Chapter 7

The Nonsolvable Group Technique

Louis Deaett • Robert Swier

April 25, 2001

CSC486

Definitions

Before embarking on a proof, we offer some definitions of essential terms and classes.

Bounded width-k branching program (informal) – A program which essentially prescribes the manipulation of a pebble placed on a board of k squares.

- The pebble starts at square 1.

- A series of instructions is executed where each instruction is a 3-tuple (i_j, μ_j^0, μ_j^1) where j is the step of the computation, i_j is the index of the input bit to be examined, and depending on whether it is 0 or 1, μ_j^0 or μ_j^1 is executed, where each of these is a mapping of $\{1,\ldots,k\}$ into $\{1,\ldots,k\}$ (essentially saying where to move the pebble).

- The number of instructions is polynomial in the length of the input.

- The program accepts iff the pebble is on square 1 at the end.

Definitions

Bounded width-k branching program (formal) – A polynomial-length sequence of instructions P of the form (i_j, μ_j^0, μ_j^1) for $1 \le i_j \le n$, where n is the input length and μ_j^0, μ_j^1 are mappings of $\{1,\ldots,k\}$ into itself.

- Given input $x = x_1 \cdots x_n$ the product of P wrt x, $P(x)$, is $\mu_m^{x_{i_m}} \circ \cdots \circ \mu_1^{x_{i_1}}$ (from right to left), where m is the length of the program P
- The program accepts iff $P(x)$ is a function which maps 1 to 1.

Definitions

k-PBP = {L | L is accepted by width-k polynomial-size branching programs}

What does this mean?

A language is accepted by width-k polynomial-size branching programs if there is a family $\{P_n\}_{n\geq 1}$ of width-k branching programs such that

1. There exists a polynomial p such that for every n, length of P_n is at most $p(n)$ and

2. For all x, $x \in L$ iff $P_{|x|}$ accepts x.

Definitions

$\mathbf{NC^k}$ = {L | L is accepted by a logspace uniform $O(\log^k n)$-depth, polynomial-size, bounded fan-in circuit family}.

$\mathbf{nonuniform-NC^k}$ = {L | L is accepted by a $O(\log^k n)$-depth, polynomial-size, bounded fan-in circuit family}.

What does this mean? First, the definition of a circuit:

boolean (fan-in) circuit – C_n with n inputs is a labeled, directed acyclic graph with nodes having indegree zero or at least two.

- Nodes with indegree zero are labeled from the set $\{0,1,x_1,\ldots,x_n,\bar{x}_1,\ldots,\bar{x}_n\}$
- All other nodes are labeled by either by $\wedge$ or $\vee$, and compute the operation indicated by their label.
- A language L is accepted by a family $F = \{C_n\}_{n\geq 1}$ of boolean circuits if for every x, $x \in L$ iff $C_{|x|}$ on x evaluates to 1.

Definitions

logspace-uniform – a family $\{C_n\}_{n\geq 1}$ is logspace-uniform iff there exists a logarithmic space-bounded TM that computes the description of C_n given 1^n.

bounded fan-in circuit – A fan-in circuit such that all AND and OR gates have indegree two.

Definitions

Group – A group is some set G, which is closed under some binary operation $*$, where $*$ is associative, has an identity element in G, and is such that each element in G has an inverse wrt $*$.

Trivial Group – The group $\{e\}$, where e is the identity element.

Permutation Group – If A is a nonempty set and S_A is a set of all the permutations of A, then S_A is a permutation group closed under the permutation multiplication operation.

Example of permutation multiplication:

Suppose that $A = \{1,2,3,4,5\}$, $\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 2 & 5 & 3 & 1 \end{pmatrix}$, and $\tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 4 & 2 & 1 \end{pmatrix}$. Then

$$\sigma * \tau = \sigma\tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 2 & 5 & 3 & 1 \end{pmatrix}\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 4 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 3 & 2 & 4 \end{pmatrix}$$

- Multiplication is right-to-left.
- This is just the composition of the two permutation functions.

Definitions

Cycle Notation – (a,b,c,d,e) is the permutation that maps a to b, b to c, c to d, d to e, and e to a.

Subgroup – A is a subgroup of B if $A \subseteq B$ and A is forms a group using the same binary operation used in B.

Commutator – $a^{-1}b^{-1}ab$ is a commutator of a and b.

Commutator Subgroup – A commutator subgroup of G is the group generated by the elements $\left\{a^{-1}b^{-1}ab \mid a,b \in G\right\}$.

Derived Series – A series $G_0, G_1...$ such that $G_0 = G$ and for $i \leq 1$, each G_i is the commutator subgroup of G_{i-1}.

Solvable Group – A group is solvable iff its derived series converges to the trivial group.

Nonsolvable Group – A group is nonsolvable if it is not solvable.

The point of these definitions

We wonder whether polynomial-size bounded-width branching programs can simulate nonuniform-NC^1? If so, what is the smallest k such that polynomial size width-k branching programs can simulate nonuniform-NC^1?

- Answer to first question: YES. (Proof is coming shortly).

- Answer to the second question: Probably 5.
    - The permutation group over $\{1 \ldots k\}$ is solvable for $k = 1,2,3,4$, but it is nonsolvable for $k \geq 5$. This appears to be the crucial difference. We can exploit the nonsolvability of the permutation groups over $\{1,2,3,4,5\}$ to design a family of polynomial size, width-5 branching programs for each language in nonuniform-NC^1.
    - In other words, nonuniform-NC^1=5-BPB.

-

And now the proof...

Theorem: $5-\mathrm{PBP} = \textit{nonuniform}-\mathrm{NC}^1$

Proof Roadmap

- Show containment in both directions
- $k-\mathrm{PBP} \subseteq \textit{nonuniform}-\mathrm{NC}^1$ for all $k \in \mathrm{N}^+$
  - Take an arbitrary language in k-PBP
  - It's decided by a family of width-k branching programs
  - From each program, construct a circuit for inputs of the same length
  - Result: Family of circuits that decides the language
  - Thus: The language must be in NC^1
- $\textit{nonuniform}-\mathrm{NC}^1 \subseteq 5-\mathrm{PBP}$
  - Take an arbitrary language in $\textit{nonuniform}-\mathrm{NC}^1$
  - It's decided by a family of polynomial size, logarithmic depth circuits
  - From each circuit, construct a width-5 branching program for inputs of the same length
  - Result: Family of width-5 branching programs that decides the language
  - Thus: The language must be in $5-\mathrm{PBP}$
- **QED**

$\forall k \in \mathrm{N}^+,\ k - \mathrm{PBP} \subseteq \mathit{nonuniform} - \mathrm{NC}^1$

- Let k be any positive integer
- Let L be any language in $k - \mathrm{PBP}$
- Then L is decided be some family of width-k branching programs, call it

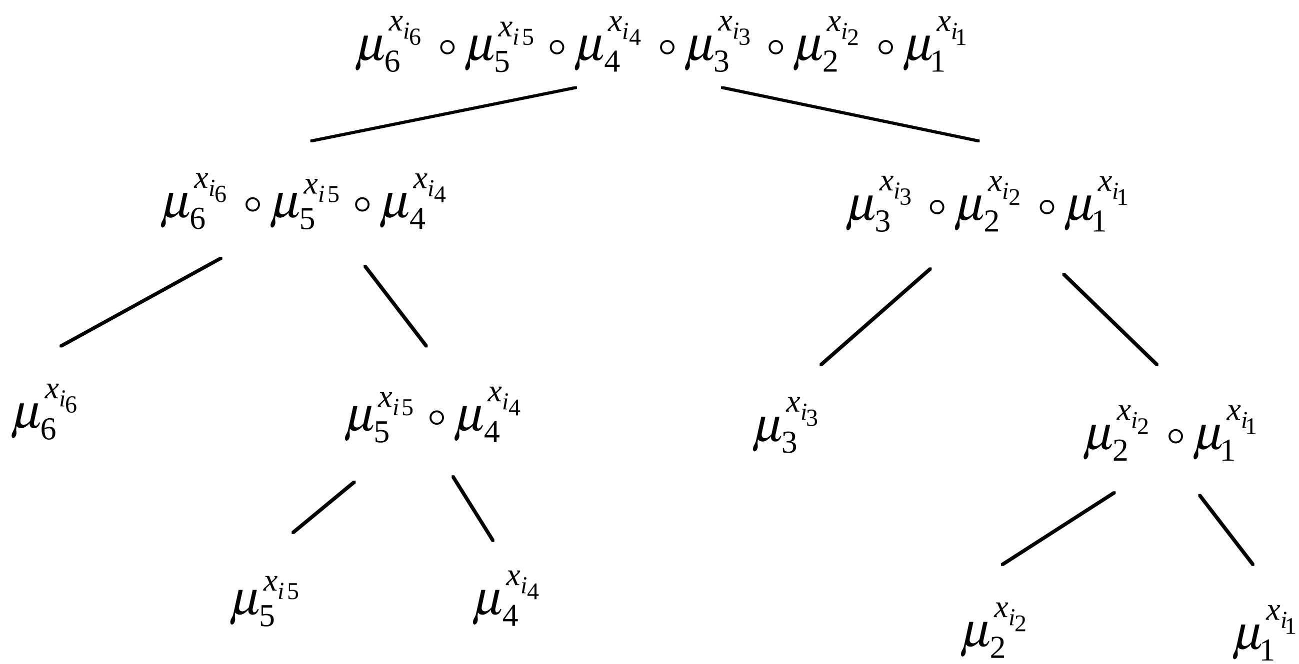
$$P_L = \{P_n\}_{n \geq 1}$$

such that

- $\forall x \in \Sigma^n,\ x \in L \Leftrightarrow P_n$ accepts x
- $\exists$ a polynomial q such that the size (number of instructions) of P_n is at most $q(n)$ for each n

- We want to represent each P_n by a circuit
- First, fix a binary encoding of all functions from $\{1,\ldots,k\}$ to $\{1,\ldots,k\}$. Since there are k^k such functions, the code for each function will be of length $\lceil \log(k^k) \rceil = \lceil k \log k \rceil$. Call this length d.
- We will construct a full binary tree T_n representing the computation of P_n, and we will label the nodes of T_n using this encoding of the functions.

- Say $P_n = \{(i_1, \mu_1^0, \mu_1^1), (i_2, \mu_2^0, \mu_2^1), \ldots, (i_{m-1}, \mu_{m-1}^0, \mu_{m-1}^1), (i_m, \mu_m^0, \mu_m^1)\}$
- Then $P_n(x) = \mu_m^{x_{i_m}} \circ \mu_{m-1}^{x_{i_{m-1}}} \circ \cdots \circ \mu_2^{x_{i_2}} \circ \mu_1^{x_{i_1}}$
- The tree T_n looks something like this...



...just to give you an idea! Now let's make this precise.

To construct T_n:

1) Label the root

$$\mu_m^{x_{i_m}} \circ \mu_{m-1}^{x_{i_{m-1}}} \circ \cdots \circ \mu_2^{x_{i_2}} \circ \mu_1^{x_{i_1}}$$

2) If a node is labeled by

$$\mu_s^{x_{i_s}} \circ \cdots \circ \mu_t^{x_{i_t}}$$

for some s and t with $1 \le t < s \le m$,
then label its left child by

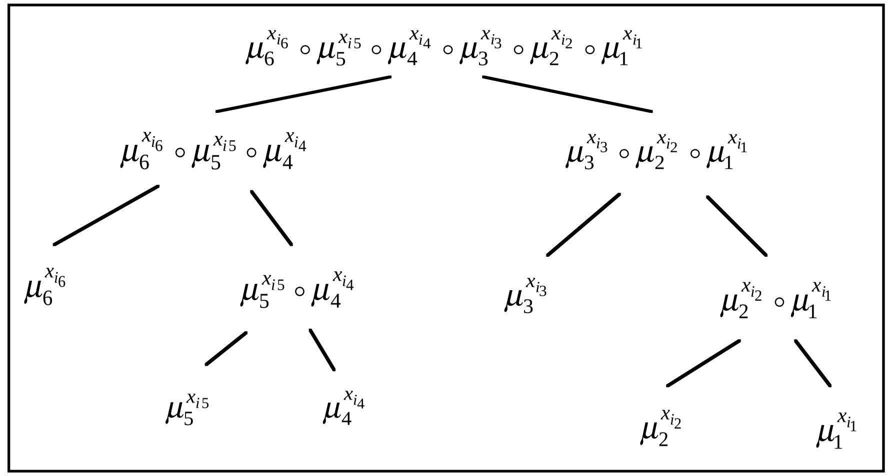
$$\mu_s^{x_{i_s}} \circ \cdots \circ \mu_{l+1}^{x_{i_{l+1}}}$$

and label its right child by

$$\mu_l^{x_{i_l}} \circ \cdots \circ \mu 1_t^{x_{i_1}}$$

where $l = \left\lceil \frac{s+t}{2} \right\rceil$

3) If a node is a labeled by a single element, then it is a leaf, so we don't have to worry about labeling its children.



What's the depth of T_n? Clearly it's $\lceil \log m \rceil$.

- Having constructed T_n, we now convert it to a circuit C_n representing P_n

- We perform the conversion from the leaves up

- The conversion will replace each leaf node with a circuit having 1 input and d outputs

- The conversion will replace each interior node with a circuit having $2d$ inputs and d outputs (with a slight addition at the root)

- **Intuition:** The circuits for the leaf nodes will compute the action of each instruction

- **Intuition:** The circuits for the non-leaf nodes will compute the combined action of all the instructions together

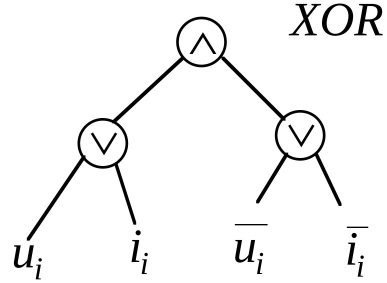
Conversion: The Leaves

- Let w be some leaf of the tree T_n
- Say w is labeled by $\mu_s^{x_{i_s}}$
- Then the circuit for w has input x_{i_s}
- Let a be the d-bit encoding of μ_s^0
- Let b be the d-bit encoding of μ_s^1
- The the circuit for w has d output bits, call them $y_1, \ldots, y_d$
- For each i, the circuit computes y_i as
  - 0 if $a_r = b_r = 0$
  - 1 if $a_r = b_r = 1$
  - x_{i_s} if $a_r = 0$ and $b_r = 1$
  - $\overline{x_{i_s}}$ if $a_r = 1$ and $b_r = 0$
- Now $y_1 y_2 \cdots y_d$ is a if $x_{i_s} = 0$ and is b if $x_{i_s} = 1$
- Size of the circuit is clearly $O(d)$, and this is $O(1)$, since d is a constant
- Depth of the circuit is clearly 2, which is also constant

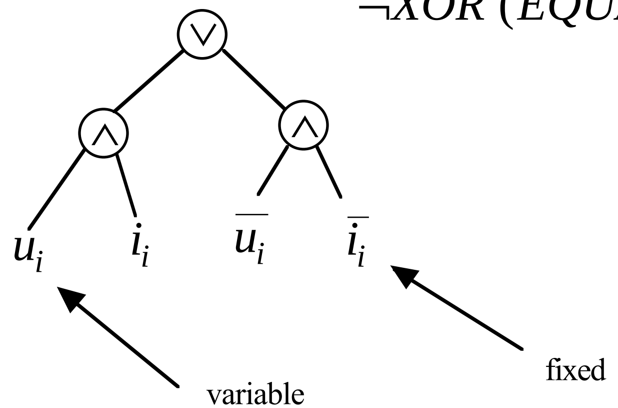
Conversion: The Non-leaf Nodes

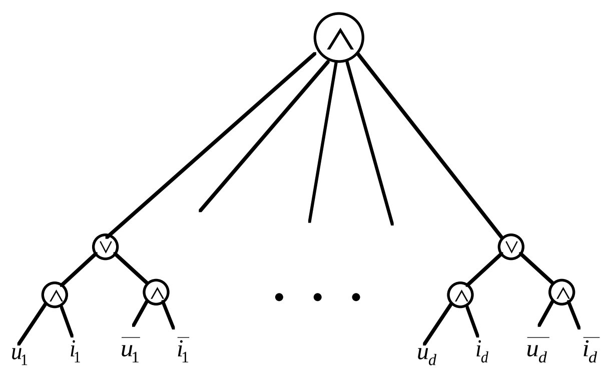
- Let w be a non-leaf node
- Let $u_1,\ldots,u_d$ be the output bits of w's left child
- Let $v_1,\ldots,v_d$ be the output bits of w's right child
- Interpret $u_1u_2\cdots u_d$ to be the encoding of some function f from $\{1,\ldots,k\}$ to $\{1,\ldots,k\}$.
- Interpret $v_1v_2\cdots v_d$ to be the encoding of some function g from $\{1,\ldots,k\}$ to $\{1,\ldots,k\}$.
- Then the circuit for w takes as input $u_1,\ldots,u_d$ and $v_1,\ldots,v_d$ and computes $y_1,\ldots,y_d$, where $y_1y_2\cdots y_d$ is an encoding of the function $f \circ g$.

XOR



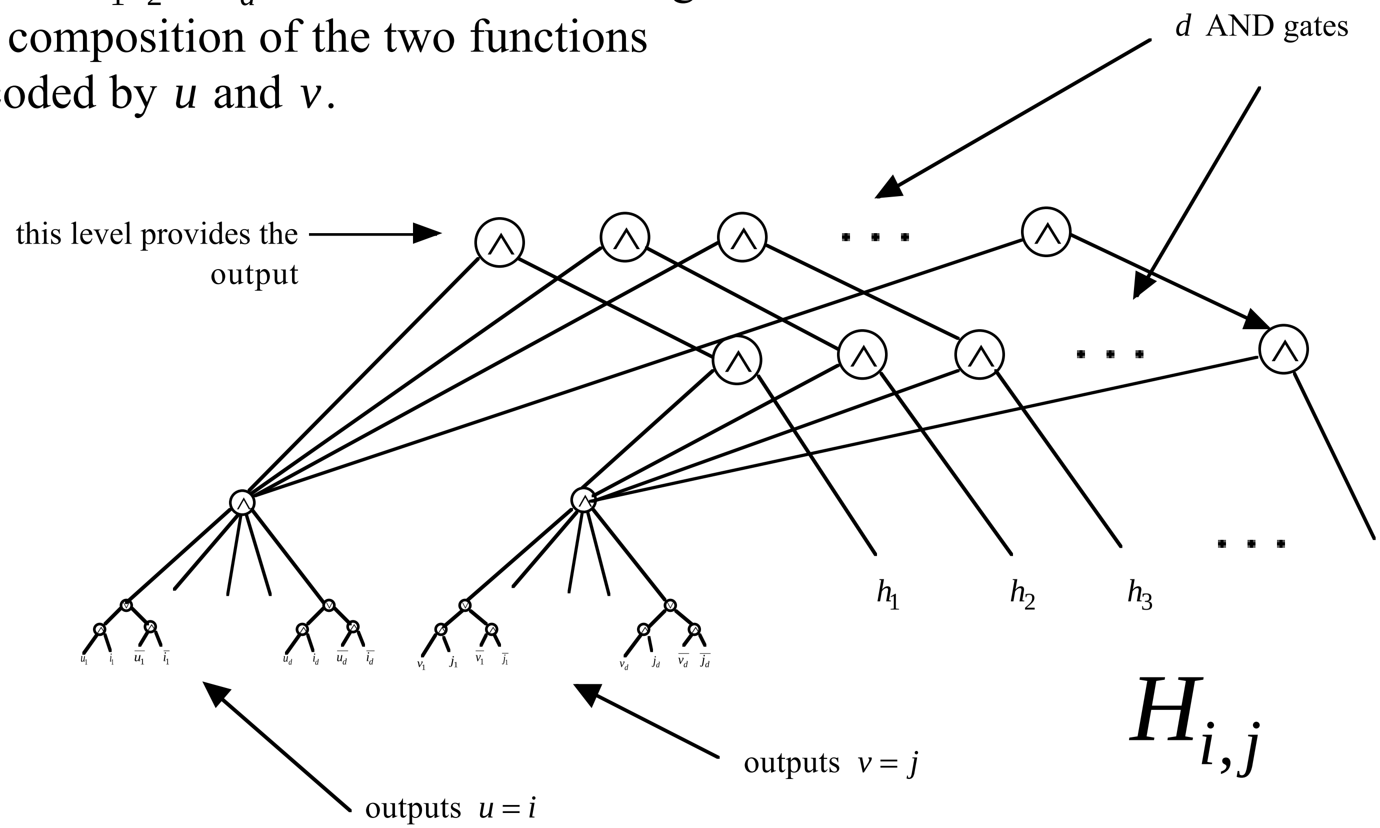
$\neg XOR$ ($EQUALS$)





All sizes and depths are functions of d and k, so they are all $O(1)$ with respect to n

Let $h = h_1 h_2 \cdots h_d$ be the d-bit encoding of the composition of the two functions encoded by u and v.



We construct a circuit H as the OR of all $H_{i,j}$ for every possible i and j. There are $(k^k)^2$ of them. Then H is the desired circuit for function composition.

Conversion: The Root

- We apply the non-leaf node conversion just described to the root as well, but we don't stop there!

- We add a circuit that takes as input the d-bit output of the root, which is the encoding of

$$\mu_m^{x_{i_m}} \circ \mu_{m-1}^{x_{i_{m-1}}} \circ \cdots \circ \mu_2^{x_{i_2}} \circ \mu_1^{x_{i_1}}$$

and outputs

1 if the encoding is of a function that fixes 1

and

0 otherwise

Conversion: The Result

- The complete circuit, call it C_n, outputs 1 if

$$\mu_m^{x_{i_m}} \circ \mu_{m-1}^{x_{i_{m-1}}} \circ \cdots \circ \mu_2^{x_{i_2}} \circ \mu_1^{x_{i_1}}$$

  fixes 1, and 0 otherwise

- That is, the circuit outputs 1 on input x if and only if the program P_n accepts x, which is the case if and only if $x \in L$

- Thus, the family of circuits

$$\{C_n\}_{n \geq 1}$$

  accepts L

- The size of C_n is $O(m) = O(p(n))$
- The depth of each C_n is $O(\log m) = O(\log(p(n))) = O(\log n)$
- Therefore, $L \in \mathit{nonuniform} - \mathrm{NC}^1$
- And so we've shown that $k - \mathrm{PBP} \subseteq \mathrm{NC}^1$

$\mathit{nonuniform} - \mathrm{NC}^1 \subseteq 5 - \mathrm{PBP}$

- Let L be any language in $\mathit{nonuniform} - \mathrm{NC}^1$
- $\mathit{nonuniform} - \mathrm{NC}^1$ is closed under complementation, so $\overline{L} \in \mathit{nonuniform} - \mathrm{NC}^1$ as well
- So there exists a family of bounded-fan-in, $O(\log n)$ depth, polynomial size boolean circuits, call it $C = \{C_n\}_{n\geq 1}$, that decides $\overline{L}$

A Brief Side Trip To S_5

- Define the following permutations:

$\alpha = (1\ 2\ 3\ 4\ 5)$	$\theta_0 = (1\ 2\ 5\ 3\ 4)$
$\beta = (1\ 3\ 5\ 4\ 2)$	$\theta_1 = (1\ 4\ 2\ 5\ 3)$
$\gamma = (1\ 3\ 2\ 5\ 4)$	$\theta_2 = (1\ 4\ 3\ 2\ 5)$
	$\theta_3 = (1\ 5\ 3\ 2\ 4)$

- Note the following facts
  1. γ is the commutator of α and β. That is, $\gamma = \beta^{-1} \circ \alpha^{-1} \circ \beta \circ \alpha$. Also, $\gamma^{-1} = \alpha^{-1} \circ \beta^{-1} \circ \alpha \circ \beta$
  2. $\theta_0^{-1} \circ \gamma \circ \theta_0 = \alpha$ (and so $\theta_0^{-1} \circ \gamma^{-1} \circ \theta_0 = \alpha^{-1}$)
  3. $\theta_1^{-1} \circ \gamma \circ \theta_1 = \beta$ (and so $\theta_1^{-1} \circ \gamma^{-1} \circ \theta_1 = \beta^{-1}$)
  4. $\theta_2^{-1} \circ \gamma \circ \theta_2 = \alpha^{-1}$ (and so $\theta_2^{-1} \circ \gamma^{-1} \circ \theta_2 = \alpha$)
  5. $\theta_3^{-1} \circ \gamma \circ \theta_3 = \beta^{-1}$ (and so $\theta_3^{-1} \circ \gamma^{-1} \circ \theta_3 = \beta$)

- Finally, note that γ does <u>not</u> fix 1

And Now, Some Notation

- If f is any gate in C_n, let $f(x)$ denote the output of f on input x.
- Suppose:

  Q is a branching program on binary inputs of length n

  f is a gate in C_n

  θ, ξ are two permutations in S_5

- Then if, for every $x \in \{0,1\}^n$,

$$Q(x) = \begin{cases} \theta \text{ if } f(x) = 0 \\ \xi \text{ if } f(x) = 1 \end{cases}$$

we say that Q is a (θ, ξ) program for f.

Construction Of The Branching Program

- Ultimately, we wish to construct an (I_5, γ) program for the output gate of C_n.

- Why?

- Because C_n decides length n strings $\overline{L}$, its output gate has value 0 (and so our program must have output I_5) if the input is in L, and its output gate has value 1 (and so out program must have output γ) if the input is not in L.

- We construct such a program by constructing, for each gate f in C_n, a (I_5, γ) program for f, which we denote $P[f]$.

- The construction begins with the input gates and proceeds upwards.

Construction: The "Input" Gates

- Let f be an input gate
- If f is labeled by x_i then let $P[f] = \{(i, I_5, \gamma)\}$
- If f is labeled by $\overline{x_i}$ then let $P[f] = \{(i, \gamma, I_5)\}$
- If f is labeled by 0 then let $P[f] = \{(1, I_5, I_5)\}$
- If f is labeled by 1 then let $P[f] = \{(1, \gamma, \gamma)\}$
- In each case, it is clear that $P[f]$ is a (I_5, γ) program for f.

Construction: The AND Gates

- Let f be an AND gate
- Let g and h be the two gates providing input to f
- Assume inductively that we have already constructed

  an (I_5,γ) program $P[g]$ of size k for g

  and

  an (I_5,γ) program $P[h]$ of size l for h

- Construct a program S_0 from $P[g]$ such that, for every input x,

  $S_0(x) = \theta_0^{-1} \circ P[g](x) \circ \theta_0$
- To construct S_0 we replace the first instruction of $P[g]$, say (i,μ^0,μ^1) by $(i,\mu^0 \circ \theta_0, \mu^1 \circ \theta_0)$ and replace the last instruction of $P[g]$, say (j,ν^0,ν^1), by $(j,\theta_0^{-1} \circ \nu^0, \theta_0^{-1} \circ \nu^1)$
- Although if $P[g]$ consists of a single instruction, then we replace that instruction, say (i,μ,ν) with $(i,\theta_0^{-1} \circ \mu \circ \theta_0, \theta_0^{-1} \circ \nu \circ \theta_0)$
- Now, since $P[g]$ is an (I_5,γ) program for g, it follows that S_0 is a $(\theta_0^{-1} \circ I_5 \circ \theta_0, \theta_0^{-1} \circ \gamma \circ \theta_0)$ program for g. But $\theta_0^{-1} \circ I_5 \circ \theta_0 = I_5$ and, as previously noted, $\theta_0^{-1} \circ \gamma \circ \theta_0 = \alpha$. Thus, S_0 is an (I_5,α) program of size k for g.

Construction: The AND Gates (continued)

- Using this procedure, we construct
- S_0, an (I_5,α) program of size k for g (as just described explicitly)
- S_1, an (I_5,β) program of size l for h using θ_1 in place of θ_0
- S_2, an (I_5,α^{-1}) program of size k for $g$using θ_2 in place of θ_0
- S_3, an (I_5,β^{-1}) program of size l for $h$using θ_3 in place of θ_0
- Now define $P[f]$ to be the program that executes S_0,S_1,S_2,S_3 in that order
- Then, for every $x \in \{0,1\}^n$,
    - • If both $g(x)=1$ and $h(x)=1$ then $P[f](x)=\beta^{-1}\circ\alpha^{-1}\circ\beta\circ\alpha=\gamma$
    - • If $g(x)=1$ and $h(x)=0$ then $P[f](x)=I_5\circ\alpha^{-1}\circ I_5\circ\alpha=I_5$
    - • If $g(x)=0$ and $h(x)=1$ then $P[f](x)=\beta^{-1}\circ I_5\circ\beta\circ I_5=I_5$
    - If $g(x)=0$ and $h(x)=0$ then $P[f](x)=I_5\circ I_5\circ I_5\circ I_5=I_5$
- So, $P[f]$ is an (I_5,γ) program for f of size $2k+2l\le 4\max\{k,l\}$

Construction: The OR Gates

- Let f be an OR gate
- Let g and h be the two gates providing input to f
- Assume inductively that we have already constructed

  an (I_5, γ) program $P[g]$ of size k for g

  and

  an (I_5, γ) program $P[h]$ of size l for h
- Construct a program T_0 from $P[g]$ such that, for every input x,

$$T_0(x) = \theta_3^{-1} \circ P[g](x) \circ \gamma^{-1} \circ \theta_3 \circ \gamma$$

- Then, since $P[g]$ is an (I_5, γ) program for g of length k, T_0 is a

$$(\theta_3^{-1} \circ I_5 \circ \gamma^{-1} \circ \theta_3 \circ \gamma, \theta_3^{-1} \circ \gamma \circ \gamma^{-1} \circ \theta_3 \circ \gamma)$$

- But $\theta_3^{-1} \circ I_5 \circ \gamma^{-1} \circ \theta_3 = \theta_3^{-1} \circ \gamma^{-1} \circ \theta_3 = \beta$, by our previously noted facts.
- Thus, $P[g]$ is an $(\beta \circ \gamma, \gamma)$ program for g of length k

Construction: The OR Gates (continued)

- Using this procedure, we construct

- T_0, an $(\beta \circ \gamma, \gamma)$ program of size k for g (as just described explicitly)

- T_1, an (α, I_5) program of size l for h by prepending $\gamma^{-1} \circ \theta_2$ and appending θ_2^{-1} to $P[h]$

- T_2, an (β^{-1}, I_5) program of size k for g by prepending $\gamma^{-1} \circ \theta_1$ and appending θ_1^{-1} to $P[g]$

- T_3, an (α^{-1}, I_5) program of size l for h by prepending $\gamma^{-1} \circ \theta_0$ and appending θ_0^{-1} to $P[h]$

Construction: The OR Gates (continued)

- Now define $P[f]$ to be the program that executes T_0, T_1, T_2, T_3 in that order
- Then, for every $x \in \{0,1\}^n$,
  - If both $g(x) = 1$ and $h(x) = 1$ then $P[f](x) = I_5 \circ I_5 \circ I_5 \circ \gamma = \gamma$
  - If $g(x) = 1$ and $h(x) = 0$ then $P[f](x) = \alpha^{-1} \circ I_5 \circ \alpha \circ \gamma = \gamma$
  - If $g(x) = 0$ and $h(x) = 1$ then $P[f](x) = I_5 \circ \beta^{-1} \circ I_5 \circ \beta \circ \gamma = \gamma$
  - If $g(x) = 0$ and $h(x) = 0$ then $P[f](x) = \alpha^{-1} \circ \beta^{-1} \circ \alpha \circ \beta \circ \gamma = \gamma^{-1} \circ \gamma = I_5$
- So, $P[f]$ is an (I_5, γ) program for f of size $2k + 2l \leq 4\max\{k, l\}$

Construction: Wrap-up

- For each gate f in C_n, we have constructed $P[f]$, a (I_5, γ) program for f

- Let s be the output gate of C_n, then $P[s]$ is our desired (I_5, γ) program for the output gate of C_n

- What is the size of $P[s]$?

- If f is an input or constant gate then the size of $P[f]$ is 1

- Otherwise, if f is an AND gate or an OR gate, the size of $P[f]$ is bounded by 4 times the maximum of the size of the programs for the gates that feed f

- Therefore, it follows by simple induction that if f is any gate of C_n, the the size of $P[f]$ is bounded by 4^m, where m is the depth of the sub-circuit rooted at f

Construction: Wrap-up (continued)

- Recall that $C = \{C_n\}_{n \geq 1}$ is a $O(\log n)$ depth family of circuits.

- Therefore, the size of C_n is $O(\log n)$

- So there exists some constant c such that $\text{depth}(C_n) \leq c \log n$

- But the sub-circuit rooted at r is all of C_n, so the depth of the sub-circuit rooted at s is therefore less than or equal to $c \log n$

- But we determined that the depth of the sub-circuit rooted at s was an upper bound on the size of $P[s]$

- Therefore, it follows that the size of $P[s]$ is at most

$$4^{c \log n} = 4^{\log n^c} = (2^{\log n^c})(2^{\log n^c}) = n^c \cdot n^c = n^{2c}$$

- Therefore, $P[s]$ is a polynomial size program deciding the length n part of L

Conclusion

- We have constructed, for an arbitrary n, a polynomial-size width-5 branching program deciding the length n part of L

- Therefore, $L \in 5-\mathrm{PBP}$

- So we have shown that $\mathit{nonuniform}-\mathrm{NC}^1 \subseteq 5-\mathrm{PBP}$

- **QED**

Bottleneck Turing Machines

- We can generalize the concept of a width-k branching program

- We do this by defining the width-k *bottleneck Turing machine*

- A width-k bottleneck Turing machine is a polynomial-time bounded deterministic Turing machine with two additional features:
  1. an auxiliary input called the counter that holds a binary string of length $p(|x|)$ (where x is the input on the tape, and p is just some polynomial)
  2. a device called the safe-storage that holds a value in $\{1,\ldots,k\}$

- When it comes to bottleneck Turing machines, we are very interested in the *change in the value of the safe-storage* from the beginning of the computation to the end. Let $f_{x,y}$ denote the function from $\{1,\ldots,k\}$ to $\{1,\ldots,k\}$ induced by a bottleneck Turing machine M on input x with counter value y. That is, $f_{x,y}(z)$ is defined to be the value in the safe-storage after we run M on input x with auxiliary input y and with z in the safe-storage

- Then we say that M accepts x if it is the case that

$$(f_{x,1^{p(|x|)}} \circ f_{x,1^{p(|x|)-1}0} \circ \ldots \circ f_{x,0^{p(|x|)-1}1} \circ f_{x,0^{p(|x|)}})(1) = 1$$

Bottleneck Turing Machines: The Symmetric Version

- A *symmetric* bottleneck Turing machine is a bottleneck Turing machine whose computation is required to yield the same result regardless of in what order we perform the steps (that is, regardless of what order we compose the functions $f_{x,y}$)

Bottleneck Turing Machines: The Intuition

- To determine where to move our pebble, we put its current location in the safe-storage, put the input string x on the tape, and put the number of the current step in the auxiliary input. Then we start the machine going, and when the computation completes, the value in the safe-storage is the new location for our pebble.

- The intuition is that, where a width-k branching program is like having a set of cards for each input length, a bottleneck Turing machine is like having a machine that takes the input string, the location of the pebble, and the number of the current step, and tells us where to move the pebble

- As always, the pebble starts on square one

- After using our machine for exactly $2^{p(|x|)}$ steps, we have accepted if the pebble is on square one, and rejected otherwise

Bottleneck Turing Machines: Some Results

- SF_k is the class of all languages that are decided by some width-k bottleneck Turing machine

- SSF_k is the class of all languages that are decided by some width-k *symmetric* bottleneck Turing machine

- Here are some results related to bottleneck Turing machine classes:

  - $SF_5 = PSPACE$
  - $SF_2 = \oplus P \,//\, OptP$
  - $SSF_2 = NP$ disjoint union $\oplus P$
  - $ProbabilisticSSF_2 = NP^{PP}$

- We will not have time to prove these, however.

The End