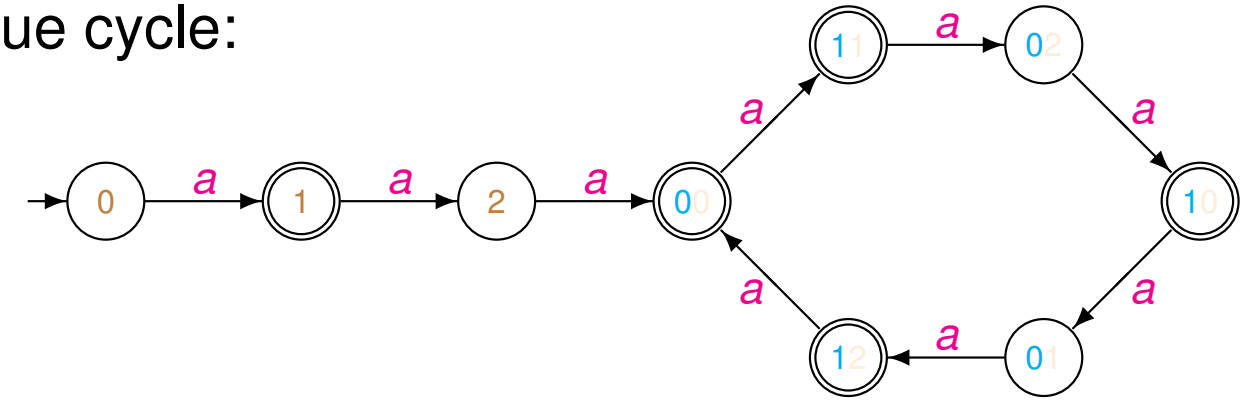
Unary nondeterministic automata

How to eliminate the nondeterminism from a unary 1nfa?

1. Convert the given 1nfa to the Chrobak normal form:



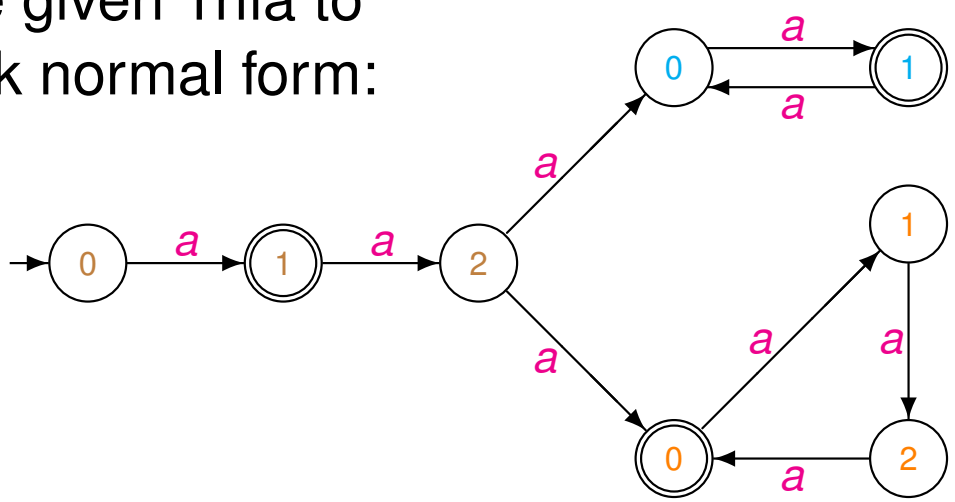
2. Copy the initial path
3. Replace the set of cycles with a unique cycle:



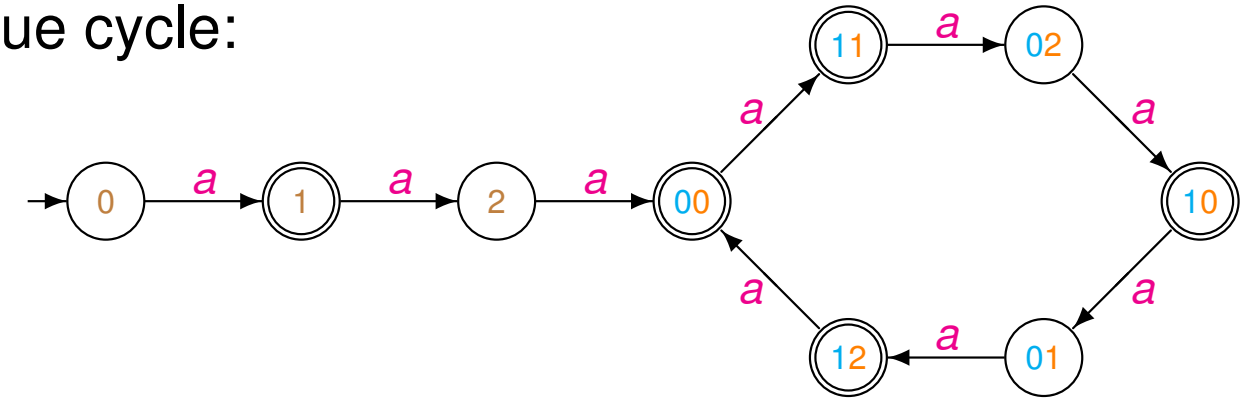
Unary nondeterministic automata

How to eliminate the nondeterminism from a unary 1nfa?

1. Convert the given 1nfa to the Chrobak normal form:



2. Copy the initial path
3. Replace the set of cycles with a unique cycle:



Simulation of unary 1nfa by 1dfa

Given an n-state 1nfa:

- Convert it in Chrobak normal form:
  - Initial path of $O(n^2)$ states
  - Cycles of lenghts $\lambda_1, \ldots, \lambda_k$, with $\lambda_1 + \ldots + \lambda_k \leq n$
- The cycles are replaced by a unique cycle of length $\mathrm{lcm}(\lambda_1, \ldots, \lambda_k)$

Hence:

- The number of states in the resulting cycle is bouned by $F(n) = \max\{\mathrm{lcm}(x_1, \ldots, x_k) \mid x_1 + \ldots + x_k = n\}$

and:

- $F(n) = e^{O(\sqrt{n \log n})}$ [Landau 1903]

Theorem ([Chrobak 1986])

Each unary n-state 1nfa can be simulated by a 1dfa with $e^{O(\sqrt{n \log n})}$ states.

Simulation of unary 1nfa by 1dfa

Given an n-state 1nfa:

- Convert it in Chrobak normal form:
  - Initial path of $O(n^2)$ states
  - Cycles of lenghts $\lambda_1, \ldots, \lambda_k$, with $\lambda_1 + \ldots + \lambda_k \leq n$
- The cycles are replaced by a unique cycle of length $\mathrm{lcm}(\lambda_1, \ldots, \lambda_k)$

Hence:

- The number of states in the resulting cycle is bouned by $F(n) = \max\{\mathrm{lcm}(x_1, \ldots, x_k) \mid x_1 + \ldots + x_k = n\}$

and:

- $F(n) = e^{O(\sqrt{n \log n})}$ [Landau 1903]

Theorem ([Chrobak 1986])

Each unary n-state 1nfa can be simulated by a 1dfa with $e^{O(\sqrt{n \log n})}$ states.

Simulation of unary 1nfa by 1dfa

Given an n-state 1nfa:

- Convert it in Chrobak normal form:
  - Initial path of $O(n^2)$ states
  - Cycles of lenghts $\lambda_1, \ldots, \lambda_k$, with $\lambda_1 + \ldots + \lambda_k \leq n$
- The cycles are replaced by a unique cycle of length $\mathrm{lcm}(\lambda_1, \ldots, \lambda_k)$

Hence:

- The number of states in the resulting cycle is bouned by $F(n) = \max\{\mathrm{lcm}(x_1, \ldots, x_k) \mid x_1 + \ldots + x_k = n\}$

and:

- $F(n) = e^{O(\sqrt{n \log n})}$ [Landau 1903]

Theorem ([Chrobak 1986])

Each unary n-state 1nfa can be simulated by a 1dfa with $e^{O(\sqrt{n \log n})}$ states.

Simulation of unary 1nfa by 1dfa

Given an n-state 1nfa:

- Convert it in Chrobak normal form:
  - Initial path of $O(n^2)$ states
  - Cycles of lenghts $\lambda_1, \ldots, \lambda_k$, with $\lambda_1 + \ldots + \lambda_k \leq n$
- The cycles are replaced by a unique cycle of length $\mathrm{lcm}(\lambda_1, \ldots, \lambda_k)$

Hence:

- The number of states in the resulting cycle is bouned by $F(n) = \max\{\mathrm{lcm}(x_1, \ldots, x_k) \mid x_1 + \ldots + x_k = n\}$

and:

- $F(n) = e^{O(\sqrt{n \log n})}$ [Landau 1903]

Theorem ([Chrobak 1986])

Each unary n-state 1nfa can be simulated by a 1dfa with $e^{O(\sqrt{n \log n})}$ states.

Simulation of unary 1nfa by 1dfa

Given an n-state 1nfa:

- Convert it in Chrobak normal form:
  - Initial path of $O(n^2)$ states
  - Cycles of lenghts $\lambda_1, \ldots, \lambda_k$, with $\lambda_1 + \ldots + \lambda_k \leq n$
- The cycles are replaced by a unique cycle of length $\mathrm{lcm}(\lambda_1, \ldots, \lambda_k)$

Hence:

- The number of states in the resulting cycle is bouned by $F(n) = \max\{\mathrm{lcm}(x_1, \ldots, x_k) \mid x_1 + \ldots + x_k = n\}$

and:

- $F(n) = e^{O(\sqrt{n \log n})}$ [Landau 1903]

Theorem ([Chrobak 1986])

Each unary n-state 1nfa can be simulated by a 1dfa with $e^{O(\sqrt{n \log n})}$ states.

Simulation of unary 1nfa by 1dfa

Given an n-state 1nfa:

- Convert it in Chrobak normal form:
  - Initial path of $O(n^2)$ states
  - Cycles of lenghts $\lambda_1, \ldots, \lambda_k$, with $\lambda_1 + \ldots + \lambda_k \leq n$
- The cycles are replaced by a unique cycle of length $\mathrm{lcm}(\lambda_1, \ldots, \lambda_k)$

Hence:

- The number of states in the resulting cycle is bouned by $F(n) = \max\{\mathrm{lcm}(x_1, \ldots, x_k) \mid x_1 + \ldots + x_k = n\}$

and:

- $F(n) = e^{O(\sqrt{n \log n})}$ [Landau 1903]

Theorem ([Chrobak 1986])

Each unary n-state 1nfa can be simulated by a 1dfa with $e^{O(\sqrt{n \log n})}$ states.

Simulation of unary 1nfa by 1dfa

Given an n-state 1nfa:

- Convert it in Chrobak normal form:
  - Initial path of $O(n^2)$ states
  - Cycles of lenghts $\lambda_1, \ldots, \lambda_k$, with $\lambda_1 + \ldots + \lambda_k \leq n$
- The cycles are replaced by a unique cycle of length $\mathrm{lcm}(\lambda_1, \ldots, \lambda_k)$

Hence:

- The number of states in the resulting cycle is bouned by $F(n) = \max\{\mathrm{lcm}(x_1, \ldots, x_k) \mid x_1 + \ldots + x_k = n\}$

and:

- $F(n) = e^{O(\sqrt{n \log n})}$ [Landau 1903]

Theorem ([Chrobak 1986])

Each unary n-state 1nfa can be simulated by a 1dfa with $e^{O(\sqrt{n \log n})}$ states.

Simulation of unary 1nfa by 1dfa

Given an n-state 1nfa:

- Convert it in Chrobak normal form:
  - Initial path of $O(n^2)$ states
  - Cycles of lenghts $\lambda_1, \ldots, \lambda_k$, with $\lambda_1 + \ldots + \lambda_k \leq n$
- The cycles are replaced by a unique cycle of length $\mathrm{lcm}(\lambda_1, \ldots, \lambda_k)$

Hence:

- The number of states in the resulting cycle is bouned by $F(n) = \max\{\mathrm{lcm}(x_1, \ldots, x_k) \mid x_1 + \ldots + x_k = n\}$

and:

- $F(n) = e^{O(\sqrt{n \log n})}$ [Landau 1903]

Theorem ([Chrobak 1986])

Each unary n-state 1nfa can be simulated by a 1dfa with $e^{O(\sqrt{n \log n})}$ states.

Simulation of unary 1nfa by 1dfa

Given an n-state 1nfa:

- Convert it in Chrobak normal form:
  - Initial path of $O(n^2)$ states
  - Cycles of lenghts $\lambda_1, \ldots, \lambda_k$, with $\lambda_1 + \ldots + \lambda_k \leq n$
- The cycles are replaced by a unique cycle of length $\mathrm{lcm}(\lambda_1, \ldots, \lambda_k)$

Hence:

- The number of states in the resulting cycle is bouned by $F(n) = \max\{\mathrm{lcm}(x_1, \ldots, x_k) \mid x_1 + \ldots + x_k = n\}$

and:

- $F(n) = e^{O(\sqrt{n \log n})}$ [Landau 1903]

Theorem ([Chrobak 1986])

Each unary n-state 1nfa can be simulated by a 1dfa with $e^{O(\sqrt{n \log n})}$ states.

Simulation of unary 1nfa by 1dfa

Given an n-state 1nfa:

- Convert it in Chrobak normal form:
  - Initial path of $O(n^2)$ states
  - Cycles of lenghts $\lambda_1, \ldots, \lambda_k$, with $\lambda_1 + \ldots + \lambda_k \leq n$
- The cycles are replaced by a unique cycle of length $\mathrm{lcm}(\lambda_1, \ldots, \lambda_k)$

Hence:

- The number of states in the resulting cycle is bouned by $F(n) = \max\{\mathrm{lcm}(x_1, \ldots, x_k) \mid x_1 + \ldots + x_k = n\}$

and:

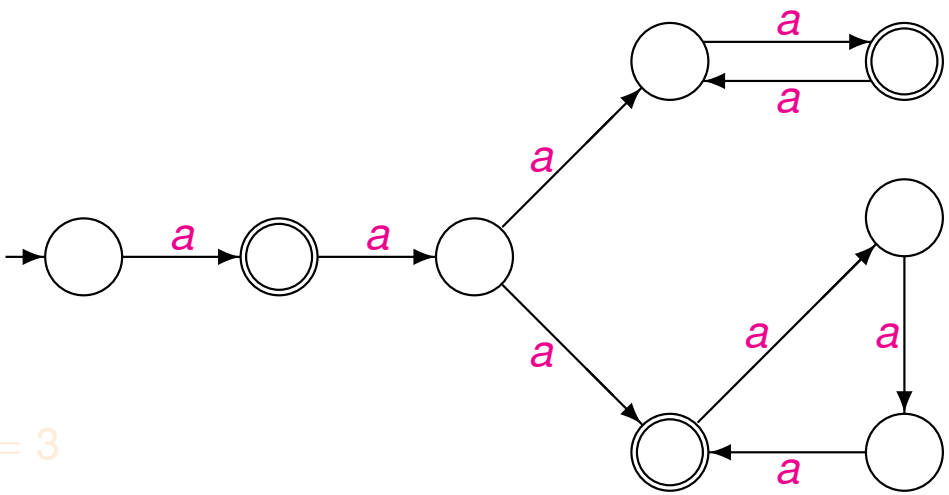
- $F(n) = e^{O(\sqrt{n \log n})}$ [Landau 1903]

Theorem ([Chrobak 1986])

Each unary n-state 1nfa can be simulated by a 1dfa with $e^{O(\sqrt{n \log n})}$ *states.*

Sakoda&Sipser question: 1nfa vs 2dfa

Example



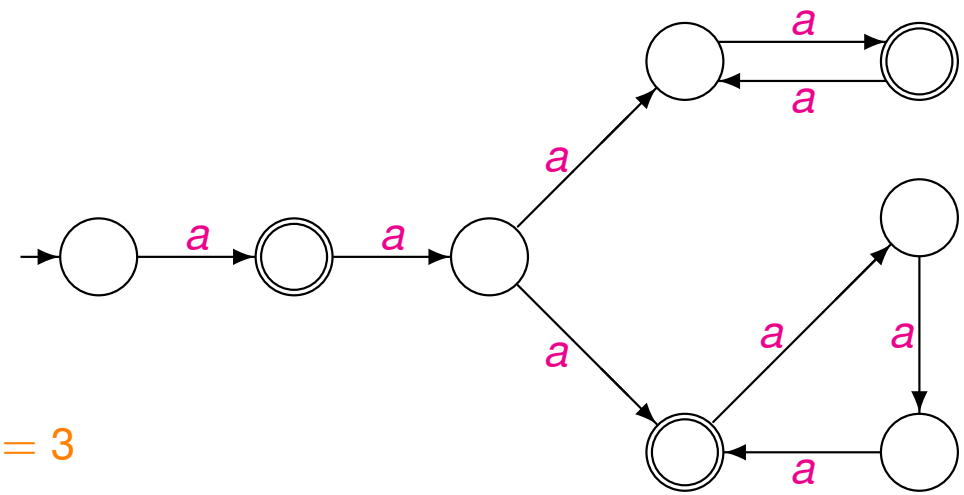
$\mu = 3, \lambda_1 = 2, \lambda_2 = 3$

On input a^m:

- First scan – simulation of the initial path:
  if $m < 3$ then stop and accept iff $m = 1$
- Second scan – simulation of the first loop:
  compute m MOD 2: if the result is 0 then stop and accept
- Third scan – simulation of the second loop:
  compute m MOD 3: if the result is 0 then stop and accept
- Finally:
  reject

Sakoda&Sipser question: 1nfa vs 2dfa

Example



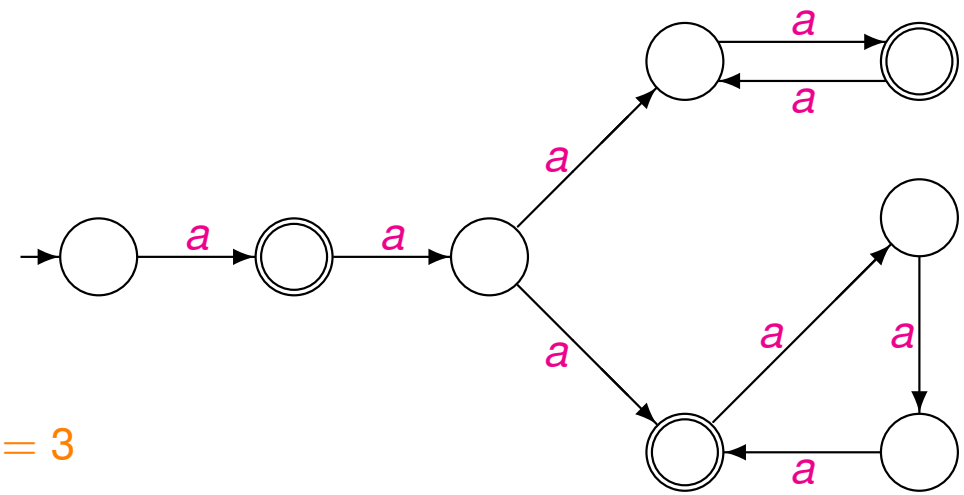
$\mu = 3, \lambda_1 = 2, \lambda_2 = 3$

On input a^m:

- First scan – simulation of the initial path:
  if $m < 3$ then stop and accept iff $m = 1$
- Second scan – simulation of the first loop:
  compute m MOD 2: if the result is 0 then stop and accept
- Third scan – simulation of the second loop:
  compute m MOD 3: if the result is 0 then stop and accept
- Finally:
  reject

Sakoda&Sipser question: 1nfa vs 2dfa

Example



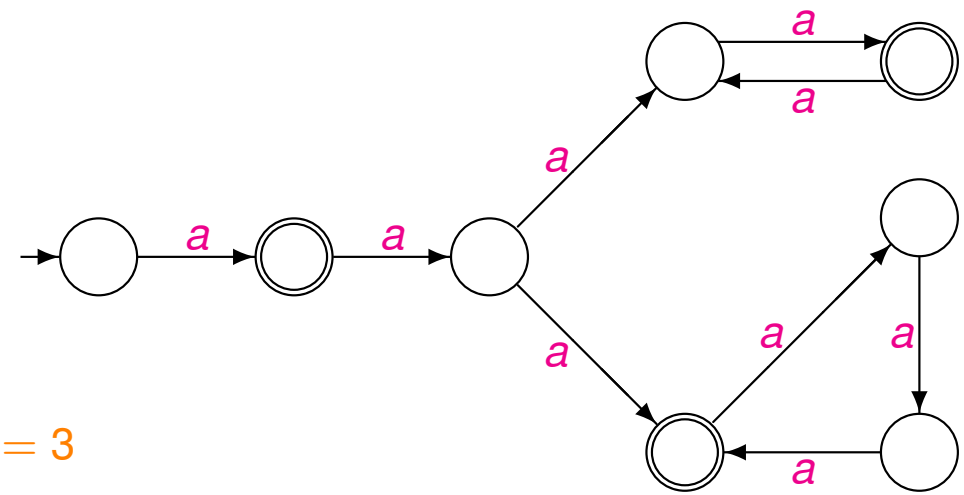
$\mu = 3, \lambda_1 = 2, \lambda_2 = 3$

On input a^m:

- First scan – simulation of the initial path:
  if $m < 3$ then stop and accept iff $m = 1$
- Second scan – simulation of the first loop:
  compute m MOD 2: if the result is 0 then stop and accept
- Third scan – simulation of the second loop:
  compute m MOD 3: if the result is 0 then stop and accept
- Finally:
  reject

Sakoda&Sipser question: 1nfa vs 2dfa

Example



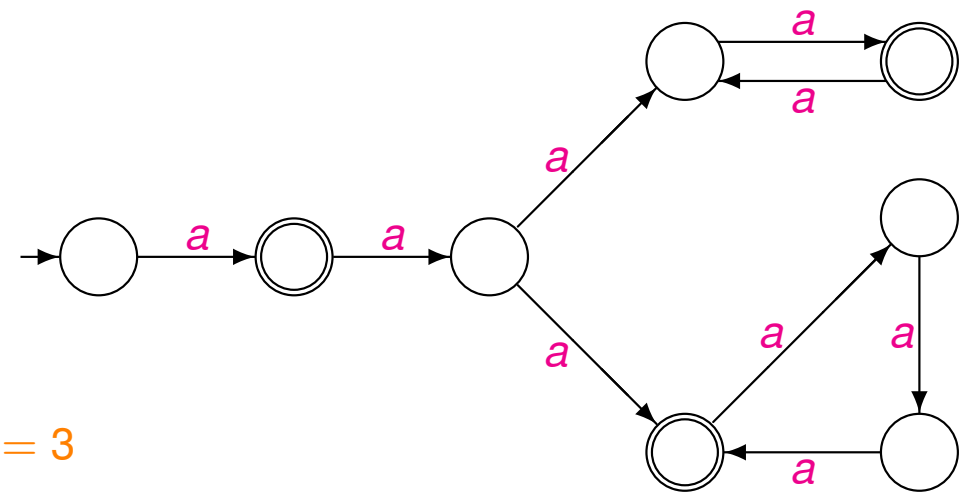
$\mu = 3, \lambda_1 = 2, \lambda_2 = 3$

On input a^m:

- First scan – simulation of the initial path:
  if $m < 3$ then stop and accept iff $m = 1$
- Second scan – simulation of the first loop:
  compute m MOD 2: if the result is 0 then stop and accept
- Third scan – simulation of the second loop:
  compute m MOD 3: if the result is 0 then stop and accept
- Finally:
  reject

Sakoda&Sipser question: 1nfa vs 2dfa

Example



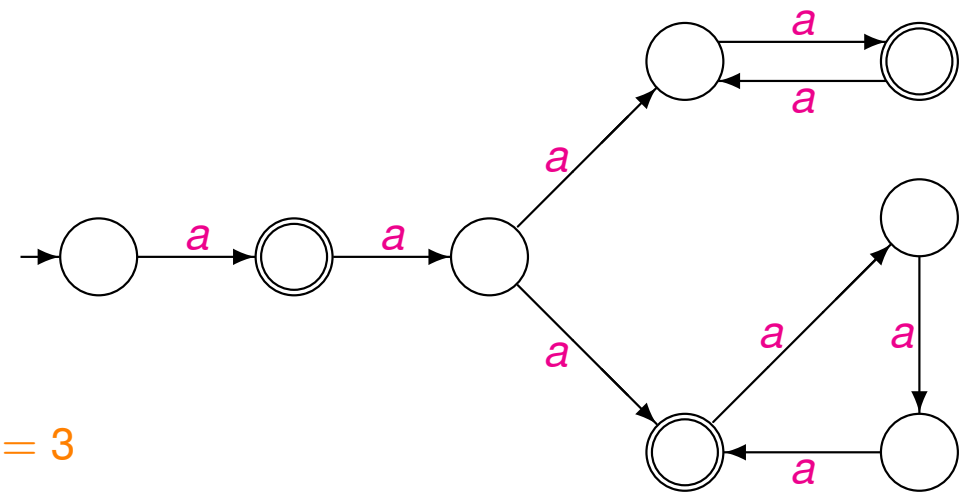
$\mu = 3, \lambda_1 = 2, \lambda_2 = 3$

On input a^m:

- First scan – simulation of the initial path:
  if $m < 3$ then stop and accept iff $m = 1$
- Second scan – simulation of the first loop:
  compute m MOD 2: if the result is 0 then stop and accept
- Third scan – simulation of the second loop:
  compute m MOD 3: if the result is 0 then stop and accept
- Finally:
  reject

Sakoda&Sipser question: 1nfa vs 2dfa

Example



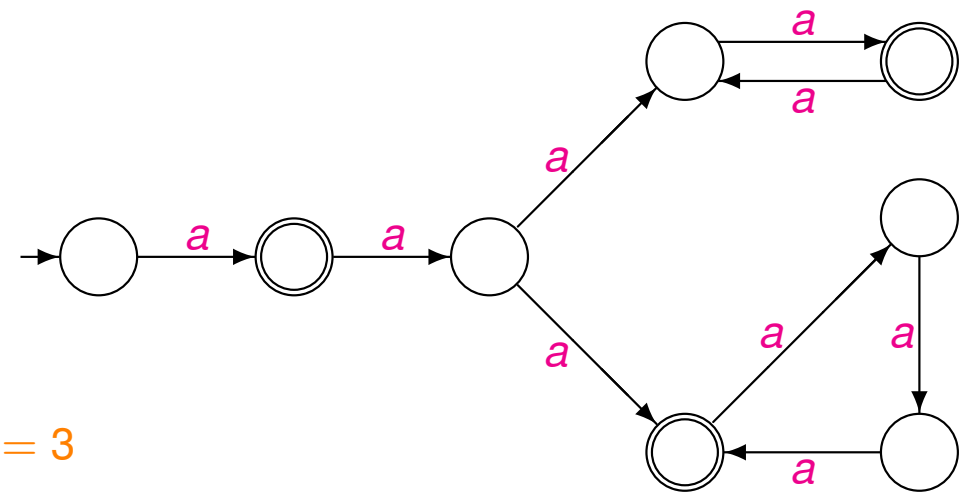
$\mu = 3, \lambda_1 = 2, \lambda_2 = 3$

On input a^m:

- First scan – simulation of the initial path:
  if $m < 3$ then stop and accept iff $m = 1$
- Second scan – simulation of the first loop:
  compute m MOD 2: if the result is 0 then stop and accept
- Third scan – simulation of the second loop:
  compute m MOD 3: if the result is 0 then stop and accept
- Finally:
  reject

Sakoda&Sipser question: 1nfa vs 2dfa

Example



$\mu = 3, \lambda_1 = 2, \lambda_2 = 3$

On input a^m:

- First scan – simulation of the initial path:
  if $m < 3$ then stop and accept iff $m = 1$
- Second scan – simulation of the first loop:
  compute m MOD 2: if the result is 0 then stop and accept
- Third scan – simulation of the second loop:
  compute m MOD 3: if the result is 0 then stop and accept
- Finally:
  reject

Sakoda&Sipser question: 1nfa vs 2dfa

How to reduce unary 1nfa to 2dfa?

Given a 1nfa in Chrobak normal form with:

- an initial path of μ states, and
- k cycles of $\lambda_1, \ldots, \lambda_k$ states

we build a 2dfa making the following steps, on input a^m:

- First scan – simulation of the initial path: μ states
  check if $m < \mu$
- Second scan – simulation of the first loop: λ_1 states
  compute $m \,\mathrm{MOD}\, \lambda_1$
- Third scan – simulation of the second loop: λ_2 states
  ...
- $(k+1)$th scan – simulation of the kth loop: λ_k states
  compute $m \,\mathrm{MOD}\, \lambda_k$

The total number of states is $\mu + \lambda_1 + \lambda_2 + \ldots + \lambda_k$

Sakoda&Sipser question: 1nfa vs 2dfa

How to reduce unary 1nfa to 2dfa?

Given a 1nfa in Chrobak normal form with:

- an initial path of μ states, and
- k cycles of $\lambda_1, \ldots, \lambda_k$ states

we build a 2dfa making the following steps, on input a^m:

- First scan – simulation of the initial path: μ states
  check if $m < \mu$
- Second scan – simulation of the first loop: λ_1 states
  compute $m \,\mathrm{MOD}\, \lambda_1$
- Third scan – simulation of the second loop: λ_2 states
  ...
- $(k+1)$th scan – simulation of the kth loop: λ_k states
  compute $m \,\mathrm{MOD}\, \lambda_k$

The total number of states is $\mu + \lambda_1 + \lambda_2 + \ldots + \lambda_k$

Sakoda&Sipser question: 1nfa vs 2dfa

How to reduce unary 1nfa to 2dfa?

Given a 1nfa in Chrobak normal form with:

- an initial path of μ states, and
- k cycles of $\lambda_1, \ldots, \lambda_k$ states

we build a 2dfa making the following steps, on input a^m:

- First scan – simulation of the initial path: μ states
  check if $m < \mu$
- Second scan – simulation of the first loop: λ_1 states
  compute $m \,\mathrm{MOD}\, \lambda_1$
- Third scan – simulation of the second loop: λ_2 states
  ...
- $(k+1)$th scan – simulation of the kth loop: λ_k states
  compute $m \,\mathrm{MOD}\, \lambda_k$

The total number of states is $\mu + \lambda_1 + \lambda_2 + \ldots + \lambda_k$

Sakoda&Sipser question: 1nfa vs 2dfa

How to reduce unary 1nfa to 2dfa?

Given a 1nfa in Chrobak normal form with:

- an initial path of μ states, and
- k cycles of $\lambda_1, \ldots, \lambda_k$ states

we build a 2dfa making the following steps, on input a^m:

- First scan – simulation of the initial path: μ states
  check if $m < \mu$
- Second scan – simulation of the first loop: λ_1 states
  compute $m \,\mathrm{MOD}\, \lambda_1$
- Third scan – simulation of the second loop: λ_2 states
  ...
- $(k+1)$th scan – simulation of the kth loop: λ_k states
  compute $m \,\mathrm{MOD}\, \lambda_k$

The total number of states is $\mu + \lambda_1 + \lambda_2 + \ldots + \lambda_k$

Sakoda&Sipser question: 1nfa vs 2dfa

How to reduce unary 1nfa to 2dfa?

Given a 1nfa in Chrobak normal form with:

- an initial path of μ states, and
- k cycles of $\lambda_1, \ldots, \lambda_k$ states

we build a 2dfa making the following steps, on input a^m:

- First scan – simulation of the initial path: μ states
  check if $m < \mu$
- Second scan – simulation of the first loop: λ_1 states
  compute $m \,\mathrm{MOD}\, \lambda_1$
- Third scan – simulation of the second loop: λ_2 states
  ...
- $(k+1)$th scan – simulation of the kth loop: λ_k states
  compute $m \,\mathrm{MOD}\, \lambda_k$

The total number of states is $\mu + \lambda_1 + \lambda_2 + \ldots + \lambda_k$

Sakoda&Sipser question: 1nfa vs 2dfa

How to reduce unary 1nfa to 2dfa?

Given a 1nfa in Chrobak normal form with:

- an initial path of μ states, and
- k cycles of $\lambda_1, \ldots, \lambda_k$ states

we build a 2dfa making the following steps, on input a^m:

- First scan – simulation of the initial path: μ states
  check if $m < \mu$
- Second scan – simulation of the first loop: λ_1 states
  compute $m \,\mathrm{MOD}\, \lambda_1$
- Third scan – simulation of the second loop: λ_2 states
  ...
- $(k+1)$th scan – simulation of the kth loop: λ_k states
  compute $m \,\mathrm{MOD}\, \lambda_k$

The total number of states is $\mu + \lambda_1 + \lambda_2 + \ldots + \lambda_k$

Sakoda&Sipser question: 1nfa vs 2dfa

How to reduce unary 1nfa to 2dfa?

Given a 1nfa in Chrobak normal form with:

- an initial path of μ states, and
- k cycles of $\lambda_1, \ldots, \lambda_k$ states

we build a 2dfa making the following steps, on input a^m:

- First scan – simulation of the initial path: μ states
  check if $m < \mu$
- Second scan – simulation of the first loop: λ_1 states
  compute $m \,\mathrm{MOD}\, \lambda_1$
- Third scan – simulation of the second loop: λ_2 states
  ...
- $(k+1)$th scan – simulation of the kth loop: λ_k states
  compute $m \,\mathrm{MOD}\, \lambda_k$

The total number of states is $\mu + \lambda_1 + \lambda_2 + \ldots + \lambda_k$

Sakoda&Sipser question: 1nfa vs 2dfa

How to reduce unary 1nfa to 2dfa?

Given a 1nfa in Chrobak normal form with:

- an initial path of μ states, and
- k cycles of $\lambda_1, \ldots, \lambda_k$ states

we build a 2dfa making the following steps, on input a^m:

- First scan – simulation of the initial path: μ states
  check if $m < \mu$
- Second scan – simulation of the first loop: λ_1 states
  compute $m \,\mathrm{MOD}\, \lambda_1$
- Third scan – simulation of the second loop: λ_2 states
  ...
  $\ldots$
- $(k+1)$th scan – simulation of the kth loop: λ_k states
  compute $m \,\mathrm{MOD}\, \lambda_k$

The total number of states is $\mu + \lambda_1 + \lambda_2 + \ldots + \lambda_k$

Sakoda&Sipser question: 1nfa vs 2dfa

How to reduce unary 1nfa to 2dfa?

Given a 1nfa in Chrobak normal form with:

- an initial path of μ states, and
- k cycles of $\lambda_1, \ldots, \lambda_k$ states

we build a 2dfa making the following steps, on input a^m:

- First scan – simulation of the initial path: μ states
  check if $m < \mu$
- Second scan – simulation of the first loop: λ_1 states
  compute $m \,\mathrm{MOD}\, \lambda_1$
- Third scan – simulation of the second loop: λ_2 states
  ...
  $\ldots$
- $(k+1)$th scan – simulation of the kth loop: λ_k states
  compute $m \,\mathrm{MOD}\, \lambda_k$

The total number of states is $\mu + \lambda_1 + \lambda_2 + \ldots + \lambda_k$

Sakoda&Sipser question: 1nfa vs 2dfa

Cost of the simulation of unary 1nfa by 2dfa

- Each unary 1nfa in Chrobak normal form with:
  - an initial path of μ states, and
  - k cycles of $\lambda_1, \ldots, \lambda_k$ states

  can be simulated by a 2dfa with $\mu + \lambda_1 + \ldots + \lambda_k$ states.
- Each unary n-state 1nfa can be simulated by a 1nfa in Chrobak normal form with $O(n^2)$ states.

Hence:

Theorem ([Chrobak 1986])

Each unary n-state 1nfa can be simulated by a 2dfa with $O(n^2)$ states.

This gives a *quadratic upper bound* for the simulation of 1nfa by 2dfa in the unary case.

Sakoda&Sipser question: 1nfa vs 2dfa

Cost of the simulation of unary 1nfa by 2dfa

- Each unary 1nfa in Chrobak normal form with:
  - an initial path of μ states, and
  - k cycles of $\lambda_1, \ldots, \lambda_k$ states

  can be simulated by a 2dfa with $\mu + \lambda_1 + \ldots + \lambda_k$ states.
- Each unary n-state 1nfa can be simulated by a 1nfa in Chrobak normal form with $O(n^2)$ states.

Hence:

Theorem ([Chrobak 1986])

Each unary n-state 1nfa can be simulated by a 2dfa with $O(n^2)$ states.

This gives a *quadratic upper bound* for the simulation of 1nfa by 2dfa in the unary case.

Sakoda&Sipser question: 1nfa vs 2dfa

Cost of the simulation of unary 1nfa by 2dfa

- Each unary 1nfa in Chrobak normal form with:
  - an initial path of μ states, and
  - k cycles of $\lambda_1, \ldots, \lambda_k$ states

  can be simulated by a 2dfa with $\mu + \lambda_1 + \ldots + \lambda_k$ states.
- Each unary n-state 1nfa can be simulated by a 1nfa in Chrobak normal form with $O(n^2)$ states.

Hence:

Theorem ([Chrobak 1986])

Each unary n-state 1nfa can be simulated by a 2dfa with $O(n^2)$ states.

This gives a *quadratic upper bound* for the simulation of 1nfa by 2dfa in the unary case.

Sakoda&Sipser question: 1nfa vs 2dfa

Cost of the simulation of unary 1nfa by 2dfa

- Each unary 1nfa in Chrobak normal form with:
  - an initial path of μ states, and
  - k cycles of $\lambda_1, \ldots, \lambda_k$ states

  can be simulated by a 2dfa with $\mu + \lambda_1 + \ldots + \lambda_k$ states.
- Each unary n-state 1nfa can be simulated by a 1nfa in Chrobak normal form with $O(n^2)$ states.

Hence:

Theorem ([Chrobak 1986])

Each unary n-state 1nfa can be simulated by a 2dfa with $O(n^2)$ states.

This gives a *quadratic upper bound* for the simulation of 1nfa by 2dfa in the unary case.

Sakoda&Sipser question: 1nfa vs 2dfa

Cost of the simulation of unary 1nfa by 2dfa

- Each unary 1nfa in Chrobak normal form with:
  - an initial path of μ states, and
  - k cycles of $\lambda_1, \ldots, \lambda_k$ states

  can be simulated by a 2dfa with $\mu + \lambda_1 + \ldots + \lambda_k$ states.
- Each unary n-state 1nfa can be simulated by a 1nfa in Chrobak normal form with $O(n^2)$ states.

Hence:

Theorem ([Chrobak 1986])

Each unary n-state 1nfa can be simulated by a 2dfa with $O(n^2)$ states.

This gives a *quadratic upper bound* for the simulation of 1nfa by 2dfa in the unary case.

Sakoda&Sipser question: 1nfa vs 2dfa

Cost of the simulation of unary 1nfa by 2dfa

- Each unary 1nfa in Chrobak normal form with:
  - an initial path of μ states, and
  - k cycles of $\lambda_1, \ldots, \lambda_k$ states

  can be simulated by a 2dfa with $\mu + \lambda_1 + \ldots + \lambda_k$ states.
- Each unary n-state 1nfa can be simulated by a 1nfa in Chrobak normal form with $O(n^2)$ states.

Hence:

Theorem ([Chrobak 1986])

Each unary n-state 1nfa can be simulated by a 2dfa with $O(n^2)$ states.

This gives a *quadratic upper bound* for the simulation of 1nfa by 2dfa in the unary case.

Sakoda&Sipser question: 1nfa vs 2dfa

What about the lower bound?

For each $n \geq 2$ consider the language

$$L_n = \{a^m \mid m = \alpha n + \beta(n-1), n, m \in \mathbf{N}\}$$

It is possible to prove that:

- L_n is accepted by a 1nfa with n states
- each 2dfa accepting L_n requires $\Omega(n^2)$ states.

Hence, even the lower bound is *quadratic*.

This solves the 1nfa vs 2dfa question in the unary case [Chrobak 1986]

Sakoda&Sipser question: 1nfa vs 2dfa

What about the lower bound?

For each $n \geq 2$ consider the language

$$L_n = \{a^m \mid m = \alpha n + \beta(n-1), n, m \in \mathbf{N}\}$$

It is possible to prove that:

- L_n is accepted by a 1nfa with n states
- each 2dfa accepting L_n requires $\Omega(n^2)$ states.

Hence, even the lower bound is *quadratic*.

This solves the 1nfa vs 2dfa question in the unary case [Chrobak 1986]

Sakoda&Sipser question: 1nfa vs 2dfa

What about the lower bound?

For each $n \geq 2$ consider the language

$$L_n = \{a^m \mid m = \alpha n + \beta(n-1), n, m \in \mathbf{N}\}$$

It is possible to prove that:

- L_n is accepted by a 1nfa with n states
- each 2dfa accepting L_n requires $\Omega(n^2)$ states.

Hence, even the lower bound is *quadratic*.

This solves the 1nfa vs 2dfa question in the unary case [Chrobak 1986]

Sakoda&Sipser question: 1nfa vs 2dfa

What about the lower bound?

For each $n \geq 2$ consider the language

$$L_n = \{a^m \mid m = \alpha n + \beta(n-1), n, m \in \mathbf{N}\}$$

It is possible to prove that:

- L_n is accepted by a 1nfa with n states
- each 2dfa accepting L_n requires $\Omega(n^2)$ states.

Hence, even the lower bound is *quadratic*.

This solves the 1nfa vs 2dfa question in the unary case [Chrobak 1986]

Sakoda&Sipser question: 1nfa vs 2dfa

What about the lower bound?

For each $n \geq 2$ consider the language

$$L_n = \{a^m \mid m = \alpha n + \beta(n-1), n, m \in \mathbf{N}\}$$

It is possible to prove that:

- L_n is accepted by a 1nfa with n states
- each 2dfa accepting L_n requires $\Omega(n^2)$ states.

Hence, even the lower bound is *quadratic*.

This solves the 1nfa vs 2dfa question in the unary case [Chrobak 1986]

Sakoda&Sipser question: 1nfa vs 2dfa

What about the lower bound?

For each $n \geq 2$ consider the language

$$L_n = \{a^m \mid m = \alpha n + \beta(n-1), n, m \in \mathbf{N}\}$$

It is possible to prove that:

- L_n is accepted by a 1nfa with n states
- each 2dfa accepting L_n requires $\Omega(n^2)$ states.

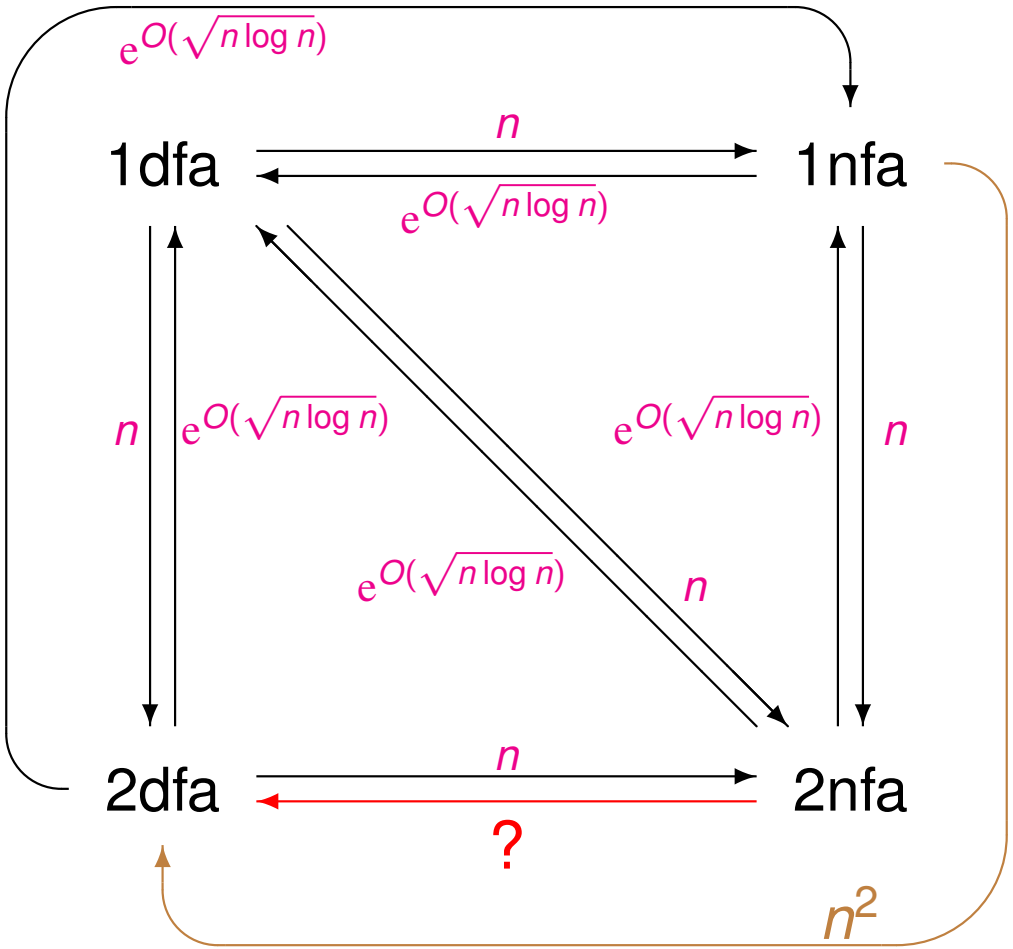
Hence, even the lower bound is *quadratic*.

This solves the 1nfa vs 2dfa question in the unary case [Chrobak 1986]

Simulations between unary automata

Costs ot the optimal simulations between unary automata

[Chrobak 1986, Mereghetti and Pighizzini 2001]



Sakoda&Sipser question: 2nfa vs 2dfa

What about the simulation of unary 2nfa by 2dfa?

- Exponential upper bound: $e^{O(\sqrt{n \log n})}$ (simulation of 2nfa by 1dfa)
- Quadratic lower bound: $\Omega(n^2)$ (simulation of 1nfa by 2dfa)

It is possible either to increase the lower bound or to decrease the upper bound?

Theorem ([Geffert, Mereghetti, Pighizzini 2003])

Each unary n-state 2nfa can be simulated by a 2dfa with $n^{O(\log n)}$ states.

Remark: The function $n^{O(\log n)}$ is not polynomial, but it grows less than any exponential function $2^{n^{O(1)}}$.

Sakoda&Sipser question: 2nfa vs 2dfa

What about the simulation of unary 2nfa by 2dfa?

- Exponential upper bound: $e^{O(\sqrt{n \log n})}$
  (simulation of 2nfa by 1dfa)
- Quadratic lower bound: $\Omega(n^2)$
  (simulation of 1nfa by 2dfa)

It is possible either to increase the lower bound or to decrease the upper bound?

Theorem ([Geffert, Mereghetti, Pighizzini 2003])

Each unary n-state 2nfa can be simulated by a 2dfa with $n^{O(\log n)}$ states.

Remark: The function $n^{O(\log n)}$ is not polynomial, but it grows less than any exponential function $2^{n^{O(1)}}$.

Sakoda&Sipser question: 2nfa vs 2dfa

What about the simulation of unary 2nfa by 2dfa?

- Exponential upper bound: $e^{O(\sqrt{n \log n})}$
  (simulation of 2nfa by 1dfa)
- Quadratic lower bound: $\Omega(n^2)$
  (simulation of 1nfa by 2dfa)

It is possible either to increase the lower bound or to decrease the upper bound?

Theorem ([Geffert, Mereghetti, Pighizzini 2003])

Each unary n-state 2nfa can be simulated by a 2dfa with $n^{O(\log n)}$ states.

Remark: The function $n^{O(\log n)}$ is not polynomial, but it grows less than any exponential function $2^{n^{O(1)}}$.

Sakoda&Sipser question: 2nfa vs 2dfa

What about the simulation of unary 2nfa by 2dfa?

- Exponential upper bound: $e^{O(\sqrt{n \log n})}$
  (simulation of 2nfa by 1dfa)
- Quadratic lower bound: $\Omega(n^2)$
  (simulation of 1nfa by 2dfa)

It is possible either to increase the lower bound or to decrease the upper bound?

Theorem ([Geffert, Mereghetti, Pighizzini 2003])

Each unary n-state 2nfa can be simulated by a 2dfa with $n^{O(\log n)}$ states.

Remark: The function $n^{O(\log n)}$ is not polynomial, but it grows less than any exponential function $2^{n^{O(1)}}$.

Sakoda&Sipser question: 2nfa vs 2dfa

What about the simulation of unary 2nfa by 2dfa?

- Exponential upper bound: $e^{O(\sqrt{n \log n})}$
  (simulation of 2nfa by 1dfa)
- Quadratic lower bound: $\Omega(n^2)$
  (simulation of 1nfa by 2dfa)

It is possible either to increase the lower bound or to decrease the upper bound?

Theorem ([Geffert, Mereghetti, Pighizzini 2003])

Each unary n-state 2nfa can be simulated by a 2dfa with $n^{O(\log n)}$ states.

Remark: The function $n^{O(\log n)}$ is not polynomial, but it grows less than any exponential function $2^{n^{O(1)}}$.

Sakoda&Sipser question: 2nfa vs 2dfa

What about the simulation of unary 2nfa by 2dfa?

- Exponential upper bound: $e^{O(\sqrt{n \log n})}$
  (simulation of 2nfa by 1dfa)
- Quadratic lower bound: $\Omega(n^2)$
  (simulation of 1nfa by 2dfa)

It is possible either to increase the lower bound or to decrease the upper bound?

Theorem ([Geffert, Mereghetti, Pighizzini 2003])

Each unary n-state 2nfa can be simulated by a 2dfa with $n^{O(\log n)}$ states.

Remark: The function $n^{O(\log n)}$ is not polynomial, but it grows less than any exponential function $2^{n^{O(1)}}$.

Simulation of unary 2nfa by 2dfa: some definitions

Definition

Two finite automata are *almost equivalent* iff their accepted languages coincide, with the possible exception of a finite number of strings.

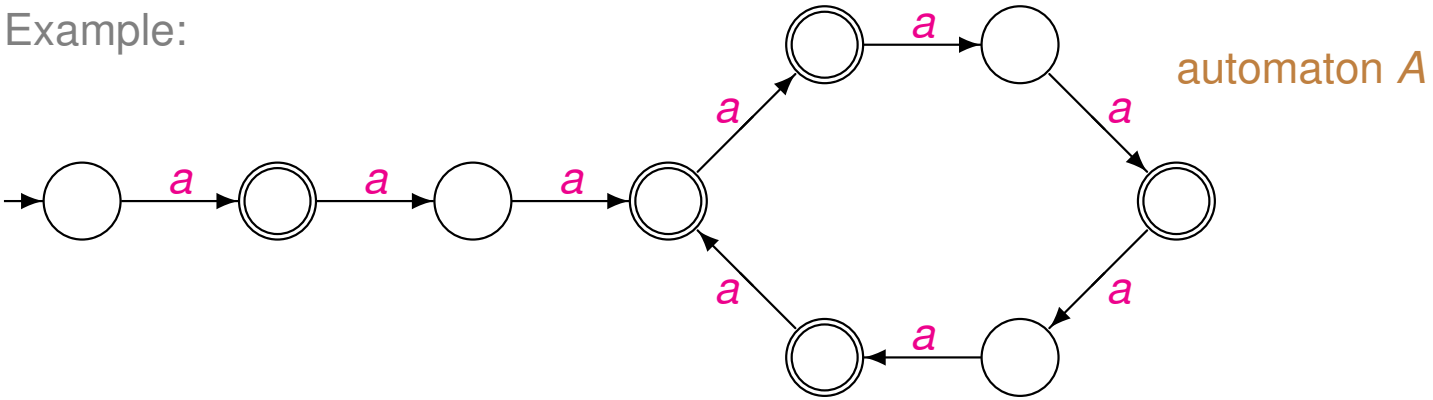
Example:

Simulation of unary 2nfa by 2dfa: some definitions

Definition

Two finite automata are *almost equivalent* iff their accepted languages coincide, with the possible exception of a finite number of strings.

Example:



$L(A) - L(B) = \{a, a^3\}$

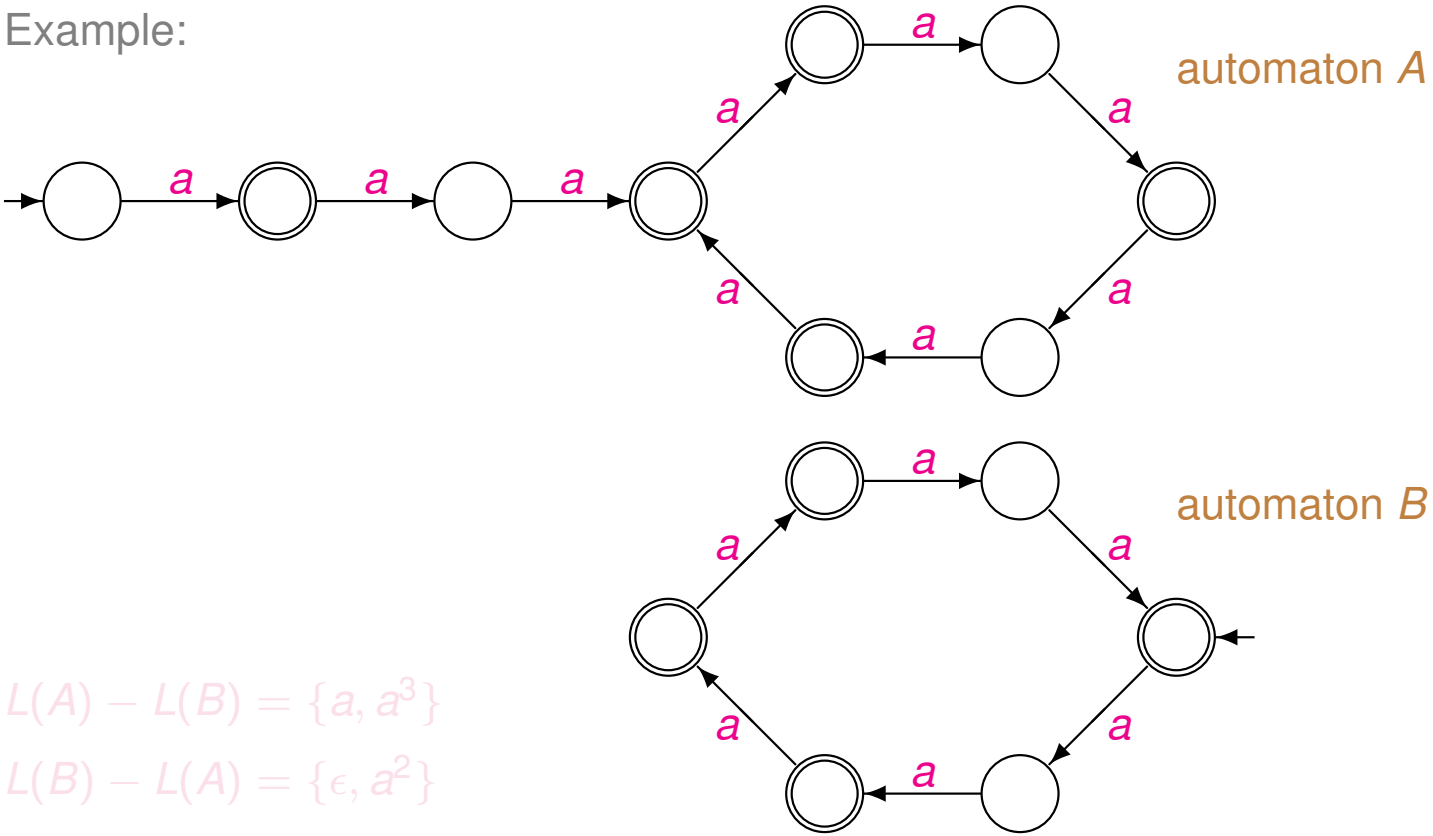
$L(B) - L(A) = \{\epsilon, a^2\}$

Simulation of unary 2nfa by 2dfa: some definitions

Definition

Two finite automata are *almost equivalent* iff their accepted languages coincide, with the possible exception of a finite number of strings.

Example:



$L(A) - L(B) = \{a, a^3\}$

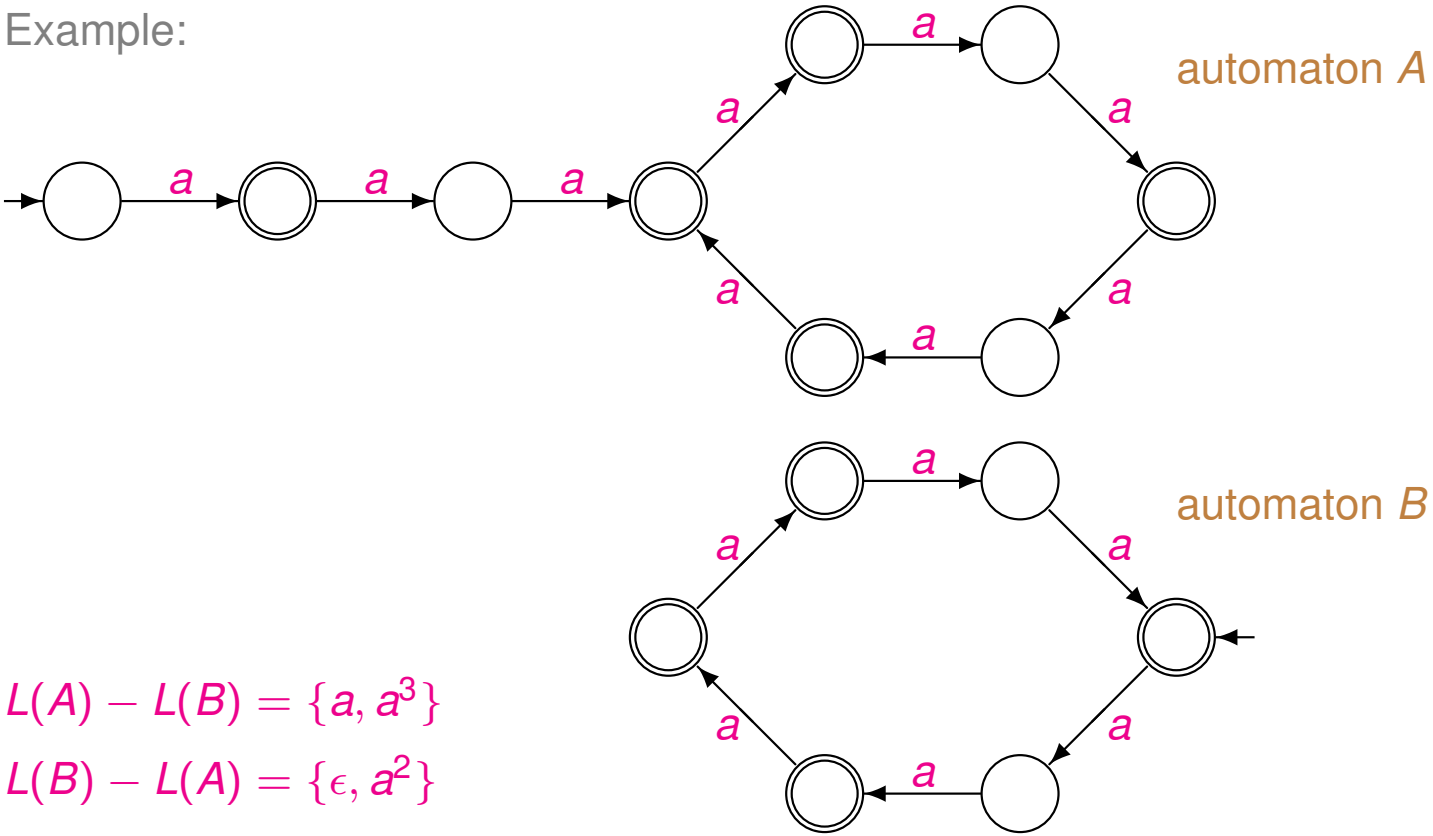
$L(B) - L(A) = \{\epsilon, a^2\}$

Simulation of unary 2nfa by 2dfa: some definitions

Definition

Two finite automata are *almost equivalent* iff their accepted languages coincide, with the possible exception of a finite number of strings.

Example:



$L(A) - L(B) = \{a, a^3\}$

$L(B) - L(A) = \{\epsilon, a^2\}$

Simulation of unary 2nfa by 2dfa: some definitions

Sweeping automaton:

- deterministic
- head reversals only at the endmarkers

Definition

A two-way automaton is *quasi sweeping* iff both

- nondeterministic choices, and
- head reversals

are possible only when the input head is scanning one of the endmarkers.

Simulation of unary 2nfa by 2dfa: some definitions

Sweeping automaton:

- deterministic
- head reversals only at the endmarkers

Definition

A two-way automaton is *quasi sweeping* iff both

- nondeterministic choices, and
- head reversals

are possible only when the input head is scanning one of the endmarkers.

Simulation of unary 2nfa by 2dfa: some definitions

Sweeping automaton:

- deterministic
- head reversals only at the endmarkers

Definition

A two-way automaton is *quasi sweeping* iff both

- nondeterministic choices, and
- head reversals

are possible only when the input head is scanning one of the endmarkers.

Simulation of unary 2nfa by 2dfa: some definitions

Sweeping automaton:

- deterministic
- head reversals only at the endmarkers

Definition

A two-way automaton is *quasi sweeping* iff both

- nondeterministic choices, and
- head reversals

are possible only when the input head is scanning one of the endmarkers.

Simulation of unary 2nfa by 2dfa: some definitions

Sweeping automaton:

- deterministic
- head reversals only at the endmarkers

Definition

A two-way automaton is *quasi sweeping* iff both

- nondeterministic choices, and
- head reversals

are possible only when the input head is scanning one of the endmarkers.

Simulation of unary 2nfa by 2dfa

Outline of the proof:

Lemma (from 2nfa to quasi sweeping 2nfa)

For each unary n-state 2nfa A there exists an almost equivalent quasi sweeping 2nfa A' with no more than $2n+2$ states.

Furthermore, the languages $L(A)$ and $L(A')$ can differ only on strings of length $\leq 5n^2$.

Lemma (from quasi sweeping 2nfa to 2dfa)

Each unary n-state quasi sweeping 2nfa can be simulated by a 2dfa with $O(n^{\lceil \log_2(n+1) \rceil + 3})$ states.

Simulation of unary 2nfa by 2dfa

Outline of the proof:

Lemma (from 2nfa to quasi sweeping 2nfa)

For each unary n-state 2nfa A there exists an almost equivalent quasi sweeping 2nfa A' with no more than $2n+2$ states.

Furthermore, the languages $L(A)$ and $L(A')$ can differ only on strings of length $\leq 5n^2$.

Lemma (from quasi sweeping 2nfa to 2dfa)

Each unary n-state quasi sweeping 2nfa can be simulated by a 2dfa with $O(n^{\lceil \log_2(n+1) \rceil + 3})$ states.

Simulation of unary 2nfa by 2dfa

Outline of the proof:

Lemma (from 2nfa to quasi sweeping 2nfa)

For each unary n-state 2nfa A there exists an almost equivalent quasi sweeping 2nfa A' with no more than $2n+2$ states.

Furthermore, the languages $L(A)$ and $L(A')$ can differ only on strings of length $\leq 5n^2$.

Lemma (from quasi sweeping 2nfa to 2dfa)

Each unary n-state quasi sweeping 2nfa can be simulated by a 2dfa with $O(n^{\lceil \log_2(n+1) \rceil + 3})$ states.

Simulation of unary 2nfa by 2dfa

Outline of the proof:

Lemma (from 2nfa to quasi sweeping 2nfa)

For each unary n-state 2nfa A there exists an almost equivalent quasi sweeping 2nfa A' with no more than $2n+2$ states.

Furthermore, the languages $L(A)$ and $L(A')$ can differ only on strings of length $\leq 5n^2$.

Lemma (from quasi sweeping 2nfa to 2dfa)

Each unary n-state quasi sweeping 2nfa can be simulated by a 2dfa with $O(n^{\lceil \log_2(n+1) \rceil + 3})$ states.

Simulation of unary 2nfa by 2dfa

Outline of the proof:

0. $A :=$ the unary n-state 2nfa we have to simulate
1. $A' :=$ 2nfa built from A s.t.:
   - A' almost equivalent to A and quasi sweeping
   - $2n + 2$ states
2. $B :=$ 2dfa built from A' s.t.:
   - B equivalent to A'
   - B almost equivalent to A
   - $n^{O(\log n)}$ states
3. $C :=$ the following dfa:

   first C checks if the input length is $\leq n^2$ and, in this case, it accepts iff A accepts

   hence if the input length is $> n^2$ then C directly simulates B
   - C is equivalent to A
   - C has $n^{O(\log n)}$ states

Simulation of unary 2nfa by 2dfa

Outline of the proof:

0. $A :=$ the unary n-state 2nfa we have to simulate
1. $A' :=$ 2nfa built from A s.t.:
   - A' almost equivalent to A and quasi sweeping
   - $2n + 2$ states
2. $B :=$ 2dfa built from A' s.t.:
   - B equivalent to A'
   - B almost equivalent to A
   - $n^{O(\log n)}$ states
3. $C :=$ the following dfa:

   first C checks if the input length is $\leq n^2$ and, in this case, it accepts iff A accepts

   hence if the input length is $> n^2$ then C directly simulates B
   - C is equivalent to A
   - C has $n^{O(\log n)}$ states

Simulation of unary 2nfa by 2dfa

Outline of the proof:

0. $A :=$ the unary n-state 2nfa we have to simulate

Lemma (from 2nfa to quasi sweeping 2nfa)

For each unary n-state 2nfa A there exists an almost equivalent quasi sweeping 2nfa A' with no more than $2n + 2$ states.

Furthermore, the languages $L(A)$ and $L(A')$ can differ only on strings of length $\leq 5n^2$.

1. $A' :=$ 2nfa built from A s.t.:
   - A' almost equivalent to A and quasi sweeping
   - $2n + 2$ states
2. $B :=$ 2dfa built from A' s.t.:
   - B equivalent to A'
   - B almost equivalent to A
   - $n^{O(\log n)}$ states
3. $C :=$ the following dfa:

Simulation of unary 2nfa by 2dfa

Outline of the proof:

0. $A :=$ the unary n-state 2nfa we have to simulate

Lemma (from 2nfa to quasi sweeping 2nfa)

For each unary n-state 2nfa A there exists an almost equivalent quasi sweeping 2nfa A' with no more than $2n + 2$ states.

Furthermore, the languages $L(A)$ and $L(A')$ can differ only on strings of length $\leq 5n^2$.

1. $A' :=$ 2nfa built from A s.t.:
   - A' almost equivalent to A and quasi sweeping
   - $2n + 2$ states
2. $B :=$ 2dfa built from A' s.t.:
   - B equivalent to A'
   - B almost equivalent to A
   - $n^{O(\log n)}$ states
3. $C :=$ the following dfa:

Simulation of unary 2nfa by 2dfa

Outline of the proof:

0. $A :=$ the unary n-state 2nfa we have to simulate

Lemma (from 2nfa to quasi sweeping 2nfa)

For each unary n-state 2nfa A there exists an almost equivalent quasi sweeping 2nfa A' with no more than $2n + 2$ states.

Furthermore, the languages $L(A)$ and $L(A')$ can differ only on strings of length $\leq 5n^2$.

1. $A' :=$ 2nfa built from A s.t.:
   - A' almost equivalent to A and quasi sweeping
   - $2n + 2$ states
2. $B :=$ 2dfa built from A' s.t.:
   - B equivalent to A'
   - B almost equivalent to A
   - $n^{O(\log n)}$ states
3. $C :=$ the following dfa:

Simulation of unary 2nfa by 2dfa

Outline of the proof:

0. $A :=$ the unary n-state 2nfa we have to simulate
1. $A' :=$ 2nfa built from A s.t.:
   - A' almost equivalent to A and quasi sweeping
   - $2n + 2$ states
2. $B :=$ 2dfa built from A' s.t.:
   - B equivalent to A'
   - B almost equivalent to A
   - $n^{O(\log n)}$ states
3. $C :=$ the following dfa:

   first C checks if the input length is $\leq n^2$ and, in this case, it accepts iff A accepts

   hence if the input length is $> n^2$ then C directly simulates B
   - C is equivalent to A
   - C has $n^{O(\log n)}$ states

Simulation of unary 2nfa by 2dfa

Outline of the proof:

0. $A :=$ the unary n-state 2nfa we have to simulate
1. $A' :=$ 2nfa built from A s.t.:
   - A' almost equivalent to A and quasi sweeping
   - $2n + 2$ states

Lemma (from quasi sweeping 2nfa to 2dfa)

Each unary n-state quasi sweeping 2nfa can be simulated by a 2dfa with $O(n^{\lceil \log_2(n+1) \rceil + 3})$ states.

2. $B :=$ 2dfa built from A' s.t.:
   - B equivalent to A'
   - B almost equivalent to A
   - $n^{O(\log n)}$ states
3. $C :=$ the following dfa:
   first C checks if the input length is $\leq n^2$ and, in this case, it accepts iff A accepts
   hence if the input length is $> n^2$ then C directly simulates B

Simulation of unary 2nfa by 2dfa

Outline of the proof:

0. $A :=$ the unary n-state 2nfa we have to simulate
1. $A' :=$ 2nfa built from A s.t.:
   - A' almost equivalent to A and quasi sweeping
   - $2n + 2$ states

Lemma (from quasi sweeping 2nfa to 2dfa)

Each unary n-state quasi sweeping 2nfa can be simulated by a 2dfa with $O(n^{\lceil \log_2(n+1) \rceil + 3})$ states.

2. $B :=$ 2dfa built from A' s.t.:
   - B equivalent to A'
   - B almost equivalent to A
   - $n^{O(\log n)}$ states
3. $C :=$ the following dfa:
   first C checks if the input length is $\leq n^2$ and, in this case, it accepts iff A accepts
   hence if the input length is $> n^2$ then C directly simulates B

Simulation of unary 2nfa by 2dfa

Outline of the proof:

0. $A :=$ the unary n-state 2nfa we have to simulate
1. $A' :=$ 2nfa built from A s.t.:
   - A' almost equivalent to A and quasi sweeping
   - $2n + 2$ states

Lemma (from quasi sweeping 2nfa to 2dfa)

Each unary n-state quasi sweeping 2nfa can be simulated by a 2dfa with $O(n^{\lceil \log_2(n+1) \rceil + 3})$ states.

2. $B :=$ 2dfa built from A' s.t.:
   - B equivalent to A'
   - B almost equivalent to A
   - $n^{O(\log n)}$ states
3. $C :=$ the following dfa:
   first C checks if the input length is $\leq n^2$ and, in this case, it accepts iff A accepts
   hence if the input length is $> n^2$ then C directly simulates B

Simulation of unary 2nfa by 2dfa

Outline of the proof:

0. $A :=$ the unary n-state 2nfa we have to simulate
1. $A' :=$ 2nfa built from A s.t.:
   - A' almost equivalent to A and quasi sweeping
   - $2n + 2$ states
2. $B :=$ 2dfa built from A' s.t.:
   - B equivalent to A'
   - B almost equivalent to A
   - $n^{O(\log n)}$ states
3. $C :=$ the following dfa:

   first C checks if the input length is $\leq n^2$ and, in this case, it accepts iff A accepts

   hence if the input length is $> n^2$ then C directly simulates B
   - C is equivalent to A
   - C has $n^{O(\log n)}$ states

Simulation of unary 2nfa by 2dfa

Outline of the proof:

0. $A :=$ the unary n-state 2nfa we have to simulate
1. $A' :=$ 2nfa built from A s.t.:
   - A' almost equivalent to A and quasi sweeping
   - $2n + 2$ states
2. $B :=$ 2dfa built from A' s.t.:
   - B equivalent to A'
   - B almost equivalent to A
   - $n^{O(\log n)}$ states
3. $C :=$ the following dfa:

   first C checks if the input length is $\leq n^2$ and, in this case, it accepts iff A accepts

   hence if the input length is $> n^2$ then C directly simulates B
   - C is equivalent to A
   - C has $n^{O(\log n)}$ states

Simulation of unary 2nfa by 2dfa

Outline of the proof:

0. $A :=$ the unary n-state 2nfa we have to simulate
1. $A' :=$ 2nfa built from A s.t.:
   - A' almost equivalent to A and quasi sweeping
   - $2n + 2$ states
2. $B :=$ 2dfa built from A' s.t.:
   - B equivalent to A'
   - B almost equivalent to A
   - $n^{O(\log n)}$ states
3. $C :=$ the following dfa:

   first C checks if the input length is $\leq n^2$ and, in this case, it accepts iff A accepts

   hence if the input length is $> n^2$ then C directly simulates B
   - C is equivalent to A
   - C has $n^{O(\log n)}$ states

Simulation of unary 2nfa by 2dfa

Outline of the proof:

0. $A :=$ the unary n-state 2nfa we have to simulate
1. $A' :=$ 2nfa built from A s.t.:
   - A' almost equivalent to A and quasi sweeping
   - $2n + 2$ states
2. $B :=$ 2dfa built from A' s.t.:
   - B equivalent to A'
   - B almost equivalent to A
   - $n^{O(\log n)}$ states
3. $C :=$ the following dfa:

   first C checks if the input length is $\leq n^2$ and, in this case, it accepts iff A accepts

   hence if the input length is $> n^2$ then C directly simulates B
   - C is equivalent to A
   - C has $n^{O(\log n)}$ states

Simulation of unary 2nfa by 2dfa

Outline of the proof:

0. $A :=$ the unary n-state 2nfa we have to simulate
1. $A' :=$ 2nfa built from A s.t.:
   - A' almost equivalent to A and quasi sweeping
   - $2n + 2$ states
2. $B :=$ 2dfa built from A' s.t.:
   - B equivalent to A'
   - B almost equivalent to A
   - $n^{O(\log n)}$ states
3. $C :=$ the following dfa:

   first C checks if the input length is $\leq n^2$ and, in this case, it accepts iff A accepts

   hence if the input length is $> n^2$ then C directly simulates B
   - C is equivalent to A
   - C has $n^{O(\log n)}$ states

Simulation of unary 2nfa by 2dfa

Outline of the proof:

0. $A :=$ the unary n-state 2nfa we have to simulate
1. $A' :=$ 2nfa built from A s.t.:
   - A' almost equivalent to A and quasi sweeping
   - $2n + 2$ states
2. $B :=$ 2dfa built from A' s.t.:
   - B equivalent to A'
   - B almost equivalent to A
   - $n^{O(\log n)}$ states
3. $C :=$ the following dfa:

   first C checks if the input length is $\leq n^2$ and, in this case, it accepts iff A accepts

   hence if the input length is $> n^2$ then C directly simulates B
   - C is equivalent to A
   - C has $n^{O(\log n)}$ states

First lemma: outline of the proof

Lemma (from 2nfa to quasi sweeping 2nfa)

For each unary n-state 2nfa A there exists an almost equivalent quasi sweeping 2nfa A' with no more than $2n+2$ states. Furthermore, the languages $L(A)$ and $L(A')$ can differ only on strings of length $\leq 5n^2$.

- An accepting computation of a two-way machine is a sequence of *segments*
- The segments are delimited by the configurations in which the input head visits the endmarkers
- Two kinds of segments:
  - traversals
  - *U*-turns

The simulation "simplifies" the segments of accepting computations.

First lemma: outline of the proof

Lemma (from 2nfa to quasi sweeping 2nfa)

For each unary n-state 2nfa A there exists an almost equivalent quasi sweeping 2nfa A' with no more than $2n + 2$ states. Furthermore, the languages $L(A)$ and $L(A')$ can differ only on strings of length $\leq 5n^2$.

- An accepting computation of a two-way machine is a sequence of *segments*
- The segments are delimited by the configurations in which the input head visits the endmarkers
- Two kinds of segments:
  - traversals
  - *U*-turns

The simulation "simplifies" the segments of accepting computations.

First lemma: outline of the proof

Lemma (from 2nfa to quasi sweeping 2nfa)

For each unary n-state 2nfa A there exists an almost equivalent quasi sweeping 2nfa A' with no more than $2n+2$ states. Furthermore, the languages $L(A)$ and $L(A')$ can differ only on strings of length $\leq 5n^2$.

- An accepting computation of a two-way machine is a sequence of *segments*
- The segments are delimited by the configurations in which the input head visits the endmarkers
- Two kinds of segments:
  - traversals
  - U-turns

The simulation "simplifies" the segments of accepting computations.

First lemma: outline of the proof

Lemma (from 2nfa to quasi sweeping 2nfa)

For each unary n-state 2nfa A there exists an almost equivalent quasi sweeping 2nfa A' with no more than $2n + 2$ states. Furthermore, the languages $L(A)$ and $L(A')$ can differ only on strings of length $\leq 5n^2$.

- An accepting computation of a two-way machine is a sequence of *segments*
- The segments are delimited by the configurations in which the input head visits the endmarkers
- Two kinds of segments:
  - traversals
  - *U*-turns

The simulation "simplifies" the segments of accepting computations.

First lemma: outline of the proof

Lemma (from 2nfa to quasi sweeping 2nfa)

For each unary n-state 2nfa A there exists an almost equivalent quasi sweeping 2nfa A' with no more than $2n+2$ states. Furthermore, the languages $L(A)$ and $L(A')$ can differ only on strings of length $\leq 5n^2$.

- An accepting computation of a two-way machine is a sequence of *segments*
- The segments are delimited by the configurations in which the input head visits the endmarkers
- Two kinds of segments:
  - traversals
  - U-turns

The simulation "simplifies" the segments of accepting computations.

First lemma: outline of the proof

Traversal: segment of computation starting at one endmarker and ending at the other one.

Traversals in 2nfa can be very complicated.
However, in the unary case:

For each traversal T of the input from a state q to a state p there exists another traversal T' from q to p with a very "simple" structure [Geffert 1991]:

First lemma: outline of the proof

Traversal: segment of computation starting at one endmarker and ending at the other one.

Traversals in 2nfa can be very complicated.
However, in the unary case:

For each traversal T of the input from a state q to a state p there exists another traversal T' from q to p with a very "simple" structure [Geffert 1991]:

- initial and final parts consuming $O(n^2)$ input symbols
- in the middle: a "dominant" loop repeated many times

Hence, traversals are essentially used to compute the input lenght modulo one Integer: they can be simulated by *deterministic loops* and nondeterministic moves on the endmarkers.

First lemma: outline of the proof

Traversal: segment of computation starting at one endmarker and ending at the other one.

Traversals in 2nfa can be very complicated.
However, in the unary case:

For each traversal T of the input from a state q to a state p there exists another traversal T' from q to p with a very "simple" structure [Geffert 1991]:

- initial and final parts consuming $O(n^2)$ input symbols
- in the middle: a "dominant" loop repeated many times

Hence, traversals are essentially used to computed the input lenght modulo one integer: they can be simulated by *deterministic loops* and nondeterministic moves at the endmarkers.

First lemma: outline of the proof

Traversal: segment of computation starting at one endmarker and ending at the other one.

Traversals in 2nfa can be very complicated.
However, in the unary case:

For each traversal T of the input from a state q to a state p there exists another traversal T' from q to p with a very "simple" structure [Geffert 1991]:

- initial and final parts consuming $O(n^2)$ input symbols
- in the middle: a "dominant" loop repeated many times

Hence, traversals are essentially used to computed the input lenght modulo one integer: they can be simulated by *deterministic loops* and nondeterministic moves at the endmarkers.

First lemma: outline of the proof

Traversal: segment of computation starting at one endmarker and ending at the other one.

Traversals in 2nfa can be very complicated.
However, in the unary case:

For each traversal T of the input from a state q to a state p there exists another traversal T' from q to p with a very "simple" structure [Geffert 1991]:

- initial and final parts consuming $O(n^2)$ input symbols
- in the middle: a "dominant" loop repeated many times

Hence, traversals are essentially used to computed the input lenght modulo one integer: they can be simulated by *deterministic loops* and nondeterministic moves at the endmarkers.

First lemma: outline of the proof

U-turn: segment starting and ending at the same endmarker.

For sufficiently long inputs ($> n^2$), the set of possible *U*-turns can be precomputed. Hence, they can be replaced by stationary moves [Geffert 1991].

By replacing

- traversals with deterministic loops
- *U*-turn with stationary moves,

the given n-state 2nfa A we can build an almost equivalent quasi sweeping automaton with $2n + 2$ states.

First lemma: outline of the proof

U-turn: segment starting and ending at the same endmarker.

For sufficiently long inputs ($> n^2$), the set of possible *U*-turns can be precomputed. Hence, they can be replaced by stationary moves [Geffert 1991].

By replacing

- traversals with deterministic loops
- *U*-turn with stationary moves,

the given n-state 2nfa A we can build an almost equivalent quasi sweeping automaton with $2n + 2$ states.

First lemma: outline of the proof

U-turn: segment starting and ending at the same endmarker.

For sufficiently long inputs ($> n^2$), the set of possible *U*-turns can be precomputed. Hence, they can be replaced by stationary moves [Geffert 1991].

By replacing

- traversals with deterministic loops
- *U*-turn with stationary moves,

the given n-state 2nfa A we can build an almost equivalent quasi sweeping automaton with $2n + 2$ states.

Second lemma: outline of the proof

Lemma (from quasi sweeping 2nfa to 2dfa)

Each unary n-state quasi sweeping 2nfa can be simulated by a 2dfa with $O(n^{\lceil \log_2(n+1) \rceil + 3})$ states.

We consider the following predicate *reachable*:

reachable(q, p, k) is true, for $q, p \in Q$, $k \geq 1$, iff there exists a computation path of the given 2nfa which

- starts and ends with the input head scanning the left endmarker,
- in the state q and p, respectively, and
- visits that endmarker at most $k - 1$ times.

Second lemma: outline of the proof

Lemma (from quasi sweeping 2nfa to 2dfa)

Each unary n-state quasi sweeping 2nfa can be simulated by a 2dfa with $O(n^{\lceil \log_2(n+1) \rceil + 3})$ states.

We consider the following predicate *reachable*:

reachable(q, p, k) is true, for $q, p \in Q$, $k \geq 1$, iff there exists a computation path of the given 2nfa which

- starts and ends with the input head scanning the left endmarker,
- in the state q and p, respectively, and
- visits that endmarker at most $k + 1$ times.

Second lemma: outline of the proof

Lemma (from quasi sweeping 2nfa to 2dfa)

Each unary n-state quasi sweeping 2nfa can be simulated by a 2dfa with $O(n^{\lceil \log_2(n+1) \rceil + 3})$ states.

We consider the following predicate *reachable*:

reachable(q, p, k) is true, for $q, p \in Q$, $k \geq 1$, iff there exists a computation path of the given 2nfa which

- starts and ends with the input head scanning the left endmarker,
- in the state q and p, respectively, and
- visits that endmarker at most $k + 1$ times.

Second lemma: outline of the proof

Lemma (from quasi sweeping 2nfa to 2dfa)

Each unary n-state quasi sweeping 2nfa can be simulated by a 2dfa with $O(n^{\lceil \log_2(n+1) \rceil + 3})$ states.

We consider the following predicate *reachable*:

reachable(q, p, k) is true, for $q, p \in Q$, $k \geq 1$, iff there exists a computation path of the given 2nfa which

- starts and ends with the input head scanning the left endmarker,
- in the state q and p, respectively, and
- visits that endmarker at most $k + 1$ times.

Second lemma: outline of the proof

Lemma (from quasi sweeping 2nfa to 2dfa)

Each unary n-state quasi sweeping 2nfa can be simulated by a 2dfa with $O(n^{\lceil \log_2(n+1) \rceil + 3})$ states.

We consider the following predicate *reachable*:

reachable(q, p, k) is true, for $q, p \in Q$, $k \geq 1$, iff there exists a computation path of the given 2nfa which

- starts and ends with the input head scanning the left endmarker,
- in the state q and p, respectively, and
- visits that endmarker at most $k + 1$ times.

Second lemma: outline of the proof

Lemma (from quasi sweeping 2nfa to 2dfa)

Each unary n-state quasi sweeping 2nfa can be simulated by a 2dfa with $O(n^{\lceil \log_2(n+1) \rceil + 3})$ states.

We consider the following predicate *reachable*:

reachable(q, p, k) is true, for $q, p \in Q$, $k \geq 1$, iff there exists a computation path of the given 2nfa which

- starts and ends with the input head scanning the left endmarker,
- in the state q and p, respectively, and
- visits that endmarker at most $k + 1$ times.

Second lemma: outline of the proof

If an accepting computation visits the left endmarker twice in the same state, then there exists a shorter accepting computation.

Hence:

The input is accepted iff $reachable(q_0, q_f, n)$ is true.

Second lemma: outline of the proof

If an accepting computation visits the left endmarker twice in the same state, then there exists a shorter accepting computation.

Hence:

The input is accepted iff $reachable(q_0, q_f, n)$ is true.

Second lemma: outline of the proof

How to evaluate *reachable*?

Divide–and–conquer technique (*reach1* is the base case):

function *reachable*(q, p, k)
if $k = 1$ **then return** *reach1*(q, p)
else begin
  for each state $r \in Q$ **do**
    if *reachable*(q, r, $\lceil k/2 \rceil$) **then**
      if *reachable*(r, p, $\lceil k/2 \rceil$) **then**
        return TRUE

Second lemma: outline of the proof

How to evaluate *reachable*?

Divide–and–conquer technique (*reach1* is the base case):

function *reachable*(q, p, k)
if $k = 1$ **then return** *reach1*(q, p)
else begin
  for each state $r \in Q$ **do**
    if *reachable*(q, r, $\lfloor k/2 \rfloor$) **then**
      if *reachable*(r, p, $\lceil k/2 \rceil$) **then**
        return TRUE
  return FALSE
end

Second lemma: outline of the proof

How to evaluate *reachable*?

Divide–and–conquer technique (*reach1* is the base case):

function *reachable*(q, p, k)
if $k = 1$ **then return** *reach1*(q, p)
else begin
  for each state $r \in Q$ **do**
    if *reachable*(q, r, $\lfloor k/2 \rfloor$) **then**
      if *reachable*(r, p, $\lceil k/2 \rceil$) **then**
        return TRUE
  return FALSE
end

This computation can be implemented by a 2dfa with $O(\text{size}(A) \cdot |Q|^3)$ states.

Second lemma: outline of the proof

How to evaluate *reachable*?

Divide–and–conquer technique (*reach1* is the base case):

function *reachable*(q, p, k)
if $k = 1$ **then return** *reach1*(q, p)
else begin
  for each state $r \in Q$ **do**
    if *reachable*(q, r, $\lfloor k/2 \rfloor$) **then**
      if *reachable*(r, p, $\lceil k/2 \rceil$) **then**
        return TRUE
  return FALSE
end

This computation can be implemented by a 2dfa with $O(n^{\lceil \log_2(n+1) \rceil + 3})$ states.

Second lemma: outline of the proof

How to evaluate *reachable*?

Divide–and–conquer technique (*reach1* is the base case):

function *reachable*(q, p, k)
if $k = 1$ **then return** *reach1*(q, p)
else begin
  for each state $r \in Q$ **do**
    if *reachable*(q, r, $\lfloor k/2 \rfloor$) **then**
      if *reachable*(r, p, $\lceil k/2 \rceil$) **then**
        return TRUE
  return FALSE
end

This computation can be implemented by a 2dfa with $O(n^{\lceil \log_2(n+1) \rceil + 3})$ states.

Second lemma: outline of the proof

How to evaluate *reachable*?

Divide–and–conquer technique (*reach1* is the base case):

function *reachable*(q, p, k)
if $k = 1$ **then return** *reach1*(q, p)
else begin
  for each state $r \in Q$ **do**
    if *reachable*(q, r, $\lfloor k/2 \rfloor$) **then**
      if *reachable*(r, p, $\lceil k/2 \rceil$) **then**
        return TRUE
  return FALSE
end

This computation can be implemented by a 2dfa with $O(n^{\lceil \log_2(n+1) \rceil + 3})$ states.

Sakoda&Sipser question

Current knowledge

- Upper bounds

	1nfa $\longrightarrow$ 2dfa	2nfa $\longrightarrow$ 2dfa
unary case	$O(n^2)$ [1] optimal	$n^{O(\log n)}$ [2]
general case	exponential	exponential

[1: Chrobak 1986]
[2: Geffert, Mereghetti, Pighizzini 2003]

- Lower bounds
  For all the cases, the best known lower bound is $\Omega(n^2)$ [1]

Sakoda&Sipser question

Current knowledge

- Upper bounds

	1nfa $\rightarrow$ 2dfa	2nfa $\rightarrow$ 2dfa
unary case	$O(n^2)$ [1] optimal	$n^{O(\log n)}$ [2]
general case	exponential	exponential

[1: Chrobak 1986]

[2: Geffert, Mereghetti, Pighizzini 2003]

- Lower bounds

  For all the cases, the best known lower bound is $\Omega(n^2)$ [1]

Sakoda&Sipser question

Current knowledge

- Upper bounds

	1nfa $\rightarrow$ 2dfa	2nfa $\rightarrow$ 2dfa
unary case	$O(n^2)$ [1] optimal	$n^{O(\log n)}$ [2]
general case	exponential	exponential

[1: Chrobak 1986]

[2: Geffert, Mereghetti, Pighizzini 2003]

- Lower bounds

  For all the cases, the best known lower bound is $\Omega(n^2)$ [1]

Complementation of two-way automata

Problem

Given a two-way automaton with n states accepting a language L, find the cost in term of states, of an automaton accepting the complement of L.

- Deterministic case:
  The cost is 4^n, for *any input alphabet*.
  [Geffert, Mereghetti, Pighizzini 2007]
- Nondeterministic case:
  The cost is polynomial for a *unary alphabet*.
  [Geffert, Mereghetti, Pighizzini 2007]

What about the complementation of 2nfa over *nonunary* alphabets?

Complementation of two-way automata

Problem

Given a two-way automaton with n states accepting a language L, find the cost in term of states, of an automaton accepting the complement of L.

- Deterministic case:
  The cost is $4n$, for *any input alphabet.*
  [Geffert, Mereghetti, Pighizzini 2007]
- Nondeterministic case:
  The cost is polynomial for a *unary alphabet.*
  [Geffert, Mereghetti, Pighizzini 2007]

What about the complementation of 2nfa over *nonunary* alphabets?

Complementation of two-way automata

Problem

Given a two-way automaton with n states accepting a language L, find the cost in term of states, of an automaton accepting the complement of L.

- Deterministic case:
  The cost is $4n$, for *any input alphabet.*
  [Geffert, Mereghetti, Pighizzini 2007]
- Nondeterministic case:
  The cost is polynomial for a *unary alphabet.*
  [Geffert, Mereghetti, Pighizzini 2007]

What about the complementation of 2nfa over *nonunary* alphabets?

Complementation of two-way automata

Problem

Given a two-way automaton with n states accepting a language L, find the cost in term of states, of an automaton accepting the complement of L.

- Deterministic case:
  The cost is $4n$, for *any input alphabet.*
  [Geffert, Mereghetti, Pighizzini 2007]
- Nondeterministic case:
  The cost is polynomial for a *unary alphabet.*
  [Geffert, Mereghetti, Pighizzini 2007]

What about the complementation of 2nfa over *nonunary* alphabets?

Complementation of two-way automata

Relationship with the Sakoda&Sipser question

$f(n) :=$ the cost of the simulation of a n-state 2nfa by a 2dfa.

Given an n-state 2nfa accepting L we can find:

- a 2dfa accepting L with $f(n)$ states

Complementation of two-way automata

Relationship with the Sakoda&Sipser question

$f(n) :=$ the cost of the simulation of a n-state 2nfa by a 2dfa.

Given an n-state 2nfa accepting L we can find:

- a 2dfa accepting L with $f(n)$ states
- a 2nfa (actually a 2dfa) accepting L^R with $4f(n)$ states

Complementation of two-way automata

Relationship with the Sakoda&Sipser question

$f(n) :=$ the cost of the simulation of a n-state 2nfa by a 2dfa.

Given an n-state 2nfa accepting L we can find:

- a 2dfa accepting L with $f(n)$ states
- a 2nfa (actually a 2dfa) accepting L^c with $4f(n)$ states

Hence:

the complementation of 2nfa costs no more than their determinization.

Moreover:

If the complementation of 2nfa requires an exponential number of states, then the gap between 2nfa and 2dfa is exponential.

Complementation of two-way automata

Relationship with the Sakoda&Sipser question

$f(n) :=$ the cost of the simulation of a n-state 2nfa by a 2dfa.

Given an n-state 2nfa accepting L we can find:

- a 2dfa accepting L with $f(n)$ states
- a 2nfa (actually a 2dfa) accepting L^c with $4f(n)$ states

Hence:
the complementation of 2nfa costs no more than their determinization.

Theorem

If the complementation of 2nfa requires an exponential number of states then the gap between 2nfa and 2dfa is exponential.

Complementation of two-way automata

Relationship with the Sakoda&Sipser question

$f(n) :=$ the cost of the simulation of a n-state 2nfa by a 2dfa.

Given an n-state 2nfa accepting L we can find:

- a 2dfa accepting L with $f(n)$ states
- a 2nfa (actually a 2dfa) accepting L^c with $4f(n)$ states

Hence:
the complementation of 2nfa costs no more than their determinization.

Theorem

If the complementation of 2nfa requires an exponential number of states then the gap between 2nfa and 2dfa is exponential.

Complementation of two-way automata

Relationship with the Sakoda&Sipser question

$f(n) :=$ the cost of the simulation of a n-state 2nfa by a 2dfa.

Given an n-state 2nfa accepting L we can find:

- a 2dfa accepting L with $f(n)$ states
- a 2nfa (actually a 2dfa) accepting L^c with $4f(n)$ states

Hence:
the complementation of 2nfa costs no more than their determinization.

Theorem

If the complementation of 2nfa requires an exponential number of states then the gap between 2nfa and 2dfa is exponential.