# Determinism versus Nondeterminism in Two-Way Finite Automata

## Recent Results around the Sakoda and Sipser Question

Giovanni Pighizzini

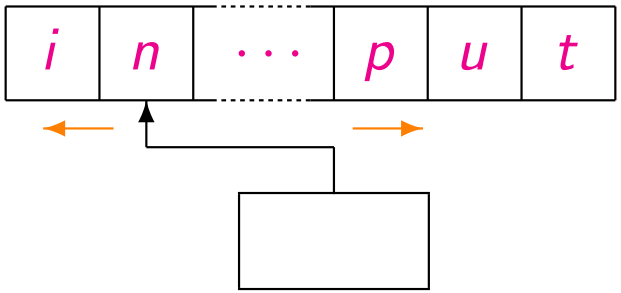
Dipartimento di Informatica
Università degli Studi di Milano

NCMA 2012
Fribourg, Switzerland
August 23-24, 2012

UNIVERSITÀ DEGLI STUDI
DI MILANO

# Outline

# Finite State Automata




Base version:

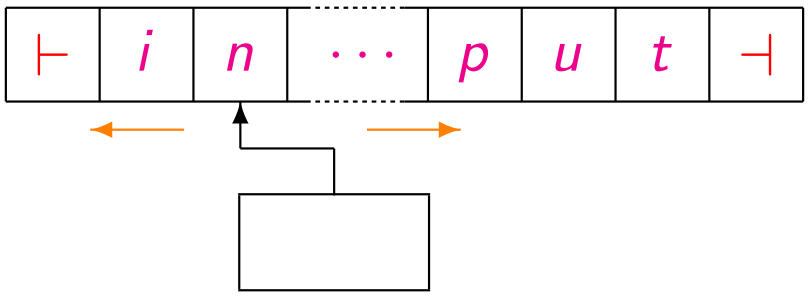
one-way deterministic finite automata (1DFA)

- one-way input tape
- deterministic transitions

Possibile variants allowing:

- nondeterministic transitions
  - one-way nondeterministic finite automata (1NFA)
- input head moving forth and back
  - two-way deterministic finite automata (2DFA)
  - two-way nondeterministic finite automata (2NFA)
- alternation
- ...

# Two-Way Automata: Technical Details




- Input surrounded by the endmarkers $\vdash$ and $\dashv$
- $w \in \Sigma^*$ is accepted iff there is a computation
  - with input tape $\vdash w \dashv$
  - starting at the left endmarker $\vdash$ in the initial state
  - reaching a final state

# 1DFA, 1NFA, 2DFA, 2NFA

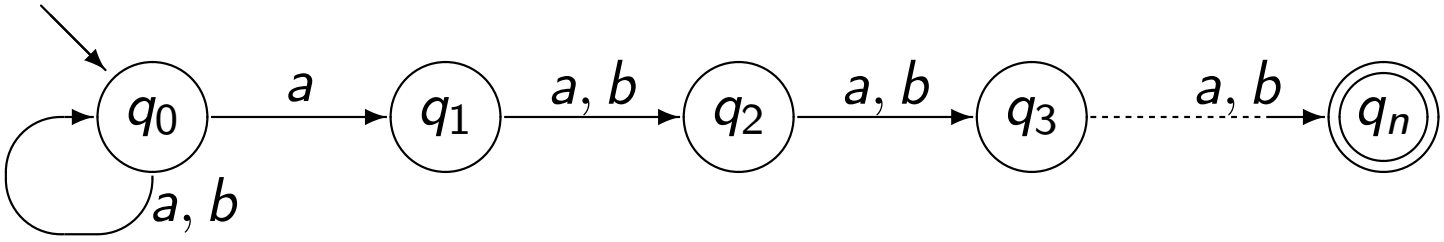
What about the power of these models?

They share the same computational power, namely they characterize the class of *regular languages*, however...

...some of them are more succinct

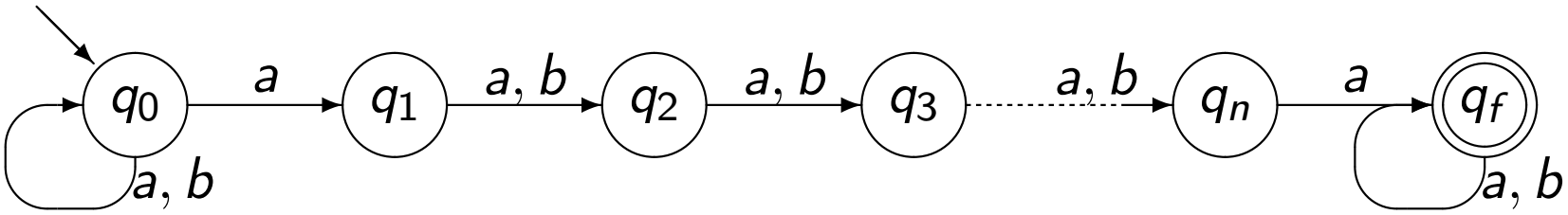
# Example: $L_n = (a+b)^* a (a+b)^{n-1}$

- $L_n$ is accepted by a 1NFA with $n+1$ states



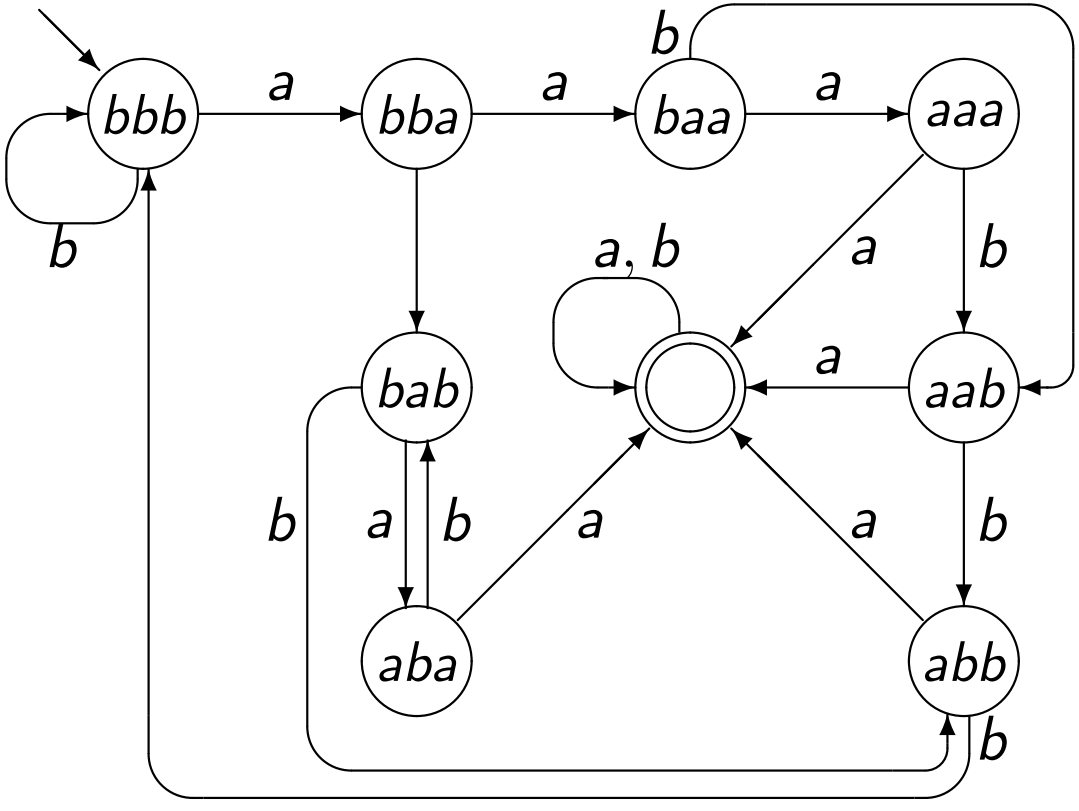

- The minimum 1DFA accepting $L_n$ requires $2^n$ states
- We can get a *deterministic* automaton for $L_n$ with $n+2$ states, which reverses the input head direction just one time
- Hence $L_n$ is accepted by
  - a 1NFA and a 2DFA with approx. the same number of states
  - a minimum 1DFA exponentially larger

# Example: $L_n = (a+b)^* a (a+b)^{n-1} a (a+b)^*$




1NFA: $n + 2$ states

# Example: $L_n = (a+b)^* a (a+b)^{n-1} a (a+b)^*$




$n = 3$

Minimum 1DFA: $2^n + 1$ states

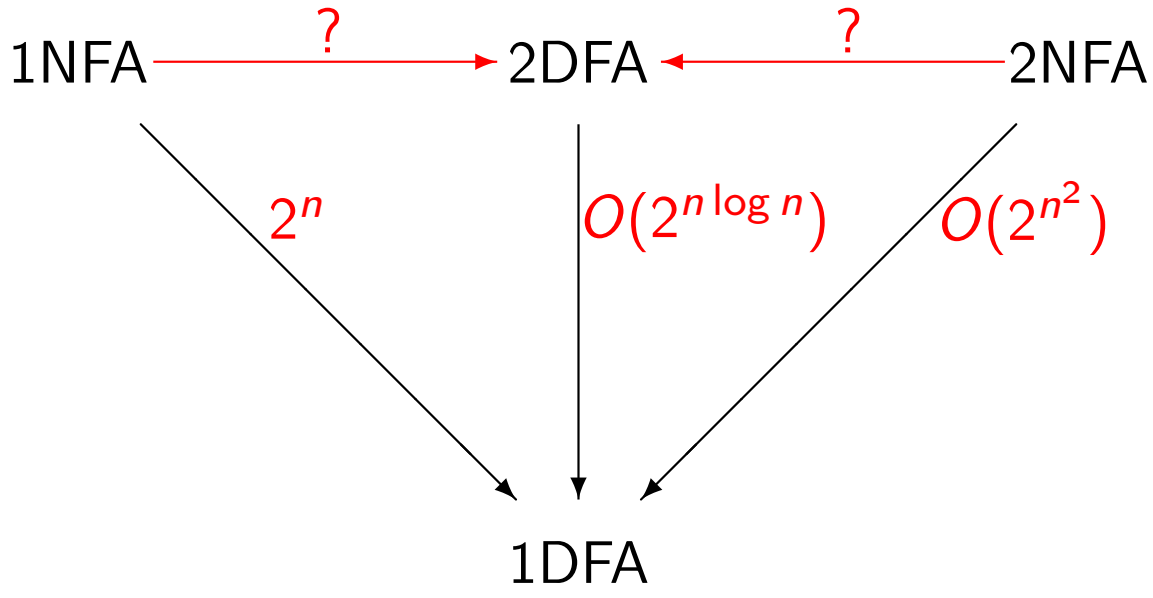
# Example: $L_n = (a+b)^* a (a+b)^{n-1} a (a+b)^*$


$n = 4$

**while** input symbol $\neq a$ **do** move to the right
move $n$ squares to the right
**if** input symbol $= a$ **then accept**
**else** move $n-1$ cells to the left
**repeat** from the first step
*Exception:* **if** input symbol $=\dashv$ **then reject**

2DFA: $O(n)$ states

# Costs of the Optimal Simulations Between Automata

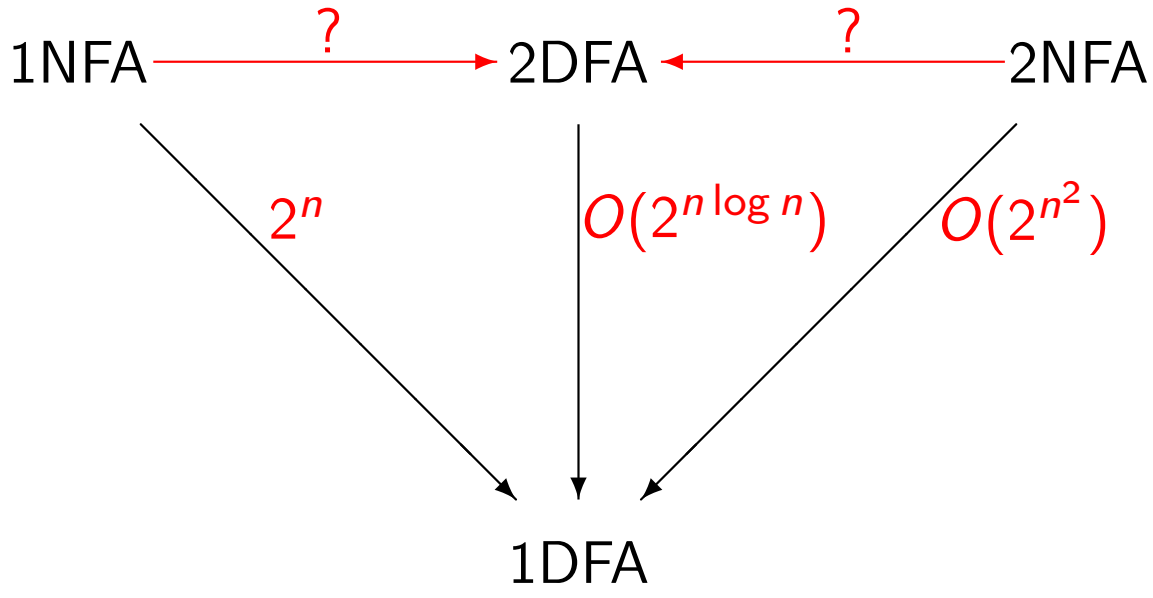



[Rabin&Scott '59, Shepardson '59, Meyer&Fischer '71, . . . ]

## Question

*How much the possibility of moving the input head forth and back is useful to eliminate the nondeterminism?*

# Costs of the Optimal Simulations Between Automata




## Problem ([Sakoda&Sipser '78])

*Do there exist polynomial simulations of*

- *1NFAs by 2DFAs*
- *2NFAs by 2DFAs ?*

## Conjecture

*These simulations are not polynomial*

# Sakoda&Sipser Question: Upper and Lower Bounds

- Exponential upper bounds
  deriving from the simulations of 1NFAs and 2NFAs by 1DFAs

- Polynomial lower bounds
  for the cost $c(n)$ of simulation of 1NFAs by 2DFAs:
  - $c(n) \in \Omega(\frac{n^2}{\log n})$ [Berman&Lingas '77]
  - $c(n) \in \Omega(n^2)$ [Chrobak '86]

- Complete languages
  ...

# Sakoda and Sipser Question

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

  Lower and upper bounds too far
  (Polynomial vs exponential)

- Hence:

  Try to attack restricted versions of the problem!

# 2NFAs vs 2DFAs: Restricted Versions

(i) Restrictions on the resulting machines (2DFAs)

- sweeping automata [Sipser '80]
- oblivious automata [Hromkovič&Schnitger '03]
- "few reversal" automata [Kapoutsis '11]

(ii) Restrictions on the languages

- unary regular languages [Geffert Mereghetti&P '03]

(iii) Restrictions on the starting machines (2NFAs)

- outer nondeterministic automata [Guillon Geffert&P '12]

# Sweeping Automata

## Definition (Sweeping Automata)

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

- $A$ is deterministic
- the input head of $A$ can change direction only at the endmarkers

Each computation is a sequence of complete traversals of the input

- Sweeping automata can be exponentially larger than 1NFAs [Sipser '80]
- However, they can be also *exponentially larger* than 2DFAs [Berman '81, Micali '81]

# Oblivious Automata

## Definition

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

- $A$ is deterministic, and
- for each integer $n$, the "trajectory" of the input head is the same for all inputs of length $n$

Each sweeping automaton can be made oblivious with at most a quadratic growth of the number of the states

# Oblivious Automata

- Oblivious automata can be exponentially larger than 2NFAs
  [Hromkovič&Schnitger '03]
- Oblivious automata can be exponentially smaller than sweeping automata:
  - $L_k = (\{uv \mid u, v \in \{a, b\}^k \text{ and } u \neq v\}\#)^*$
  - $L_k$ is accepted by an oblivious automaton with $O(k)$ states
    [Kutrib Malcher&P '12]
  - each sweeping automaton for $L_k$ requires at least $2^{\frac{k-1}{2}}$ states
    [Hromkovič&Schnitger '03]
- Oblivious automata can be exponentially larger than 2DFAs
  - Witness: $PAD(L_k) = \bigcup_{a_1 a_2 \ldots a_m \in L_k} \$^* a_1 \$^* a_2 \$^* \cdots \$^* a_m \$^*$
    [Kutrib Malcher&P '12]

# "Few Reversal" Automata [Kapoutsis'11]

## Definition (Few Reversal Automata)

A two-way automaton $A$ makes few reversals if and only if the number of reversals on input of length $n$ is $o(n)$

Model between sweeping automata ($O(1)$ reversals) and 2NFAs

## Theorem ([Kapoutsis'11])

- *Few reversal DFAs can be exponentially larger than few reversal NFAs and, hence, than 2NFAs*
- *Sweeping automata can be exponentially larger than few reversal DFAs*
- *Few reversal DFAs can be exponentially larger than 2DFAs*

Hence, this result really extends Sipser's separation, but does not solve the full problem

# Sakoda&Sipser Question

**Problem ([Sakoda&Sipser '78])**

*Do there exist polynomial simulations of*

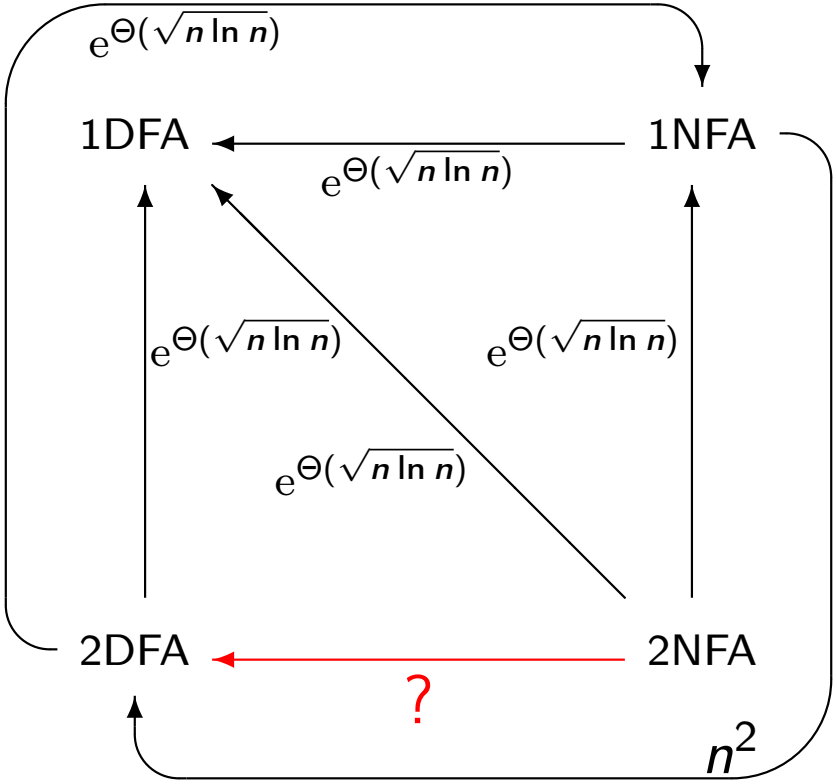
- *1NFAs by 2DFAs*
- *2NFAs by 2DFAs ?*

Another possible restriction:

The unary case $\#\Sigma = 1$

# Optimal Simulation Between Unary Automata

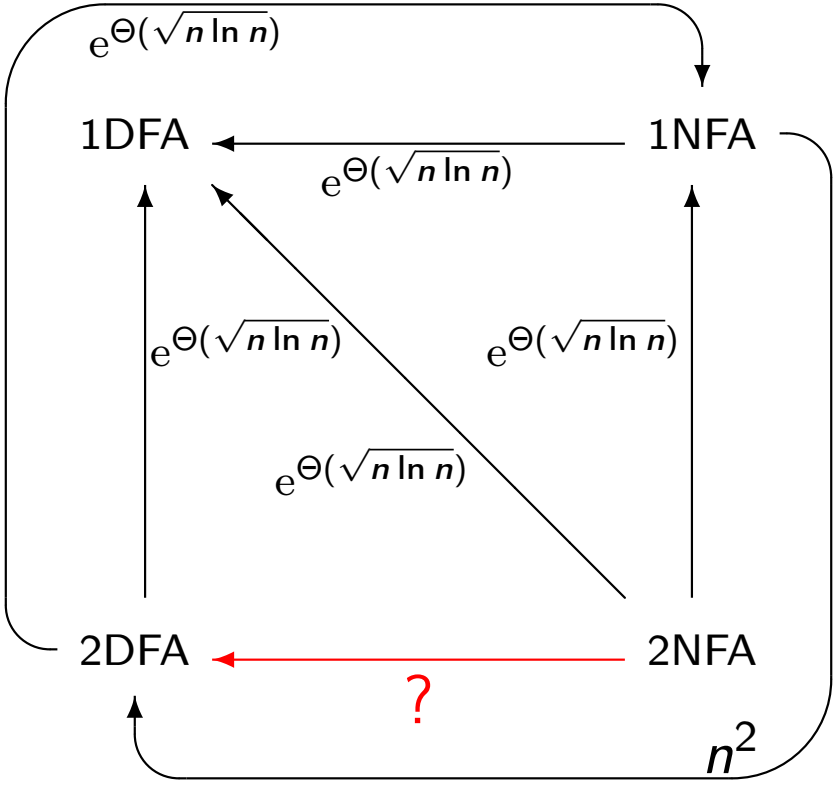
The costs of the optimal simulations between automata are different in the unary and in the general case




[Chrobak '86, Mereghetti&P '01]

# Optimal Simulation Between Unary Automata

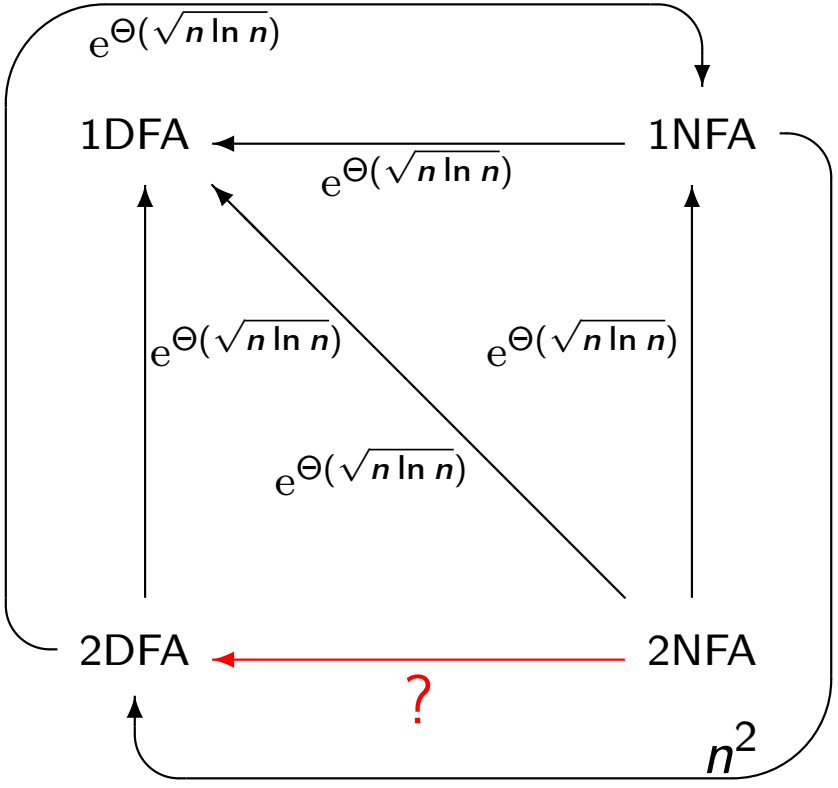
The costs of the optimal simulations between automata are different in the unary and in the general case




1NFA $\rightarrow$ 2DFA
In the unary case
this question is solved!
(polynomial conversion)

# Optimal Simulation Between Unary Automata

The costs of the optimal simulations between automata are different in the unary and in the general case




2NFA $\to$ 2DFA

*Even* in the unary case this question is open!

- $e^{\Theta(\sqrt{n \ln n})}$ upper bound (from 2NFA $\to$ 1DFA)
- $\Omega(n^2)$ lower bound (from 1NFA $\to$ 2DFA)

A better upper bound $e^{O(\ln^2 n)}$ has been proved!

# Sakoda&Sipser Question: Current Knowledge

- Upper bounds

| | 1NFA$\rightarrow$ 2DFA | 2NFA$\rightarrow$ 2DFA |
|---|---|---|
| unary case | $O(n^2)$ optimal | $e^{O(\ln^2 n)}$ |
| general case | exponential | exponential |

Unary case [Chrobak '86, Geffert Mereghetti&P '03]

- Lower Bounds

  In all the cases, the best known lower bound is $\Omega(n^2)$ [Chrobak '86]

# Unary Case: Quasi Sweeping Automata

[Geffert Mereghetti&P '03]

In the study of unary 2NFA, sweeping automata with some *restricted nondeterministic capabilities* turn out to be very useful:

## Definition

A 2NFA is quasi sweeping (qsNFA) iff both

- nondeterministic choices and head reversals

are possible only at the endmarkers

## Theorem (Quasi Sweeping Simulation)

*Each n-state unary 2NFA A can be transformed into a 2NFA M s.t.*

- *M is quasi sweeping*
- *M has at most* $N \leq 2n + 2$ *states*
- *M and A are "almost equivalent"*
  *(differences are possible only for inputs of length* $\leq 5n^2$*)*

# Quasi Sweeping Simulation: Consequences

Several results using quasi sweeping simulation of unary 2NFAs have been found:

(i) Subexponential simulation of unary 2NFAs by 2DFAs
Each unary $n$-state 2NFA can be simulated by a 2DFA with $e^{O(\ln^2 n)}$ states [Geffert Mereghetti&P '03]

(ii) Polynomial complementation of unary 2NFAs
Inductive counting argument for qsNFAs
[Geffert Mereghetti&P '07]

(iii) Polynomial simulation of unary 2NFAs by 2DFAs
*under the condition* L = NL [Geffert&P '10]

(iv) Polynomial simulation of unary 2NFAs by unambiguous 2NFAs (unconditional) [Geffert&P '10]

We are going to discuss (iii)

# Logspace Classes and Graph Accessibility Problem

L: class of languages accepted in logarithmic space by *deterministic* machines

NL: class of languages accepted in logarithmic space by *nondeterministic* machines

**Problem**

$\mathsf{L} \stackrel{?}{=} \mathsf{NL}$

*Graph Accessibility Problem* GAP

- Given $G = (V, E)$ oriented graph, $s, t \in V$
- Decide whether or not $G$ contains a path from $s$ to $t$

**Theorem ([Jones '75])**

GAP *is complete for* NL *(under logspace reductions)*

$\Rightarrow$ GAP $\in$ L iff L = NL

More in general, GAP $\in \mathcal{C}$ implies $\mathcal{C} \supseteq$ NL for each class $\mathcal{C}$ closed under logspace reductions

# Polynomial Deterministic Simulation (under L = NL)

## Outline

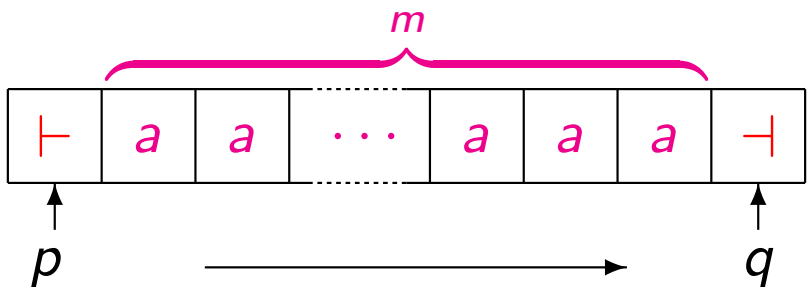
- Let $A$ be an $n$-state unary 2NFA
- Reduction from $L(A)$ to GAP

  i.e, from each string $a^m$ we compute a graph $G(m)$ s.t.

$$a^m \in L(A) \iff G(m) \in \text{GAP}$$

- *Under the hypothesis* L = NL
  this reduction is used to build a 2DFA equivalent to $A$,
  with a number of states polynomial in $n$
- Actually we do not work directly with $A$:
  we use the qsNFA $M$ obtained from $A$
  according to the quasi sweeping simulation

# The Graph $G(m)$




Given the qsNFA $M$ with $N$ states and an input $a^m$
the graph $G(m)$ is defined as:

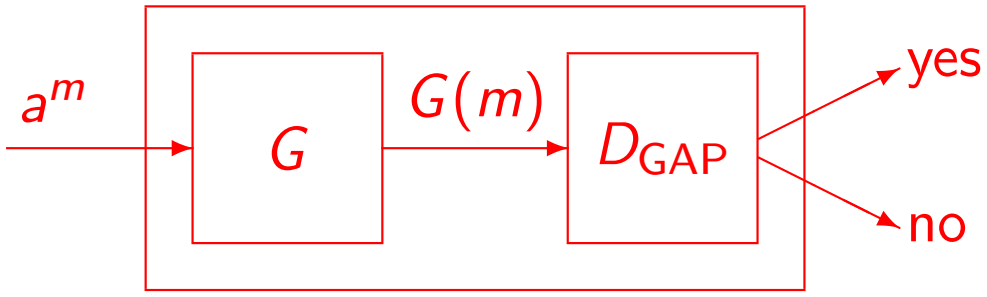
- the vertices are the states of $M$
- $(p, q)$ is an edge iff $M$ can traverse the input
  - from one endmarker in the state $p$
  - to the opposite endmarker in the state $q$
  - without visiting the endmarkers in the meantime

Then

$$a^m \in \mathrm{L(M)} \text{ iff } G(m) \text{ contains a path from } q_0 \text{ to } q_F$$

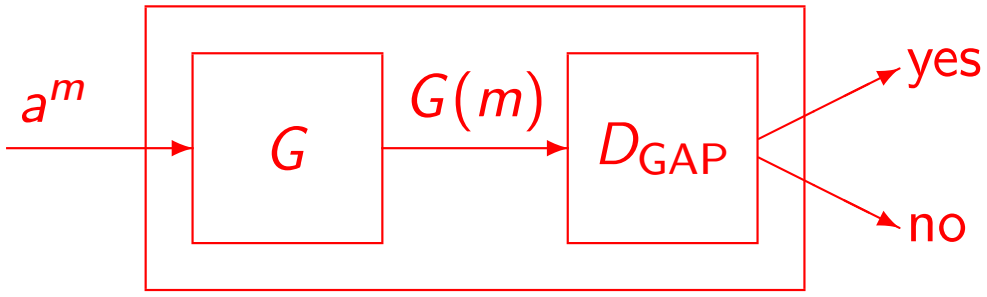
The existence of the edge $(p, q)$ can be verified by a subroutine, implemented by a finite automaton $A_{p,q}$ with $N$ states

# Deterministic Simulation




- Suppose L = NL
- Let $D_{\text{GAP}}$ be a logspace bounded *deterministic* machine solving GAP
- On input $a^m$, compute $G(m)$ and give the resulting graph as input to $D_{\text{GAP}}$
- This decides whether or not $a^m \in L(M)$

# Deterministic Simulation




- The graph $G(m)$ has $N$ vertices, the number of states of $M$
- $D_{\mathrm{GAP}}$ uses space $O(\log N)$
- $M$ is fixed. Hence $N$ is constant, independent on the input $a^m$

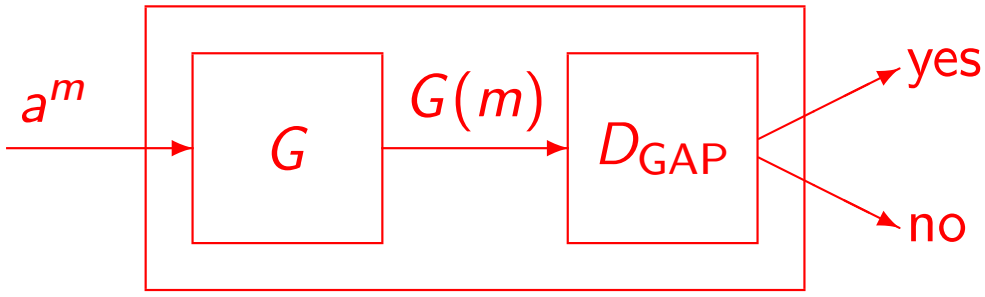
  *The worktape of $D_{\mathrm{GAP}}$ can be encoded in a finite control using a number of states polynomial in $N$*

- The graph $G(m)$ can be represented with $N^2$ bits

  *Representing the graph in a finite control would require exponentially many states*

- To avoid this we compute input bits for $D_{\mathrm{GAP}}$ "on fly"

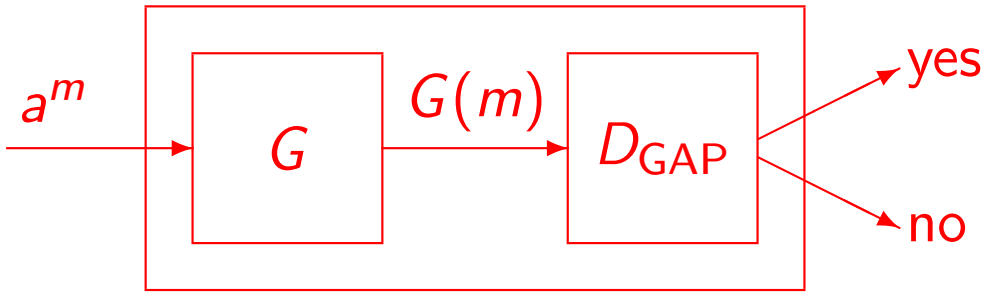
# Deterministic Simulation




We define a unary 2DFA $M'$ equivalent to $M$

- $M'$ keeps in its finite control:
  - The input head position of $D_{\mathrm{GAP}}$
  - The worktape content of $D_{\mathrm{GAP}}$
  - The finite control of $D_{\mathrm{GAP}}$
- This uses a number of states polynomial in $N$

# Deterministic Simulation




We define a unary 2DFA $M'$ equivalent to $M$

- On input $a^m$, $M'$ simulates $D_{\mathrm{GAP}}$ on input $G(m)$
- Input bits for $D_{\mathrm{GAP}}$ are the entries of $G(m)$ adjacency matrix
- Each time $D_{\mathrm{GAP}}$ needs an input bit, a subroutine $A_{p,q}$ is called
- Each $A_{p,q}$ uses no more than $N$ states
- Considering all possible $(p, q)$, this part uses at most $N^3$ states

# Summing Up... (under L = NL)

We described the following simulation:

- $M$ is *almost equivalent* to the original 2NFA $A$
- Hence, $M'$ is *almost equivalent* to $A$
- Possible differences for input length $\leq 5n^2$
- They can be fixed in a preliminary scan ($5n^2 + 2$ more states)
- The resulting automaton has polynomially many states

| | | |
|---|---|---|
| $A$ | given unary 2NFA | $n$ states |
| $\Downarrow$ | | Conversion into Normal Form |
| $M$ | almost equivalent to $A$ | $N \leq 2n + 2$ states |
| $\Downarrow$ | | Deterministic Simulation |
| $M'$ | 2DFA equivalent to $M$ | $poly(N)$ states |
| $\Downarrow$ | Preliminary scan to accept/reject inputs of length $\leq 5n^2$ then simulation of $M'$ for longer inputs | |
| $M''$ | 2DFA equivalent to $A$ | $poly(n)$ states |

# Polynomial Deterministic Simulation (under L = NL)

**Theorem ([Geffert&P '10])**

*If* L = NL *then each n-state unary 2NFA can be simulated by an equivalent 2DFA with poly(n) many states*

*Hence, proving the Sakoda&Sipser conjecture for unary 2NFAs would separate* L *and* NL

What about the converse?

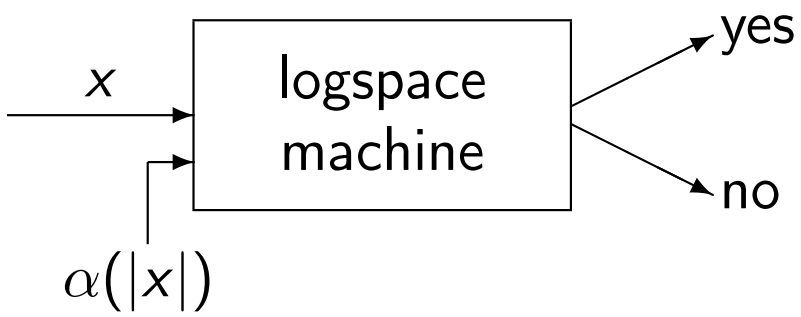
It has been proved under the following uniformity assumption:

*The transformation from unary 2NFAs to 2DFAs must be computable in deterministic logspace* [Geffert&P '10]

Uniformity?

# Nonuniform Deterministic Logspace

- L/poly
  class of languages accepted by deterministic logspace machines with a *polynomial advice*




**Problem**

L/poly $\supseteq$ NL *?*

# Polynomial Deterministic Simulation (under L = NL)

We did not used the uniformity of L !

- L can be replaced by L/poly:

  *If* L/poly $\supseteq$ NL *then each n-state unary 2NFA can be simulated by an equivalent 2DFA with poly*$(n)$ *many states*

- We can prove the converse using GAP:

  *If the simulation of unary 2NFAs by 2DFAs is polynomial in states then there is a deterministic logspace machine with a polynomial advice which solves* GAP

# Solving GAP with Two-Way Automata

## Binary Encoding: Languages BGAP

- Let $n$ be a fixed integer
- $\mathrm{GAP}_n$ denotes GAP restricted to graphs with vertex set $V_n = \{0, \ldots, n-1\}$
- The *binary encoding* of a graph $G = (V_n, E)$ is the standard encoding of its adjacency matrix, i.e., a string

  $$\langle G \rangle_2 = x_1 x_2 \cdots x_{n^2} \in \{0,1\}^{n^2}$$

  with $x_{i \cdot n + j + 1} = 1$ if and only if $(i, j) \in E$
- $\mathrm{BGAP}_n := \{\langle G \rangle_2 \mid G \text{ has a path from 0 to } n-1\}$
  $= \{\langle G \rangle_2 \mid G \in \mathrm{GAP}_n\}$

# Solving GAP with Two-Way Automata

## Recognizing $\mathrm{BGAP}_n$

Standard nondeterministic algorithm solving graph accessibility

$i \leftarrow 0$ // input head on the left endmarker
**while** $i \neq n-1$ **do**
  guess $j \neq i$ // try the edge $(i,j)$
  move to the input cell $i \cdot n + j + 1$
  **if** the input symbol is 0 **then reject** // $(i,j) \notin E$
  move the input head to the left endmarker
  $i \leftarrow j$
**endwhile**
**accept**

- Implementation using $O(n^3)$ states

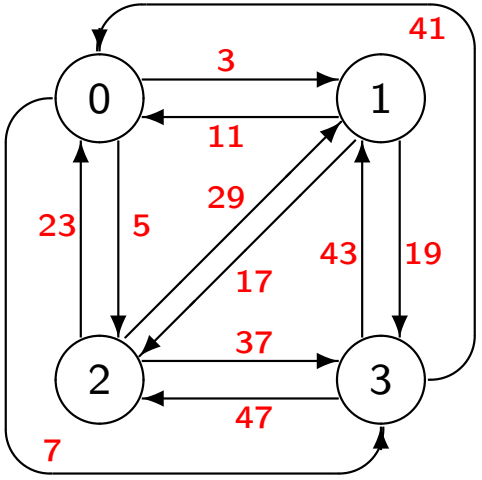
# Solving GAP with Two-Way Automata

## Unary Encoding: Languages UGAP

- $K_n :=$ complete directed graph with vertex set $V_n = \{0, \ldots, n-1\}$
- With each edge $(i, j)$ we associate a different prime number $p_{(i,j)}$
- A subgraph $G = (V_n, E)$ of $K_n$ is encoded by the string $a^{m_G}$, where

$$m_G = \prod_{(i,j) \in E} p_{(i,j)}$$

- Graph $K_n(m)$: $\exists$ edge $(i, j)$ iff $p_{(i,j)}$ divides $m$
- $\text{UGAP}_n := \{a^m \mid K_n(m) \text{ has a path from 0 to } n-1\}$

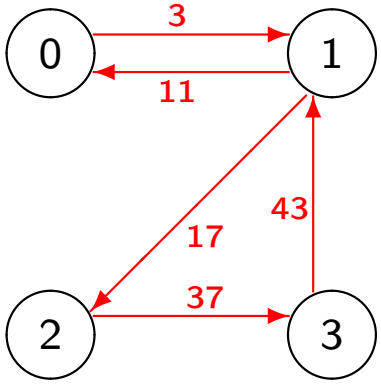


# Solving GAP with Two-Way Automata

## Unary Encoding: Languages UGAP

- $K_n :=$ complete directed graph with vertex set $V_n = \{0, \ldots, n-1\}$
- With each edge $(i, j)$ we associate a different prime number $p_{(i,j)}$
- A subgraph $G = (V_n, E)$ of $K_n$ is encoded by the string $a^{m_G}$, where

$$m_G = \prod_{(i,j) \in E} p_{(i,j)}$$




$$m_G = 3 \cdot 11 \cdot 17 \cdot 37 \cdot 43 = 892551$$

- Graph $K_n(m)$: $\exists$ edge $(i, j)$ iff $p_{(i,j)}$ divides $m$
- $\text{UGAP}_n := \{a^m \mid K_n(m) \text{ has a path from 0 to } n-1\}$

# Solving GAP with Two-Way Automata

## Recognizing $\mathrm{UGAP}_n$

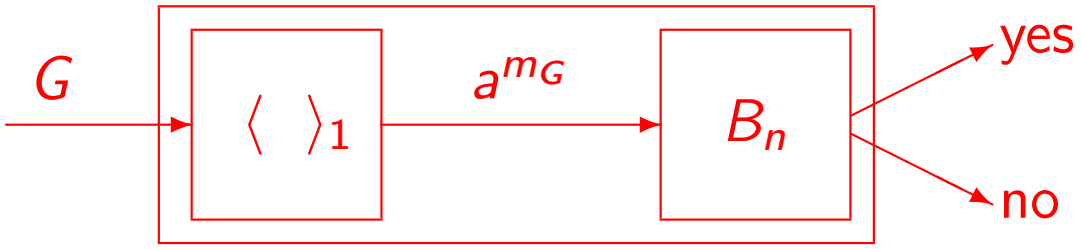
Unary version of the algorithm for $\mathrm{BGAP}_n$

$i \leftarrow 0$ // input head on the left endmarker
**while** $i \neq n-1$ **do**
  guess $j \neq i$ // try the edge $(i,j)$
  scan the input string counting modulo $p_{(i,j)}$
  **if** reminder $\neq 0$ **then reject** // $(i,j) \notin E$
  move the input head to the left endmarker
  $i \leftarrow j$
**endwhile**
**accept**

- Implementation using $O(n^4 \log n)$ states

# Solving GAP with Two-Way Automata

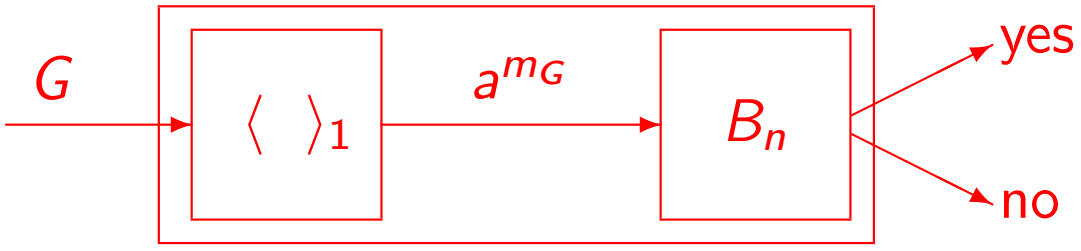
## Outline of the Construction




- Suppose the conversion of unary 2NFAs into 2DFAs is polynomial
- Let $B_n$ be a 2DFA with $poly(n)$ states recognizing $\mathrm{UGAP}_n$
- Given a graph $G = (V_n, E)$, compute its unary encoding $a^{m_G}$ and give it as input to $B_n$
- This decides whether or not $G \in \mathrm{GAP}$

# Solving GAP with Two-Way Automata

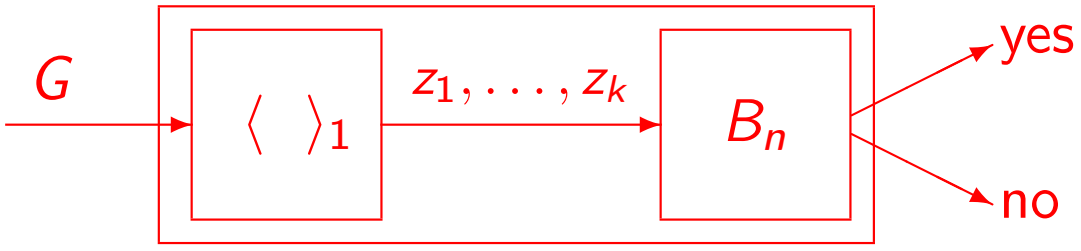
## Outline of the Construction




- Our goal:
  - a deterministic machine
  - working in logarithmic space
  - using a polynomial advice
- The input is the graph $G$ (size $n^2$)
- $B_n$ *is the advice:* polynomial size in $n$
- Representing $a^{m_G}$ would require too much space!

# Solving GAP with Two-Way Automata

## Outline of the Construction



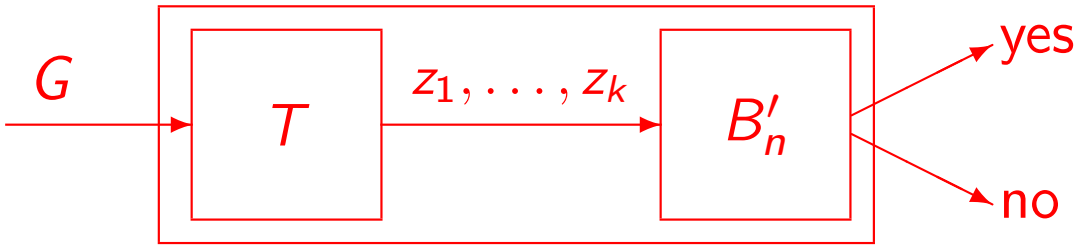

*Prime encoding*:

A list of prime powers $z_1, \ldots, z_k$ factorizing $m_G$

$a^{m_G}$ is replaced by the prime encoding

# Solving GAP with Two-Way Automata

## Replacing Unary Encodings by Prime Encodings




- $m_G = \prod_{(i,j)\in E} p_{(i,j)}$
- *Prime encoding of* $a^{m_G}$ *:*
  list of all $p_{(i,j)}$ associated with the edges of $G$
- It can be computed by in logarithmic space
  by *a deterministic transducer* $T$
- We replace $B_n$ by an "equivalent" 2DFA $B'_n$:

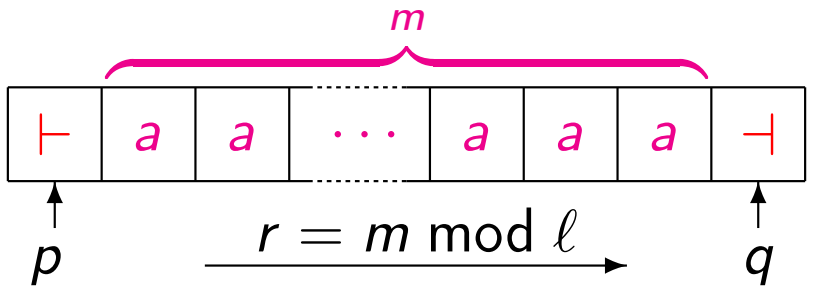
  $B'_n$ inputs represent prime encodings of $B_n$ inputs

# How to Obtain $B'_n$?

- $s :=$ number of states of $B_n$
- $B_n \to M_n$
  - $M_n$ sweeping, $O(s)$ states
  - in each traversal $M_n$ counts the input length modulo a number $\ell$
  - $M_n$ and $B_n$ almost equivalent (differences for length $O(s)$)
- $M_n \to B'_n$
  - $poly(s)$ many states
  - $B'_n$ reads the prime encoding of an integer $m$
  - If $m$ is "small" then $B'_n$ gives the output according to a finite table
  - otherwise, $B'_n$ on its input simulates $M_n$ on $a^m$

# How to Obtain $B'_n$?

## Simulation on Long Inputs

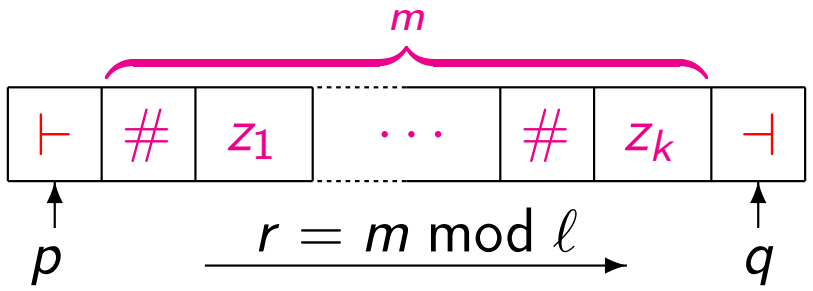



In a sweep:

- $M_n$ counts the input length modulo an integer $\ell$
- The value of $\ell$ depends only on the starting state $p$
- The ending state $q$ depends on $p$ and on $r = m \bmod \ell$

# How to Obtain $B'_n$?

## Simulation on Long Inputs




In a sweep:

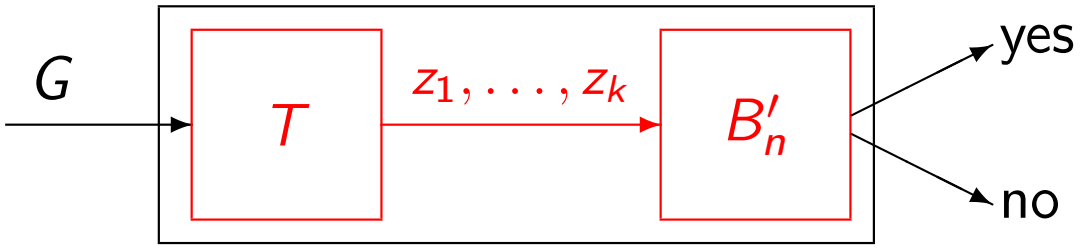
- $M_n$ counts the input length modulo an integer $\ell$
- The value of $\ell$ depends only on the starting state $p$
- The ending state $q$ depends on $p$ and on $r = m \bmod \ell$

$B'_n$ simulates the same sweep on input $z_1, z_2, \ldots, z_k$,
a prime encoding of $m$:

$$m \bmod \ell = ((\cdots((z_1 \bmod \ell) \cdot z_2) \bmod \ell \cdots) \cdot z_k) \bmod \ell$$

# Solving GAP with Two-Way Automata

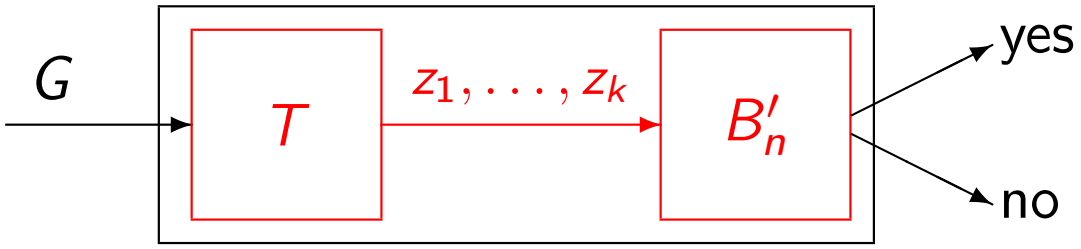
## Combining All Together




- We replace:
  - The machine which computes $m_G = \langle G \rangle_1$ by a logspace transducer $T$ which outputs a prime encoding of $m_G$
  - The unary 2DFA $B_n$ by an "equivalent" 2DFA $B'_n$ working on prime encodings
- The resulting machine still decides whether $G \in \mathrm{GAP}_n$
- The symbols of $z_1, \ldots, z_k$ are computed "on fly", by restarting $T$ each time $B'_n$ needs them

# Solving GAP with Two-Way Automata

## Combining All Together




- $B'_n$ has number of states polyomial in $n$
- $T$ works in space $O(\log n)$
- Hence the resulting machine works in logarithmic space

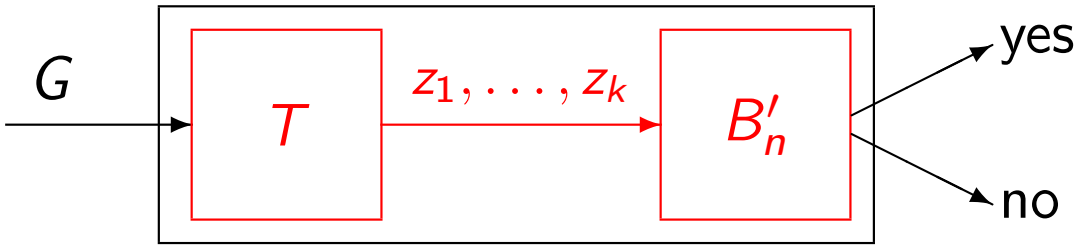
We did not provided $B'_n$ in a constructive way!

- Its existence follows from the hypothesis that the simulation of unary 2NFAs by 2DFAs is polynomial
- Hence the resulting machine is nonuniform

$B'_n$ is the advice!

# Solving GAP with Two-Way Automata

## Combining All Together




Since GAP is complete for NL we obtain:

**Theorem ([Kapoutsis&P '12])**

*If each n-state unary 2NFA can be simulated by a 2DFA with a polynomial number of states then* L/poly $\supseteq$ NL

Hence

**Corollary**

L/poly $\supseteq$ NL *if and only if*
*the state cost of the simulation of unary 2NFAs by 2DFAs is poly*

# Outer Nondeterministic Automata (ONFAs)

## Definition

A two-way automaton is said to be *outer nondeterministic* iff nondeterministic choices are allowed *only* when the input head is scanning the endmarkers

Hence:

- No restrictions on the *input alphabet*
- No restrictions on *head reversals*
- *Deterministic transitions* on "real" input symbols
- *Nondeterministic choices* only at the endmarkers

# Outer Nondeterministic Automata (ONFAs)

*All the results* we obtained for the unary case
can be extended to ONFAs:
[Guillon Geffert&P '12, Kapoutsis&P '12]

(i) Subexponential simulation of 2ONFAs by 2DFAs

(ii) Polynomial complementation of unary 2ONFAs

(iii) Polynomial simulation of 2ONFAs by 2DFAs
*if and only if* L/poly $\supseteq$ NL

(iv) Polynomial simulation of 2ONFAs by unambiguous 2ONFAs

While in the unary case all the proofs rely on the conversion
of 2NFAs into quasi sweeping automata,
in the case of 2ONFAs we do not have a similar tool!

# Final Remarks

- The question of Sakoda and Sipser is very challenging
- In the investigation of restricted versions many interesting and not artificial models have been considered
- The results obtained for restricted versions of the problem, even if not solving the full problem, are nontrivial and, in many cases, very deep
- Strong connections with open questions in structural complexity
- Many times techniques used in space complexity can be adapted for the investigation of automata and vice versa

# Two Further Directions

- The results obtained in the unary case have been extended to the general case for outer nondeterministic automata

**Question**

Does it is possible to extend the same results (or some of them) to some less restricted models of computation?

- Input head reversals are a critical resource that deserves further investigation

**Theorem ([Kapoutsis&P '12])**

*Given $k > 0$, there exists a language $L$ such that each 2DFA accepting $L$ with less than $k$ head reversals is exponentially larger than each 2DFA with $k$ reversals*

**Question**

What about the power of head reversals combined with nondeterminism?

Thank you for your attention!