(1)

Defined, $\text{PolyL} := \bigcup_{c>0} \text{SPACE}(\log^c n)$

SC := polytime and poly log space

Is PATH $\in$ SC ?

Part-I: Savitch's theorem gives a constructive argument to show $\text{NSPACE}(S(n)) \subseteq \text{DSPACE}(S^2(n))$.

But the algorithm suggested in the theorem has time complexity $2^{\Omega(S^2(n))}$.

It is known, PATH $\in$ NL = NSPACE($\log n$)

$\Rightarrow$ PATH $\in$ DSPACE($\log^2 n$) [Savitch]

As per Savitch, Runtime is $2^{\Omega(\log^2 n)}$

This is Quasi-polynomial.

Hence, not sufficient to show PATH $\in$ SC.

Part-II: No, SC $\subseteq$ PolyL $\cap$ P [By definition]

But PolyL $\cap$ P contains those languages also that is decided by two separate Turing machine TM_1 and TM_2, where TM_1 is polytime and TM_2 is poly log space machine.

SC requires a single TM having both feature.

(2) Certificate definition of NL.

$L \in NL$ if there is a log-space verifier 'M' & a poly-function 'q' s.t.

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x,u)=1$$

Constrain: where 'u' is on read-only tape.

Claim: if 'u' is on regular tape where back and forth motion for head is allowed. Then class defined is equal to NP.

Strategy:
- We show 3-SAT is the member of class (I) → defined in the above way.
- Every problem in NP is log-space reducible (II) → to 3-SAT.

Part (I): Let, $x \in 3SAT$

Example: $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_5 \vee x_6) \wedge (\cdots)$

assume certificate: 10101...

Verification by log-space TM

→ It starts with first clause, look at the first literal store its location on log sized pointer. Go to certificate tape. Read the value of corresponding literal on certificate tape.

→ Come back to work tape and note down its value. Do the same for other literals in the current clause. Finally evaluate the value of the clause.

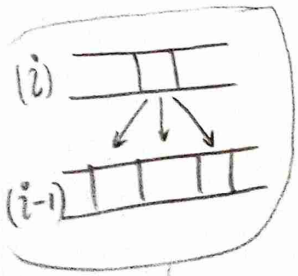
→ Repeat the process for next clause. Evaluate its value using log-space like before.
Keep adding the output of current clause to the previous. In this way, we don't need to store all outputs of clauses.

→ This show 3-SAT is in the newly defined class.

Part (II): 3-SAT is NP-complete under log-space reduction.

if $L \in NL \Rightarrow L \leq_L 3\text{-SAT}$

→ We recall in case of poly time reduction, local nature of computation was utilized. Previous three cells were used to compute the current cell. This operation was replaced by a CNF (circuit).



→ The main challenge here is to come-up with log-space computable function $f : x \rightarrow \phi_x$ s.t

$$x \in L \iff \phi_x \in 3\text{-SAT}.$$

more precise,

$$\exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1 \iff \phi_x \text{ is satisfiable}$$

→ Given 'x' will be hardwired in the constructed circuit. 'u' will become input to circuit.

→ The function 'f' is log-space computable as it need to keep track of a finite number of pointer to do the local computation. Each pointer take log-space

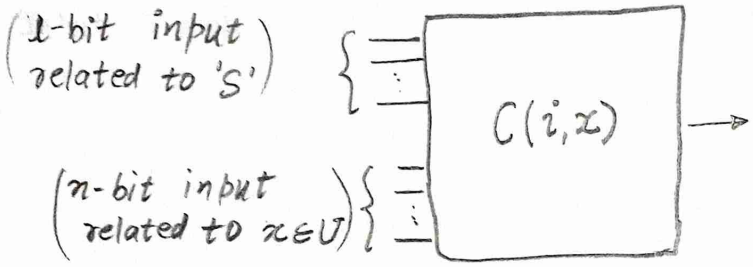
③

Given,

set $S = \{S_i\}_{i=1}^{m} \subset U$

VC(S) is an integer equal to Cardinality of set $X \subseteq U$ s.t for every subset $X' \subseteq X$ there is 'i' s.t $S_i \cap X = X'$

Given Boolean Circuit C



$C(i,x) = 1$ only for set of elements $x \in U$ that is member of set S_i.

So, we are given access to this circuit ank integer 'k'.

Language VC-DIMENSIONS =

$\{ \langle C, K \rangle$: C represents a collection S such that $VC(S) \geq K \}$

Proof strategy : we will analyze an explicit polytime computable predicate with alternating quantifier of form: $\exists \cdot \forall \cdot E$. We show the logical statement evaluates to 1 iff the input belongs to the language.

Logical statement:

$$VC(S) \geq K \iff$$

ⓐ $(\exists x_1, \ldots, x_k \in \{0,1\}^n)$

ⓑ $(\forall ss \in \{0,1\}^k)$

ⓒ $(\exists i \in \{0,1\}^L)$

$(\forall j \in \{1, \ldots, k\})$

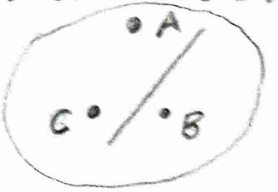
ⓓ $M[C(i,x) = ss[j]] = 1$

Explanation:

part ⓐ demands if there exists a set $\{x_1, \ldots, x_k\} \subseteq U$

It is not required that all set of size k shall fulfull upcoming requirements. [shattering by S]

For example VC-DIM of Binary classifier in plane is 3 even though there exist a set that can't be shattered by the classifier.



It is possible to classify $\{A, C\}$ and $\{B\}$ by line

No line can classify as $\{A, C\}$ and $\{B\}$

part ⓑ: Once we have the set $\{x_1, \ldots, x_k\}$, we check shattering condition over all its subsets x'.

It has 2^k subsets. we index them as

$\{x_1\} = 1000\cdots$

$\{x_1, x_2\} = 1100\cdots$ and so on

Hence, $\{0,1\}^k$ contain all the string indexing the subsets.

part ⓒ : Now,

for each subset x' we search if there exists S_i s.t $S_i \cap X = x'$

$\exists i \in \{0,1\}^l$ capture this requirement.

part ⓓ

as mentioned in part ⓑ,

a given string $ss \in \{0,1\}^k$ that index subset of x. we check via turing Machine if j^{th} bit of ss matches with Circuit output $C(i, x_j)$.

Example; Suppose, $ss = 100\cdots0 = \{x_1\} \subseteq X$

assume $S_2 \cap X = \{x_1\}$

then, $C(2, x_1) = ss[1] = 1$

$C(2, x_2) = ss[2] = 0$ and so on.

Any mismatch implies the circuit doesn't represent collection 'S'.

→ Runtime of this check is polynomial in input size.

→ Using the fact that $VC(S)$ has an upper bound. This is logarithm of (number of different classifier it could return).

$$VC(S) \leq \log|S| = l$$

→ Hence, TM need to check over those 'k' where $k < l$.

→ Also, n, k, l used in the Logical statement are also bounded.

Finally we have three alternating quantifiers given by ⓐ, ⓑ and ⓒ. There is a Turing Machine that runs in poly time and carry out this "Verification-like" process.

This shows VC-DIM $\in \Sigma_3$ in PH.

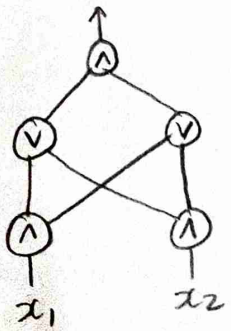
④ Definition of NC^1 ⇒ poly-size and log-depth circuit family $\{C_n\}$; C_n is Boolean circuit
[Definition 6.24, Arora, Barak]

In question to show
$L \in NC^1 \Leftrightarrow$ L is decided by a poly-size circuit family $\{C_n\}$; C_n is Boolean function.

→ First we show a poly-size and log-depth circuit can be converted to a poly-size Boolean formula.
Since, formula is a special circuit with fan-out of one.

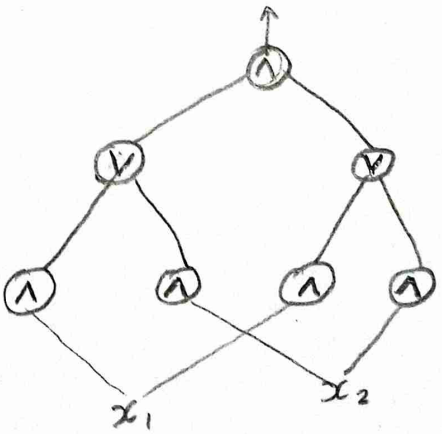
→ Method to change a circuit to formula:
"Unwinding" the circuit graph into a tree:
check how many paths exists between a certain gate 'g' (say) to the output gate. Now duplicate that many gate 'g'.

Example:



Circuit

⇔



Formula: Tree

Note: The depth remain conserved in this process which implies the blow up in size is atmost by a polynomial factor.

→ Now, it remains to show if any poly-size formula without any depth restriction can be equivalently converted to a poly-size and log-depth formula.

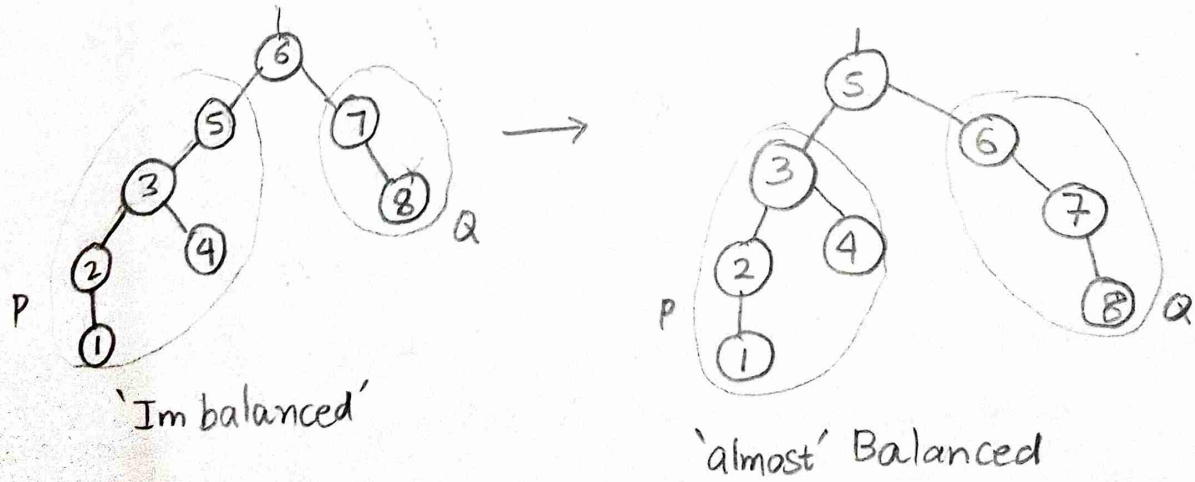
→ This neccessarly implies the class of language computed by poly-size formula remains same with and without log-depth restriction.

→ Indeed there is a technique to convert a poly-size formula (without log-depth) to a poly-size formula with log-depth. [claim-I]

→ Proof of Claim-I by formula "Re-Balancing" method.

Re-Balance step-I: first a vertex that splits the computation tree into "almost" equal part. [meaning of 'almost' discussed later]

Ex:



Meaning of word 'almost': If the size of formula is 's' then we choose that vertex which splits the tree into two subtree 'P' and 'Q' such that $|P|, |Q|$ lies $[\ s/3\ ,\ 2s/3\ ]$

→ The fact is there always exists such a vertex if we follow the below algorithm.

(1.) Take any vertex and check the sizes of its subtrees.

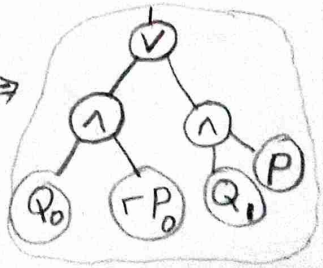
(2.) If they lie between $[S/3, 2S/3]$; then exit

(3) Otherwise go into the subtree having size $> \frac{2S}{3}$. Repeat the step ① and ② untill success

The algorithm must converge/stop, as we are decreasing the size of larger subtrees in each iterations.

'Re-Balance' step-II: Step-I gives two almost balanced subtree 'P' and 'Q'. Now the crucial idea is to seperately run computation of 'P' and 'Q' sub-parts to simulate the entire computation of original tree. without loss of generality assume 'Q' depends on 'P' (But not vice-versa). There could be two possible outcome of 'P'. Let, if $P=0$, Q evaluates to Q_0. other Q evaluates to Q_1. Now, we build a selector formula that selects the appropriate copy of Q based on the computation of P.

Or, $f_{selector} = (Q_0 \wedge \neg P) \vee (Q_1 \wedge P) \Leftrightarrow$



It creates extra copy of 'Q', but also ensures depth reduction. This is fine, if blow in size is polynomial.

('Re-Balance' step-III): Now, we recursively re-balance P and Q_1. Then we plug the result to the tree which further increase depth of the tree.

Analyse: we calculate maximum number of such recursive call. Since on each call the size of subparts to be considered next decreases by $2/3$ (atmost). Hence atmost $\log_{3/2}(S)$ recursive call required.

Notice that while building selector formula, we add a constant number of depth to the tree/formula. Since this goes for $O(\log(S))$ times. Hence, it has log depth.

Rough

$S, S\cdot\frac{2}{3}, S\left(\frac{2}{3}\right)^2 \cdots S\left(\frac{2}{3}\right)^i$

$S\left(\frac{2}{3}\right)^i \sim 1 \Rightarrow i = \log_{3/2}(S)$

(5) **Linear Programming is P-complete**

Strategy: We take any language $L \in P$, then show its log space reduction to linear programming (LP).

More details: we define LP and explain the meaning of log space reduction.

Linear program (LP): (over variable x)

$$\text{maximize } (c^T x) \longrightarrow \text{objective}$$

$$\text{S.t } Ax \geq b \longrightarrow \text{Constraints}$$

where, c, x and b are vectors

A is a matrix

Elements of c, b and A belongs to $\{-1, 0, 1\}$

Note: This is not an Integer program. We put no restriction on elements of vector x. Due to construction of LP, it will output integers.

Log space reduction:

Let a language $L \in P$. Hence, a polytime TM decides L.

If $x^* \in L \Leftrightarrow$ $x^* \rightarrow$ [TM] $\rightarrow 1$

Runtime: $t(n)$ (say)

→ Now we claim there is a log space computable function 'f' S.t

if

$$x^* \in L \Leftrightarrow f(x^*) \rightarrow \boxed{\begin{array}{c}\text{Linear Program}\\ \text{Solver}\end{array}} \rightarrow 1$$

function 'f' takes x^* and output system of constraints

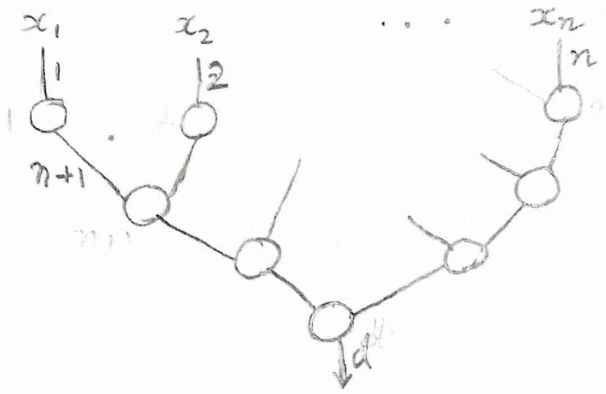
TM corresponding to 'f' can use logarithmic amount of space.

Any TM running in $t(n)$ time has an equivalent circuit of $t^2(n)$ size. [size = no of gates] [Cook Levin]

Without loss of generality lets take {OR, AND} gate set as they can represent any other gates with constant over head.

Circuit(with input) to LP conversion :

step-1 : index each wire of the circuit



$d = t^2(n)$ (say)

step-II : Visit each gate at a time and output the constrain imposed by it to variable of wires

In the work space, we need to store pointer to the current gate location, i.e, $\log(|d|)$ bit space required.

step - III :

→ Let the output of ith gate be x_i

$$0 \le x_i \le 1 \Rightarrow \begin{cases} x_i \ge 0, \text{ and} \\ -x_i \ge -1 \end{cases}$$

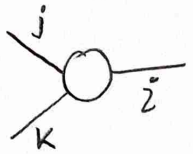
→ At input wire, constrain is

$$x_i = x_i^* \Rightarrow \begin{cases} x_i \ge x_i^* \text{ and} \\ -x_i \ge -x_i^* \end{cases}$$

→ For every NOT Gate

$$x_i = 1 - x_j \Rightarrow \begin{cases} x_i + x_j \ge 1 \text{ and} \\ -x_i - x_j \ge -1 \end{cases}$$

→ For every AND gate



$$\equiv \begin{cases} x_j - x_i \geq 0 \\ x_k - x_i \geq 0 \\ x_i - x_j - x_k \geq -1 \end{cases} \leftrightarrow$$

Consistent with AND truth table

We notice for each of the conversion, it need to store only current gate location, Output the constrain equation that requires information of local wire variable. Then move to next node and update pointer. It never need to store any other information on work tape apart from as discussed above.

Basically the system of constraints form a polyhedra.

LP necessarily maximizes the value of objective: x_d

over the polyhedra.

Due to constauction of the Linear program, $[x_i = x_i^* \in L]$

This polyhedra contain single element either 1 or 0,

based on if input x_i^* is in language or not (respectively).

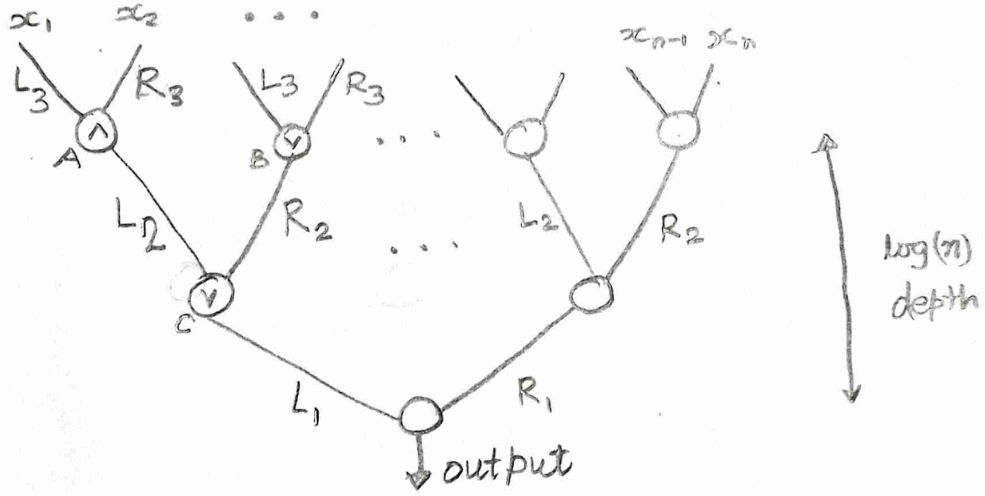
⑥ @ Logspace Uniform $NC^1 \subseteq L$

↓

It consists of those languages that is decided by a circuit family $\{C_n\}$, each of size $O(n^c)$ and depth $O(\log n)$.

Strategy: We show that on taking any circuit from above description, that circuit can be evaluated using a logspace TM.

This is sufficient to show $NC^1 \subseteq L$



Above graph is an example to reinforce the algorithm to compute circuit value using logspace.

Take a path starting from the output node to input node. This is done via depth-First-search. Path length is atmost log(n). We evaluate the gates in the path and store them in an another log(n) bit space as:

$\underbrace{L_1\ L_2\ L_3 \cdots}_{\log(n) \text{ bit}}$ (say) for the path

$\underbrace{V_1\ V_2\ V_3 \cdots}_{\log(n) \text{ bit}}$ (say) for value of gate output in the path

→ The main goal is to optimize space rather than runtime of the algorithm. So, we might revisit the previously visit path if required.

→ while evaluating n gate output at some node, we require input value from both wire. If value from one of the wire is unknown, we change path accordingly to estimate it.

Example

→ Say input $x_1 = 1$, $x_2 = 1$, we get output at gate-A [see picture].

To get output at gate-C, we need output of gate-B, hence change path as:

$(L_1 \; R_2 \; L_3)$ — Bottom to Up

$(V_1 \; V_2 \; 1)$ — ↑ value of gate-A

Once we evaluate gate value in new path, i.e, value of Gate-B, we feed it to Gate-C. This output is stored V_2. Now space for V_3 can be reused.

→ This process goes recursively. We never need to store more space than what is described above to output whole circuit value for given input.

6 (b)

$NL \subseteq NC$

Strategy: We take the hardest problem in NL.
PATH is NL-complete (under log-space red.)

We show $PATH \in NC^2$

$\Rightarrow PATH \subseteq NC$ [as $NC^2 \subseteq NC$]

→ In s-t path problem we have graph as adjacency matrix and two vertices 's' and 't'. we see an explicit algorithm to show s-t path can be decided by $O(\log^2 n)$ depth Circuit of poly(n) size.

→ Some useful fact about graph adjucancy matrix

Let 'A' be adjucancy matrix

if entry $a_{ij} = 1 \Rightarrow$ There is a path between vertices V_i and V_j in graph 'G'.

If $A^n = A'$ and entry $a'_{ij} = 1 \Rightarrow$ there is a path of length 'n' between vertex V_i and V_j.

→ Now we see how to estimate element a'_{ij} of A^n matrix using $O(\log^2 n)$ depth circuit.

There is two key facts that need to be shown

Claim-1: • $A^2_{n \times n}$ can be computed using log(n) depth Circuit and poly(n)-size circuit.

Claim-2: • A^n can be Computed using repeated squaring using log(n) steps.

Proof for claim-I :

Without loss of generality assume

$$P_{n\times n} = A_{n\times n} \cdot A_{n\times n}$$

hence,

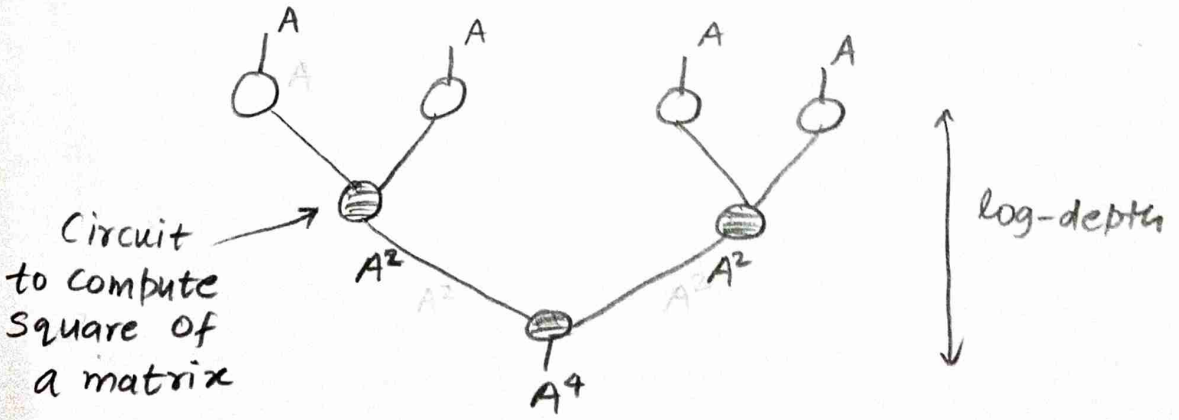
$$p_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}.$$

it requires $O(n)$ multiplication and addition that can done using poly(n) size and log depth circuit. [addition and multiplication is in NC^1]

This holds true for all n^2 entries of P_{ij}.

The depth remain $O(\log n)$ just nodes in the graph increase by n^2 factor. [or, we still need $O(\log n)$ time but $O(n^3)$ processor].

Proof of claim-II: A^n in $O(\log(n))$ steps using repeate Squaring.



→ Thus above construction implies $O(\log n)$ circuit to compute A^n. where each node compute square of input matrices using $O(\log n)$ depth circuit.

Total depth ≡ $O(\log^2 n)$

Finally we use the fact that element $a'_{st} = 1$ in A^n matrix implies a s-t path of length 'n' in the graph 'G'.

Also, s-t path length never be greater than number of vertices in the graph. This serves as stopping criteria.