Two-way Unary Automata versus Logarithmic Space

Giovanni Pighizzini

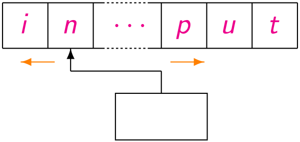
Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano

LIAFA - Paris
December 9th, 2011

UNIVERSITÀ DEGLI STUDI
DI MILANO

Outline

Finite State Automata



Base version:

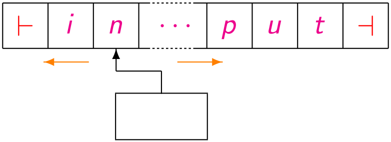
one-way deterministic finite automata (1DFA)

- 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

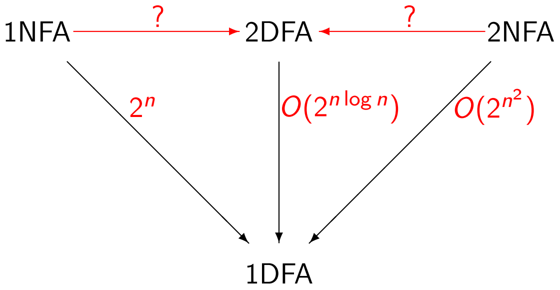
1DFA, 1NFA, 2DFA, 2NFA

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

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

$$
\begin{tikzcd}
\text{1NFA} \arrow[r, "?"] \arrow[dr, "2^n"] & \text{2DFA} \arrow[d, "O(2^{n \log n})"] & \text{2NFA} \arrow[l, "?"'] \arrow[dl, "O(2^{n^2})"] \\
& \text{1DFA} &
\end{tikzcd}
$$

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 $\mathsf{L} \stackrel{?}{=} \mathsf{NL}$
  [Geffert&P '10, Kapoutsis&P '11]

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 Conditional Simulation

Under the hypothesis L $=$ NL, the cost in states of the conversion of unary 2NFAs into 2DFAs is polynomial

[Geffert&P '10]

Outline of the proof

- Let A be an n-state unary 2NFA
- We describe a 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

  we use this reduction to build a 2DFA equivalent to A, with a number of states polynomial in n

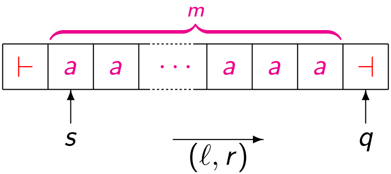
Polynomial Deterministic Conditional Simulation

We convert unary 2NFAs in a normal form:

A	fixed unary 2NFA	n states
$\Downarrow$		Conversion into Normal Form
M	2NFA almost equivalent to A	$N \leq 2n + 2$ states

- $L(M)$ and $L(A)$ can differ only on strings of length $\leq 5n^2$
- The computation of M is a sequence of traversals of the input
- The states used in each traversal form a *deterministic loop*
- Nondeterministic choices possible *only at the endmarkers*
- M has exactly one final state q_F
- q_F can be reached only at the left endmarker

Describing M Computations



Traversal of the input a^m

- starts from the leftmost input symbol in a state s
- moves at each step to the right
- finally reaches the right endmarker in a state q

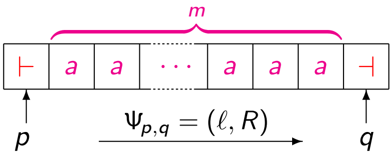
Then:

- s and q must belong to a same deterministic loop
- q depends on $m \bmod \ell$, where ℓ is the length of the loop

IDEA: Associate with (s, q), the pair of integers (ℓ, r) s.t.

$$\text{there is a traversal of } a^m \text{ from } s \text{ to } q \iff m \bmod \ell = r$$

Describing M Computations



However a traversal starts on the left endmarker

- we consider states p such that $p \xrightarrow{\vdash} s$
- actually, we associate the pair (ℓ, r) with (p, q).

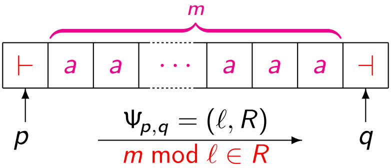
How many pairs (ℓ, r) can be associated with the same (p, q)?

- q belongs to a deterministic loop: only one possible ℓ
- on the left endmarker nondeterministic moves are possible: $p \xrightarrow{\vdash} s'$ and $p \xrightarrow{\vdash} s''$, for different s', s'' in the same loop of q, produce different remainders r: a set of possible remainders

With (p, q) we associate $\Psi_{p,q} = (\ell, R)$, where $R \subseteq \{0, \ldots, \ell - 1\}$

Similar argument for traversals from right to left

Describing M Computations



By summarizing:

Lemma

For all states p, q, input a^m, the automaton M

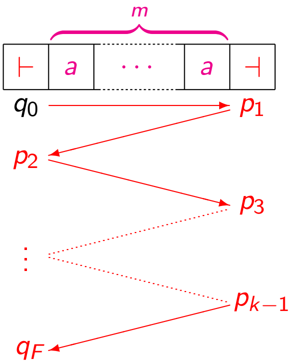
- *starting from one endmarker in the state p*
- *can reach the opposite endmarker in the state q*
- *without any visit of the endmarkers in between*

if and only if

- $\Psi_{p,q} = (\ell, R)$ *and* $m \bmod \ell \in R$

A finite automaton $A_{p,q}$ with $\ell \leq N$ states can test the existence of the traversal from p to q

An Accepting Computation



$m \bmod \ell_1 \in R_1 \qquad \Psi_{q_0,p_1} = (\ell_1, R_1)$

$m \bmod \ell_2 \in R_2 \qquad \Psi_{p_1,p_2} = (\ell_2, R_2)$

$m \bmod \ell_3 \in R_3 \qquad \Psi_{p_2,p_3} = (\ell_3, R_3)$

$\vdots \qquad \vdots$

$m \bmod \ell_k \in R_k \qquad \Psi_{p_{k-1},q_F} = (\ell_k, R_k)$

For each accepting computation
all these conditions are satified

Conversely:

- Each sequence of states $q_0 = p_1, p_2, \ldots, p_{k-1}, p_k = q_F$
  s.t. $m \bmod \ell_i \in R_i \quad (i = 1, \ldots, k)$
  describes an accepting computation for a^m

Reducing Membership for $L(M)$ to GAP

With each input a^m we associate the graph $G(m) = (Q, E(m))$, s.t.

$$(p, q) \in E(m) \text{ iff } m \bmod \ell \in R, \text{ where } \Psi_{p,q} = (\ell, R)$$

namely

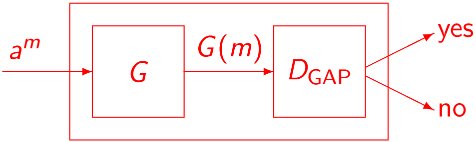
> *$G(m)$ contains the edge (p, q) if and only if there is a traversal from p to q on input a^m*

Lemma

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

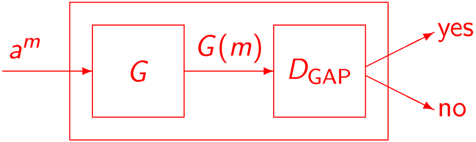
Hence this gives a reduction from $L(M)$ to GAP

Deterministic simulation



- 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 $a^m \in L(M)$

Deterministic simulation



- 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

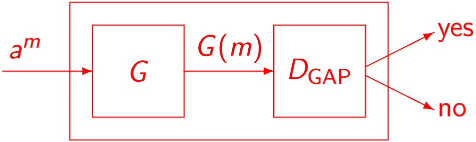
  The worktape of D_{GAP} can be encoded in a finite control using a number of states polynomial in N

- 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 we compute input bits for D_{GAP} "on demand"

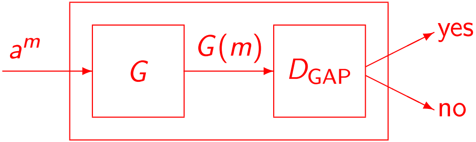
Deterministic simulation



We define a unary 2DFA M' equivalent to M

- M' keeps 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

Deterministic simulation



We define a unary 2DFA M' equivalent to M

- On input a^m, M' simulates D_{GAP} on input $G(m)$
- Input bits for D_{GAP} are the entries of $G(m)$ adjacency matrix
- Each time D_{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 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?

Later...

First we discuss a similar construction to make unary 2NFAs unambiguous

(Nonuniform) Unambiguous Logspace

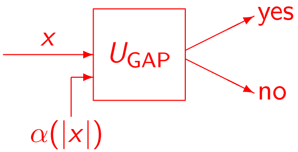
Theorem ([Reinhardt&Allender '00])

NL $\subseteq$ UL/poly

- UL/poly
  class of languages accepted by *unambiguous* 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 machine also receives the advice string $\alpha(|x|)$

Corollary

GAP $\in$ UL/poly



Making Unary 2NFAs Unambiguous

Theorem ([Geffert&P '10])

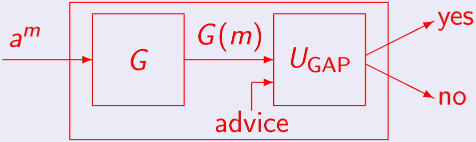
Each n-state unary 2NFA can be simulated by an equivalent unambiguous 2NFA with poly(n) many states

Proof.

- Similar to the polynomial deterministic conditional simulation
- Hypothetical machine D_{GAP} replaced with U_{GAP} and advice

Given a 2NFA the size of $G(m)$ (input of U_{GAP}) is fixed

- Hence the advice is fixed (i.e., it does not depend on a^m)
- Advice encoded in the hardware of the simulating machine



Polynomial Deterministic Conditional Simulation

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

What about the converse?

Polynomial Deterministic Conditional Simulation

- Since $\mathsf{L} \subseteq \mathsf{NL}$ is known, the statement 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*

- Ch. Kapoutsis observed that the proof does not use the uniformity of L
- Hence L can be replaced by L/poly

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

- Since $\mathsf{L} \subseteq \mathsf{L}/\mathrm{poly}$, the assumption is weaker
  So the last statement is stronger
- 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

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

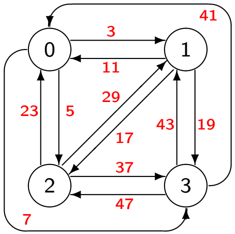
Standard nondeterministic algorithm solving graph accessibility

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

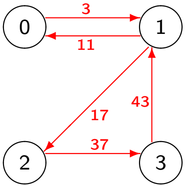
Solving GAP with Two-Way Automata

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 $\langle G \rangle_1 = a^{m_G}$



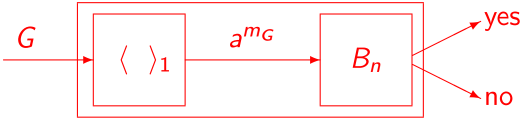
$$m_G = 3 \cdot 11 \cdot 17 \cdot 37 \cdot 43 = 892551$$

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

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

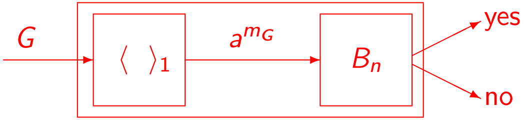
- Implementation using $O(n^4 \log n)$ states



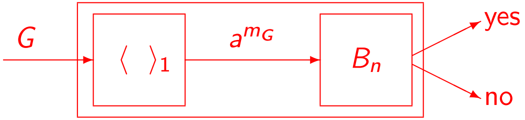
- Suppose the conversion of unary 2NFAs into 2DFAs is polynomial
- Let B_n be a 2DFA with $poly(n)$ states recognizing UGAP_n
- Given a graph $G = (V_n, E)$, compute its unary encoding a^{m_G} and give it as input to B_n
- This decides whether or not $G \in \text{GAP}$

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



Next steps

1. A different representation of m_G:

   prime encoding of input lengths

2. Replacing unary 2DFA inputs by prime encodings
3. 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 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

Given an s-state unary 2DFA B, we build an "equivalent' 2DFA B':

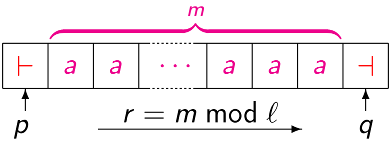
B' inputs represent prime encodings of B inputs

- First, we replace B by a 2DFA M with $O(s)$ states s.t.
  - M is sweeping
  - in each traversal M counts the input length modulo a number ℓ
  - $L(B)$ and $L(M)$ can be differ on strings of length $< s_0 \in O(s)$
- On a prime encoding of an integer m, B' works in two phases:
  1. B' checks if the input is "short", i.e., $m < s_0$
  2. otherwise, B' on its input simulates M on a^m
- The number of states of B' is polynomial in s

- 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
  - B' checks if each prime power in the factorization of t is encoded by some z_i
- If $m = t$ for some $t < s_0$ then the simulation stops, accepting or rejecting according to a finite table
- This phase is implemented using $O(s \cdot \log^2 s)$ states

Replacing Unary 2DFA Inputs by Prime Encodings

Phase 2: Simulating the 2DFA M on Long Inputs



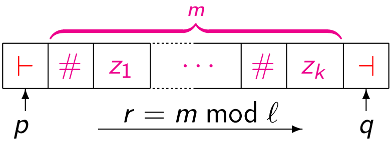
In a sweep:

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

B' simulates the same sweep on input $\#z_1\#z_2 \cdots \#z_k$,
a prime encoding of m

Replacing Unary 2DFA Inputs by Prime Encodings

Phase 2: Simulating the 2DFA M on Long Inputs



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

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

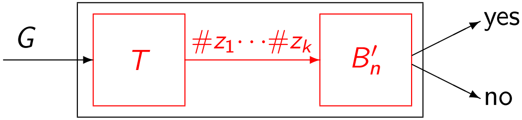
- r is obtained using the following iteration:

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

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

Solving GAP with Two-Way Automata

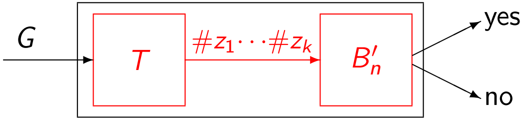
Combining All Together



- 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 $G \in \mathrm{GAP}_n$
- 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

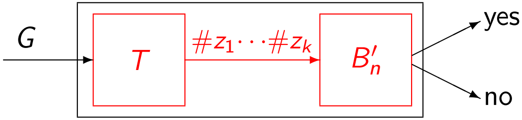
Combining All Together



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

Two-way Automata 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

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