New Results Related to the Sakoda and Sipser Question

Giovanni Pighizzini

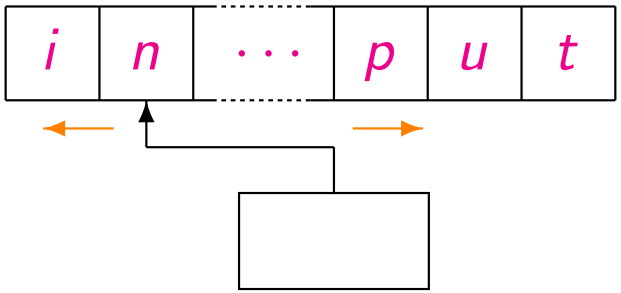
Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano

P.J. Šafárik University – Košice – Slovak Republic
November 7th, 2011

UNIVERSITÀ DEGLI STUDI
DI MILANO

Outline

Finite State Automata



Base version:

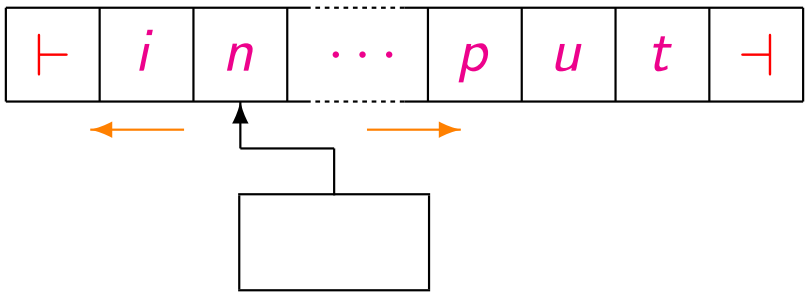
one-way deterministic finite automata (1DFA)

- 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

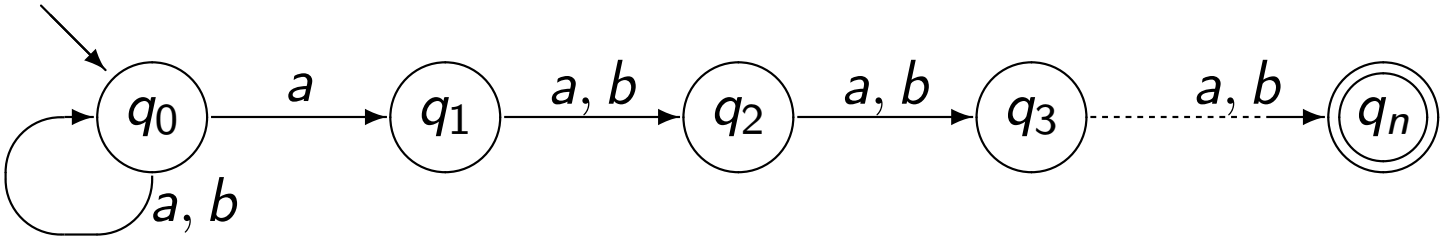
What about the power of these models?

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

...some of them are more succinct

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

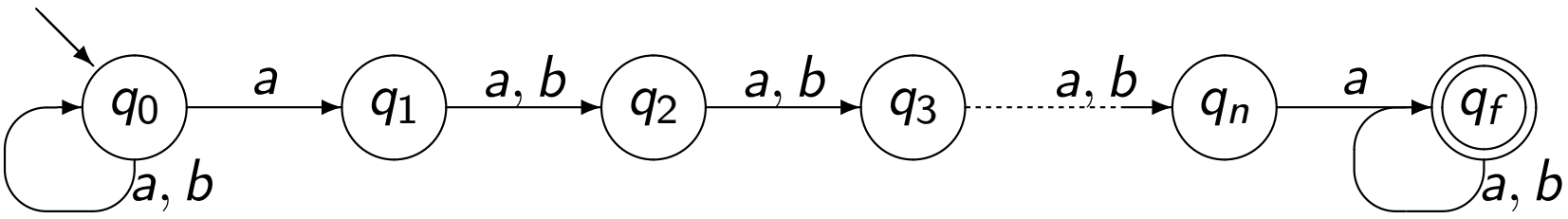
- L is accepted by a 1NFA with $n+1$ states



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

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

- L is accepted by a 1NFA with $n+2$ states



- The minimum 1DFA accepting L uses $3 \cdot 2^{n-1} + 1$ states
- Using head reversals the number of states becomes linear
- Even in this case L is accepted by
  - a 1NFA and a 2DFA with linearly related numbers of states
  - a minimum 1DFA exponentially larger

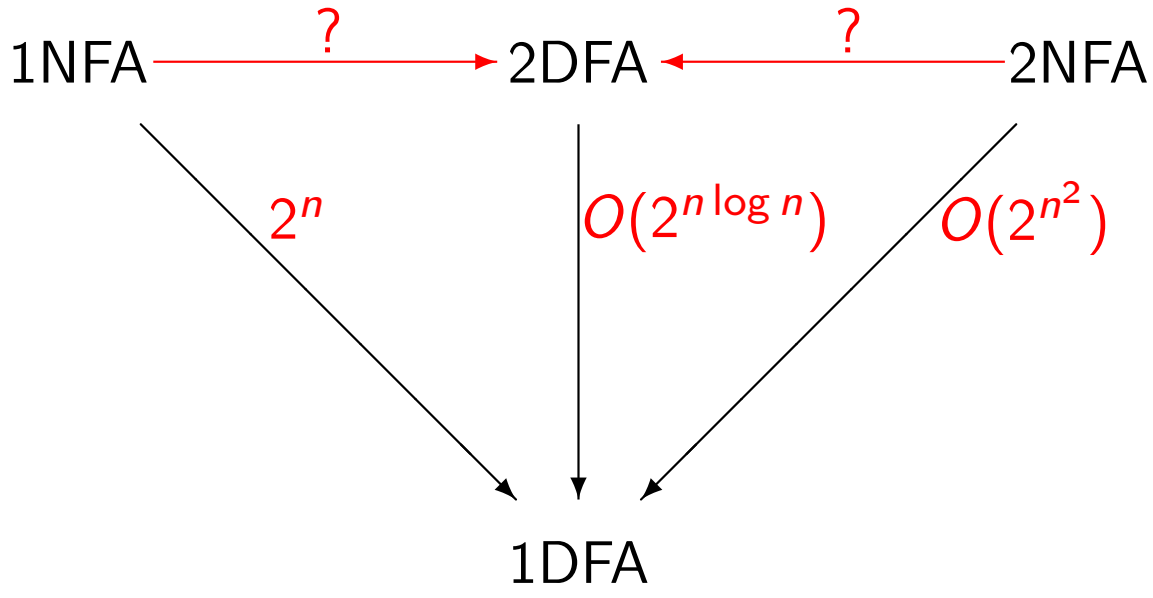
Example: $L = (a+b)^* a (a+b)^{n-1} a (a+b)^*$

$n = 4$

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**

- This can be implemented by a 2DFA with $O(n)$ states
- By a different algorithm, L can be also accepted by a 2DFA with $O(n)$ states *which changes the direction of its input head only at the endmarkers* (a *sweeping* automaton)

Costs of the Optimal Simulations Between Automata

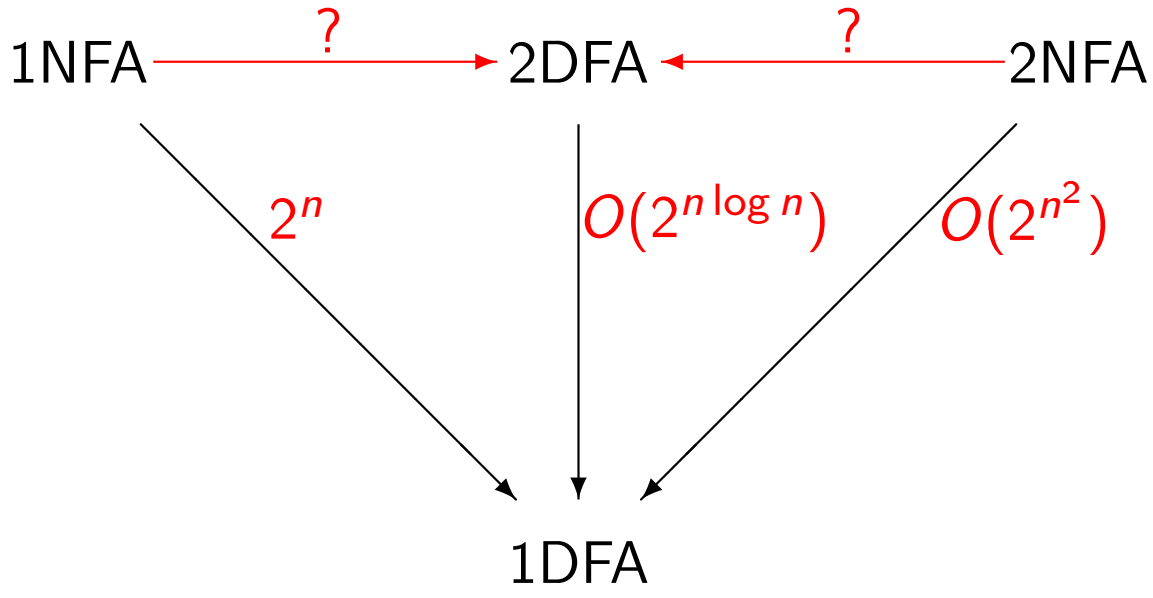


[Rabin&Scott '59, Shepardson '59, Meyer&Fischer '71, . . .]

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]

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 [Geffert Guillon&P '11]

The Unary Case: $\#\Sigma = 1$

1NFAs vs 2DFAs? Solved!

- The cost is $O(n^2)$
  [Chrobak '86]

2NFAs vs 2DFAs? It looks hard!

- Subexponential but superpolynomial upper bound $e^{O(\ln^2 n)}$
  [Geffert Mereghetti&P '03]
- Connection with the open question $\mathrm{L} \stackrel{?}{=} \mathrm{NL}$
  [Geffert&P '10, Kapoutsis&P '11]

Unary 2NFAs: A Fundamental Tool

In the investigation of unary 2NFAs,
the following notion and the next result are very useful:

Definition

A 2NFA is quasi sweeping (qsNFA) iff both

- nondeterministic choices and head reversals

are possible only at the endmarkers

Quasi Sweeping Simulation

Theorem ([Geffert Mereghetti&P '03])

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 $\mathsf{L} = \mathsf{NL}$
[Geffert&P '10]

(iv) Polynomial simulation of unary 2NFAs by unambiguous 2NFAs
[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

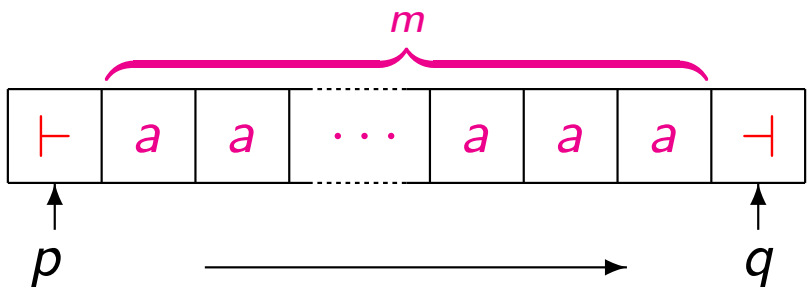
From now on, we fix an n-state unary 2NFA A

- We give a *reduction* from $L(A)$ to GAP
  i.e, for each input string a^m we define a graph $G(m)$ s.t.

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

- *Under the hypothesis* L = NL
  this reduction will be used to build 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:

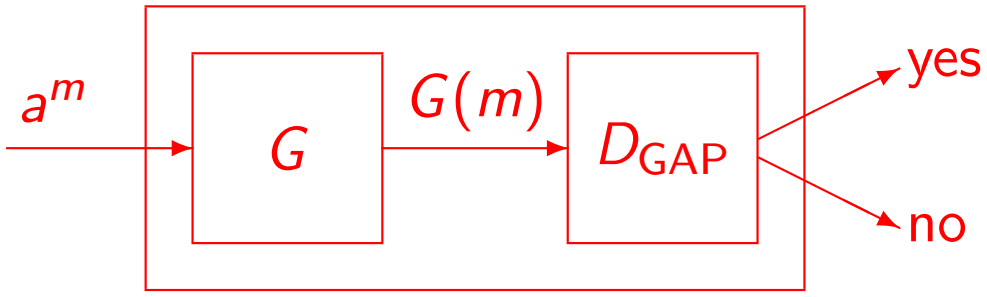
- the vertices are the states of M
- (p, q) is an edge iff M can traverse the input
  - from one endmarker in the state p
  - to the opposite endmarker in the state q
  - without visiting the endmarkers in the meantime

Then

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

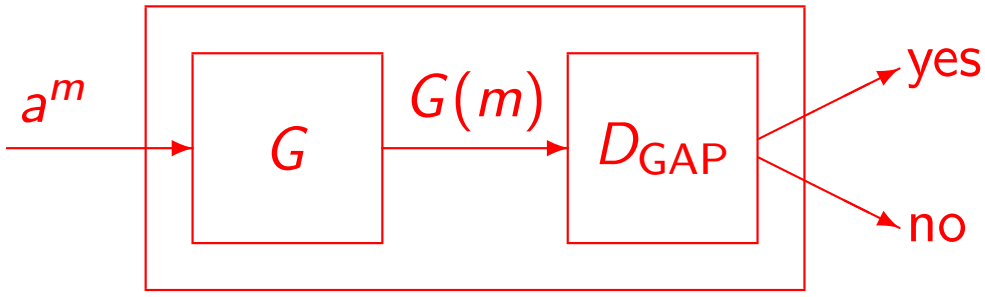
The existence of the edge (p, q) can be verified by a subroutine, implemented by a finite automaton $A_{p,q}$ with N states

Polynomial Deterministic Simulation (under L = NL)



- Suppose L = NL
- Let D_{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_{GAP}
- This decides whether or not $a^m \in L(M)$

Polynomial Deterministic Simulation (under L = NL)



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

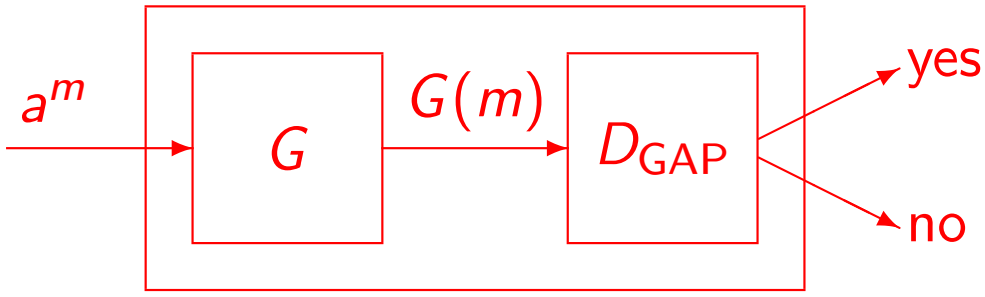
  The worktape of D_{GAP} can be encoded in a finite control using a number of states polynomial in N

- 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, input bits for D_{GAP} are computed "on demand"

Polynomial Deterministic Simulation (under L = NL)



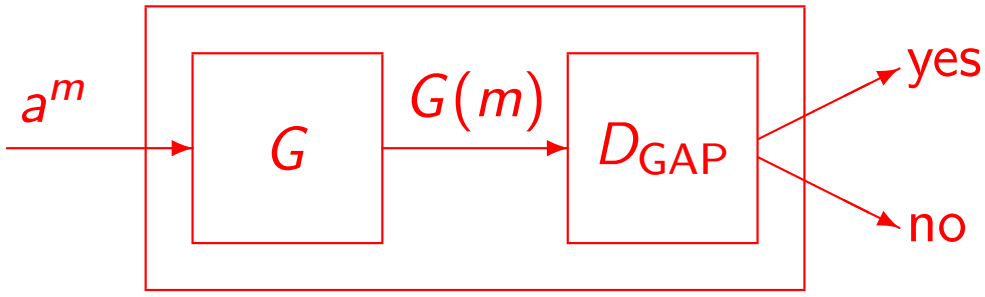
We define a unary 2DFA M' equivalent to M

- M' simulates D_{GAP} by keeping in its finite control:
  - The input head position of D_{GAP}
  - The worktape content of D_{GAP}
  - The finite control of D_{GAP}

  This uses a number of states polynomial in N
- Each time D_{GAP} needs an input bit, a subroutine $A_{p,q}$ is called

  Considering all possible (p, q), this uses at most N^3 states

Polynomial Deterministic Simulation (under L = NL)



We define a unary 2DFA M' equivalent to M

Summing Up...

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
$\Downarrow$	Quasi Sweeping Simulation	
M	qsNFA 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 Conditional Simulation

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?

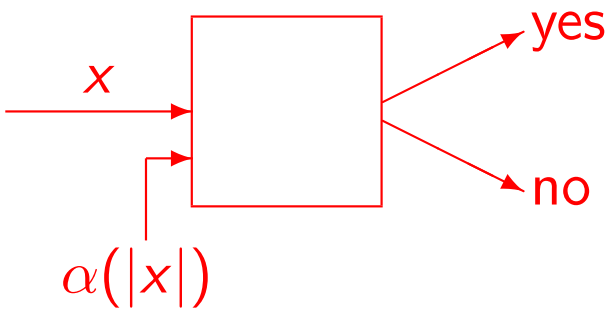
- We proved the converse *under an additional assumption*:

  The transformation from unary 2NFAs to 2DFAs must be computable in deterministic logspace

- Ch. Kapoutsis observed that the above theorem can be generalized by considering the *nonuniform* version of L
- For such generalization the converse also hold!

L/poly: Nonuniform Logarithmic Space

- L/poly
  class of languages accepted by deterministic logspace machines with a *polynomial advice*, i.e.,
- A sequence of strings $\{\alpha(n) \mid n \geq 0\}$ of polynomial length
- With each input string x, the logspace machine also receives the advice string $\alpha(|x|)$



Polynomial Deterministic Conditional Simulation

- Since $\mathsf{L} \subseteq \mathsf{NL}$ is known, our condition can be written as:

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

- The proof does not use the uniformity of L
- Hence, the condition $\mathsf{L} \supseteq \mathsf{NL}$ can be replaced by $\mathsf{L/poly} \supseteq \mathsf{NL}$:

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

- Since $\mathsf{L} \subseteq \mathsf{L/poly}$, the assumption is weaker
  So the last statement is stronger
- Even to prove the converse we use GAP

Solving GAP with Two-Way Automata

Binary Encoding: Languages BGAP

- Let n be a fixed integer
- 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 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
- Nondeterministic choices are taken only at the left endmarker
- Head reversals can be confined at the endmarkers

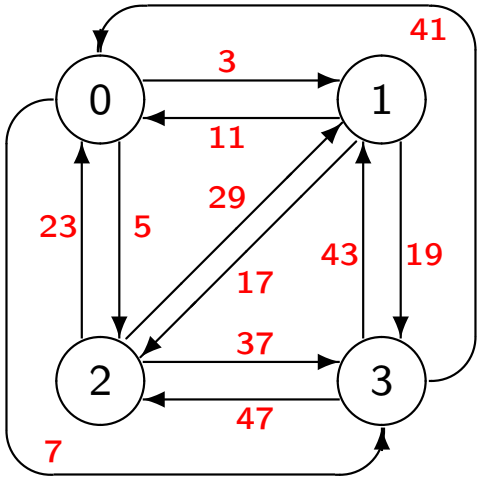
Solving GAP with Two-Way Automata

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 the prime number $p_{(i,j)} = p_{i \cdot n + j + 1}$
- A subgraph $G = (V_n, E)$ of K_n is encoded by the number

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

and by the string $\langle G \rangle_1 = a^{m_G}$



- Conversely, each string a^m denotes the graph $K_n(m)$ which contains the edge (i, j) iff $p_{(i,j)}$ divides m. Then $G = K_n(m_G)$
- $\mathrm{UGAP}_n := \{a^m \mid K_n(m) \text{ has a path from 0 to } n-1\}$

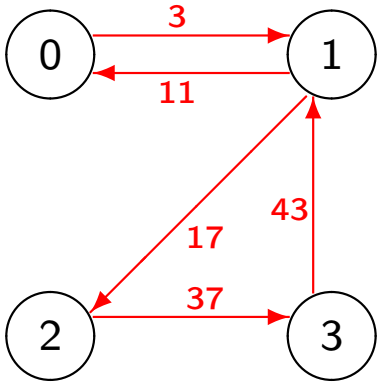
Solving GAP with Two-Way Automata

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 the prime number $p_{(i,j)} = p_{i \cdot n + j + 1}$
- A subgraph $G = (V_n, E)$ of K_n is encoded by the number

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

and by the string $\langle G \rangle_1 = a^{m_G}$



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

- Conversely, each string a^m denotes the graph $K_n(m)$ which contains the edge (i,j) iff $p_{(i,j)}$ divides m. Then $G = K_n(m_G)$
- $\mathrm{UGAP}_n := \{a^m \mid K_n(m) \text{ has a path from 0 to } n-1\}$

Solving GAP with Two-Way Automata

Recognizing UGAP_n

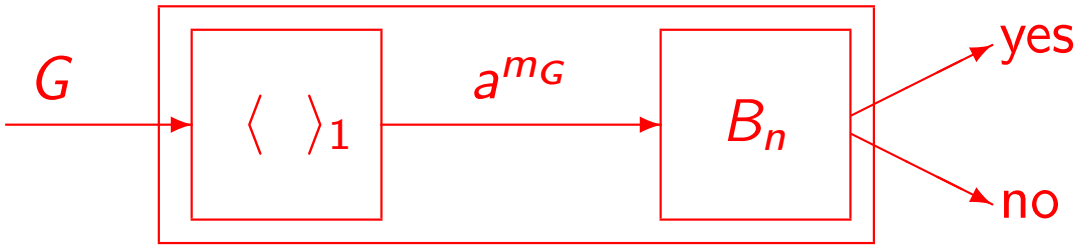
Unary version of the algorithm for BGAP_n

$i \leftarrow 0$ // 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
- Nondeterministic choices are taken only at the left endmarker
- Head reversals only at the endmarkers

Solving GAP with Two-Way Automata

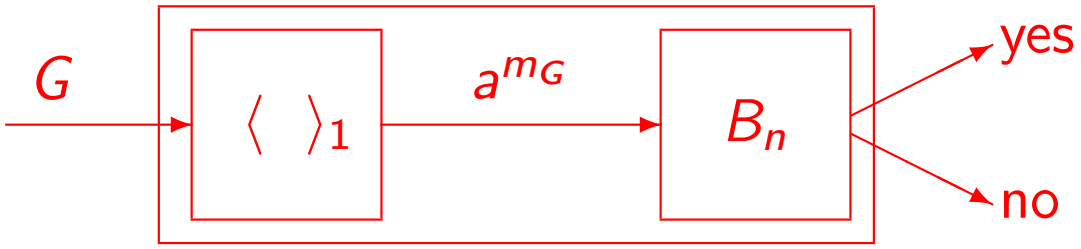
Outline of the Construction



- Suppose the conversion of unary 2NFAs into 2DFAs is polynomial
- Let B_n be a 2DFA with $poly(n)$ states recognizing 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

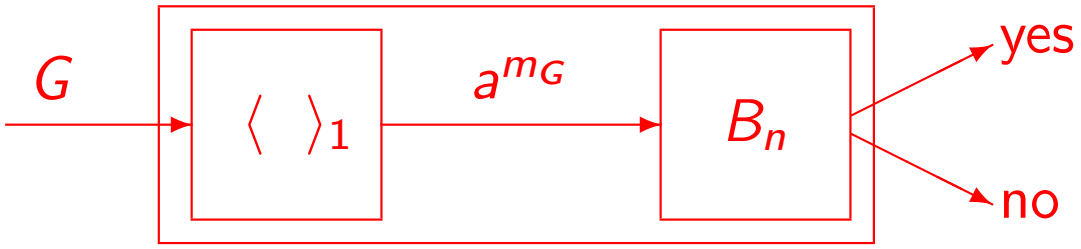
Outline of the Construction



- Our goal:
  - a deterministic machine
  - working in logarithmic space
  - using a polynomial advice
- The input is the graph G, hence its size is n^2
- The size of B_n is polynomial in n
- However representing a^{m_G} would require too much space
- Hence, we use a different strategy to represent m_G

Solving GAP with Two-Way Automata

Outline of the Construction



Next steps

1. A different representation of m_G:

   prime encoding of input lengths

2. The prime encoding of m_G
3. Replacing unary 2DFA inputs by prime encodings
4. Combining these things together to obtain a (nonuniform) logspace deterministic machine solving GAP

Prime Encodings

Given an integer m:

- $m = p_{i_1}^{\alpha_1} \cdot p_{i_2}^{\alpha_2} \cdots p_{i_k}^{\alpha_k}$ decomposition as product of prime powers
- A *prime encoding* of m is a string

$$\#z_1\#z_2 \cdots \#z_k$$

where $z_1, z_2, \ldots, z_k$ encode in an *arbitrary order* $p_{i_1}^{\alpha_1}, p_{i_2}^{\alpha_2}, \ldots, p_{i_k}^{\alpha_k}$

For simplicity:

- The factor z_i can be seen also as a number
- Hence, $m = z_1 \cdot z_2 \cdots z_k$

Prime Encodings and Graphs

Given an integer m:

- $m = p_{i_1}^{\alpha_1} \cdot p_{i_2}^{\alpha_2} \cdots p_{i_k}^{\alpha_k}$ decomposition as product of prime powers
- A *prime encoding* of m is a string

$$\#z_1 \#z_2 \cdots \#z_k$$

where $z_1, z_2, \ldots, z_k$ encode in an *arbitrary order* $p_{i_1}^{\alpha_1}, p_{i_2}^{\alpha_2}, \ldots, p_{i_k}^{\alpha_k}$

Given a graph $G = (V_n, E)$:

- A prime encoding of $m_G = \prod_{(i,j) \in E} p_{(i,j)}$ is just a list of all primes $p_{(i,j)}$ associated with the edges of G
- It can be computed in logarithmic space by *a deterministic transducer* T whose input is the adjacency matrix of G

Replacing Unary 2DFA Inputs by Prime Encodings

A Normal Form for Unary 2DFAs

From the results in [Chrobak '86, Kunc&Okhotin '11]
the following form for unary 2DFAs can be derived:

Theorem

Each s-state unary 2DFA A can be transformed into 2DFA M s.t.

- *M is sweeping*
- *in each sweep M counts the input length modulo some number ℓ*
- *M has $O(s)$ states*
- *M and A are "almost equivalent"*
  (differences are possible only for input of length $O(s)$)

Replacing Unary 2DFA Inputs by Prime Encodings

A Normal Form for Unary 2DFAs

- Let B an s-state unary 2DFA
- B is replaced by an almost equivalent 2DFA M in normal form with $O(s)$ states
- $L(B)$ and $L(M)$ can be differ on strings of length $< s_0 \in O(s)$
- We define a 2DFA B' "equivalent" to B:

  B' inputs represent prime encodings of B inputs
- On a prime encoding of an integer m,
  B' works in two phases:
  1. B' checks if the input is "short", i.e., $m < s_0$
     in this case, B' accepts or rejects according to a finite table
  2. otherwise, B' on its input simulates M on a^m
- The number of states of B' is polynomial in s

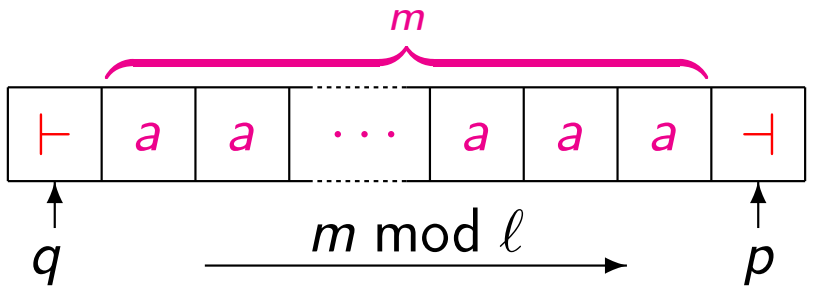
Replacing Unary 2DFA Inputs by Prime Encodings

Phase 1: Detecting Short Inputs

- Given
  - a^m, input of B
  - $\#z_1\#z_2\cdots\#z_k$, input of B', prime encoding of m
- For each $t < s_0$, B' checks if $m = t$:
  - B' checks if each z_i is a factor t
    - prime factorization of t: at most $\log t$ factors
    - length of each factor: at most $\log t$
    - one traversal of the input, $O(\log^2 t)$ states
  - B' checks if each prime power in the factorization of t is encoded by some z_i
    - at most $\log t$ traversals of the input, each using $\log t$ states
- If $m = t$ for some $t < s_0$ then the simulation stops, accepting or rejecting according to a finite table
- Since $s_0 \in O(s)$, by considering all possible t's, this phase can be implemented with $O(s \cdot \log^2 s)$ states

Replacing Unary 2DFA Inputs by Prime Encodings

Phase 2: Simulating the 2DFA M on Long Inputs



In each sweep:

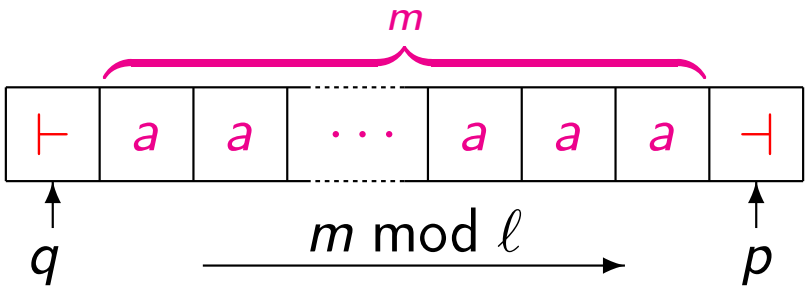
- M counts the input length modulo an integer ℓ
- The value of ℓ depends only on the starting state q
- The ending state p depends on q and on m mod ℓ

Let $\#z_1\#z_2\cdots\#z_k$ be a prime encoding of m:

- B' simulates the sweep of M with a sweep from q to p on $\#z_1\#z_2\cdots\#z_k$
- Hence, by considering the overall computation, M accepts a^m if and only if B' accepts $\#z_1\#z_2\cdots\#z_k$

Replacing Unary 2DFA Inputs by Prime Encodings

Phase 2: Simulating the 2DFA M on Long Inputs



- Since $m = z_1 \cdot z_2 \cdots z_k$ then:

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

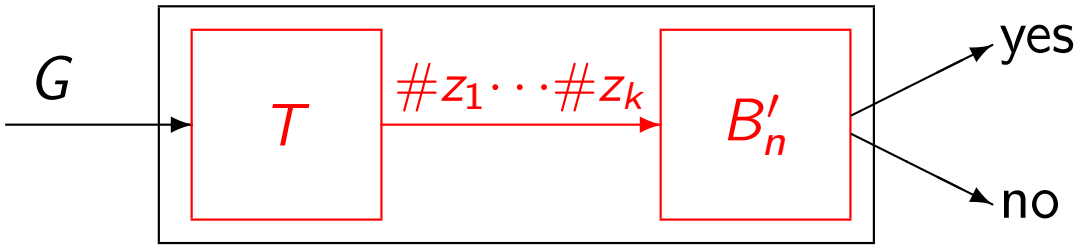
- $r = m \bmod \ell$ is obtained using the following iteration:

$r \leftarrow 1$
while there is a next factor $\#z$ **do**
  $r \leftarrow (r \cdot z) \bmod \ell$

- The state p is derived from q and r
- This phase can be implemented with $O(s^2)$ states

Solving GAP with Two-Way Automata

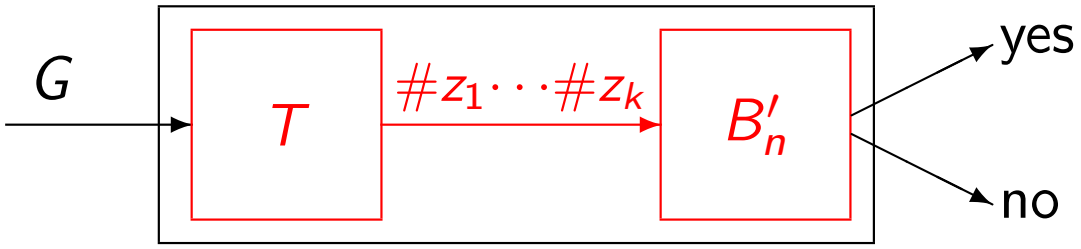
Combining All Together



- 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 $G \in \mathrm{GAP}_n$
- However, even representing $\#z_1 \cdots \#z_k$ would require too much space
- Hence, the symbols of $\#z_1 \cdots \#z_k$ are computed "on the fly", by restarting T each time B'_n needs them

Solving GAP with Two-Way Automata

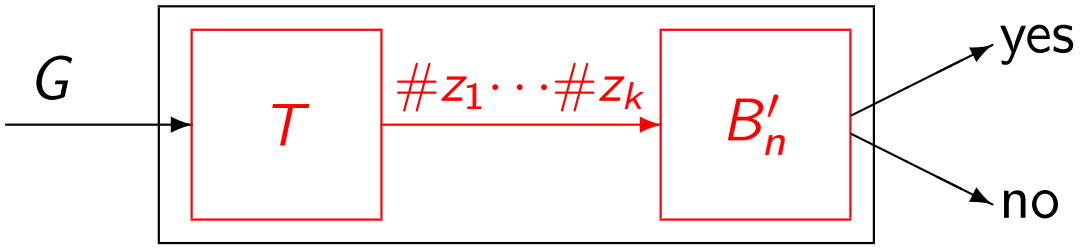
Combining All Together



- B'_n has $O(s^2)$ states, where $s = O(n^4 \log n)$ is the number of states of B_n
  Hence its size is polynomial in n^2, the size of the input G
- T works in space $O(\log n)$
- Hence the whole resulting machine works in logarithmic space
- However, we did not provided B'_n is a constructive way
  - The existence of B'_n 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

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 each n-state unary 2NFA can be simulated by a 2DFA with a polynomial number of states*

Characterizations of L/poly versus NL

2D: families of languages accepted by 2DFAs of polynomial size

2N: families of languages accepted by 2NFAs of polynomial size

2N/poly: restriction of 2N to *instances of polynomial length*

2N/unary: restriction of 2N to *unary instances*

Theorem ([Kapoutsis'11, Kapoutsis&P'11])

The following statements are equivalent:

1. L/poly $\supseteq$ NL
2. 2D $\supseteq$ 2N/poly
3. 2D $\supseteq$ 2N/unary
4. 2D $\ni$ BGAP
5. 2D $\ni$ UGAP

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: [Geffert Guillon&P '11]

(i) Subexponential simulation of 2ONFAs by 2DFAs

(ii) Polynomial complementation of unary 2ONFAs

(iii) Polynomial simulation of 2ONFAs by 2DFAs
under the condition 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!

Outer Nondeterministic Automata (ONFAs)

Main tool: procedure $reach(p, q)$

- The procedure checks the existence of a computation segment
  - from the left endmarker in the state p
  - to the left endmarker in the state q
  - not visiting the left endmarker in between
- Difficult point: possibility of infinite loops
- Implementation:
  - Modification of a technique for the complementation of 2DFAs [Geffert Mereghetti&P '07], which is a refinement of a construction for the complementation of space bounded Turing machines [Sipser '80]
  - Finite control with a *linear number* of states reading a two-way input tape

Loops involving endmarkers can be avoided by observing that for each accepting computation visiting one of the endmarkers more than $|Q|$ times there exists a shorter accepting computation

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 they do not solve 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

  An example of problem that could be investigated:

Question

Given $k > 0$, does 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?

A positive answer is known only for $k \leq 2$ [Balcerzac&Niwiński '10]