• $\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$

for $f(n) \geq \log_2 n$

Def $G_{M,x}$: each vertex is a configuration of M on input x (i.e. contents of 1st tape is x)

$2^{O(S(n))}$ vertices for $S(n) \geq \log_2 n$

start configuration: $C_0 = (q_0 x, \sqcup)$

tape 1 tape 2

edge $C \to D$ iff $C \vdash_M D$ "yields in one step"

out-degree $\leq b$

$$C \to D_1,\ D_2,\ \ldots,\ D_b$$

for b some constant depending on δ function of M

without loss of generality, there is a unique accepting configuration

$C_{accept} = (q_{acc} x, \sqcup)$

(simply have M clean up everything before accepting).

Note: • M accepts $x \iff \exists$ a path from C_0 to C_{accept} in $G_{M,x}$ (of length $2^{O(S(n))}$)

• M deterministic $\Rightarrow G_{M,x}$ has outdegree 1

Thm [Savitch] $S(n) \geq \log_2 n \Rightarrow$ $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{SPACE}(S^2(n))$

Proof idea We search for path from C_0 to C_{accept} in $G_{M,x}$ but don't write down the whole graph.

We know that if there is such a path then it has at most

$$T \leq 2^{d\,S(n)}$$

steps for some d.

Let's pretend we know T:

$$\overbrace{C_0 \xrightarrow{\leq T/2} C_{mid} \xrightarrow{\leq T/2} C_T}^{\leq T}$$

Then some C_{mid} as above exists.

Define function $\text{CANYIELD}_t(C,D) = \begin{cases} \text{true} & \text{if } C \vdash_M^* D \text{ using } \leq t \text{ steps} \\ \text{false} & \text{otherwise} \end{cases}$

configurations

not writing x since it is the same for all nodes in $G_{M,x}$

Observe that we have the following recursive properties

$$\text{CANYIELD}_0(C,D) \text{ iff } C = D$$

$$\text{CANYIELD}_1(C,D) \text{ iff } C \to D \text{, i.e.: } C \vdash_M D$$

(or $C = D$)

Algorithm: can check using δ function of M

CANYIELD$_t$(C, D) iff $\exists$ C_{mid} (config on input x)
s.t. CANYIELD$_{t/2}$(C, C_{mid})
$\wedge$ CANYIELD$_{t/2}$(C_{mid}, D)

Algorithm: Try all possible C_{mid} and use recursive calls

Space for first call reused for second call

Goal: Compute CANYIELD$_T$(C_0, C_{accept})

Space used for recursive algorithm

Total $O(S^2(n))$

\# of levels : $\log_2 T$ which is $O(S(n))$

each level of call stack:

C, D, T so $O(S(n))$

$O(S(n))$ for C, D; \# of bits of T: $O(S(n))$

Other space used at each call level
$O(S(n))$ for C_{mid}

To do this we assumed that we knew T
But we don't actually need that

We modify the above to try all possible $T = 2^{d_i}$
using $S = 1, 2, \ldots$ memory cells

Run above CANYIELD alg with above

Keep track of whether TM actually ever tries a rightward move when on last cell of work tape

If $\text{CANYIELD}_T(C_0, C_{accept})$ is true then accept
if no path found but a rightward move tried, then increase S, T
if no path found and no rightward move then reject

Total: $O(S^2(n))$ space & correct

Note: time used is worse than $2^{O(S(n))}$!
It is $2^{O(S^2(n))}$ but we only focus on space $\boxed{Z}$

$$PSPACE = \bigcup_k SPACE(n^k)$$

$$NPSPACE = \bigcup_k NSPACE(n^k)$$

Cor $\quad NPSPACE = PSPACE$

Pf $\quad NSPACE(n^k) \subseteq SPACE(n^{2k})$ $\blacksquare$

Example:

Thm $\quad EQ_{NFA} \in PSPACE$

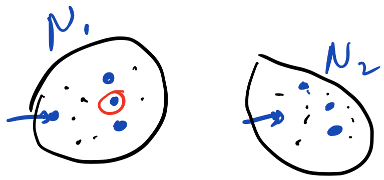
Proof Converting two NFAs to DFAs would take exponential space

We use the fact that PSPACE = NSPACE

and PSPACE closed under complement

It suffices to show that $\overline{EQ_{NFA}} \in$ NPSPACE

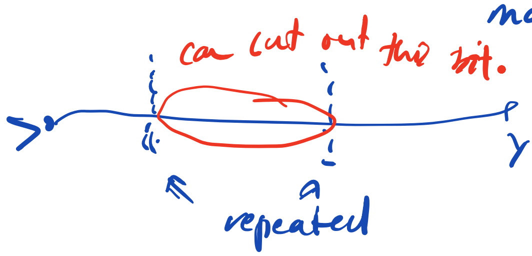
On input $\langle N_1, N_2 \rangle$ where N_1, N_2 are NFA's with state sets Q_1, Q_2 respectively



$L(N_1) \neq L(N_2) \iff \exists$ string y s.t. set of states reachable in N_1 on input y contain a final state of N_1 but set of states reachable in N_2 on input y does not (or vice versa)

<u>Claim</u> If such a y exists then one of length $\leq 2^{|Q_1| + |Q_2|}$ exists

<u>Pf of claim.</u> If y is longer than one of the sets of states reachable in the two machines repeats



Idea: Use nondeterminism to guess y.
But: y is too long to write down in only $n^{O(1)}$ symbols

Idea: Unlike Time-bounded NTMs can't convert space-bounded NTMs to guess first form
- Instead guess y symbol-by-symbol and don't write down the whole thing

Algorithm On input $\langle N_1, N_2 \rangle$
start at q_0^1, q_0^2 states of N_1, N_2
For $2^{|Q_1|+|Q_2|}$ steps
Guess next symbol of y keeping track of current set of states reached so far on y in both N_1, N_2
if one of these sets but not the other contains an accepting state then accept

If $\leq 2^{|Q_1|+|Q_2|}$ steps reached but not accepted then reject

Storage
- $|Q_1|+|Q_2|$ bits for sets of states reached
- $|Q_1|+|Q_2|$ bits for a timer.

This is $O(n)$

∎

Now $P \subseteq NP \subseteq PSPACE \subseteq EXP$
$P \neq EXP$ (proof later) but all other containments conjectured to be $\subsetneq$ (open)

Is $P = PSPACE$? If so then $P = NP$
Proving $P \neq PSPACE$ may be easier to prove than $P \neq NP$...

PSPACE contains problems we think are even harder than NP-complete problems.

Defn B is PSPACE-hard iff $\forall A \in PSPACE$, $A \leq_m^P B$.

Defn B is PSPACE-complete iff
- $B \in PSPACE$
- B is PSPACE-hard

Let φ be a Boolean formula in vars $x_1, \dots, x_n$

$\langle\varphi\rangle \in SAT \iff \exists x_1 \dots \exists x_n\, \varphi(x_1, \dots, x_n)$ is true

$\langle\varphi\rangle \in TAUT \iff \underbrace{\forall x_1 \dots \forall x_n\, \varphi(x_1, \dots, x_n) \text{ is true}}_{\text{fully quantified Boolean formula}}$

Defn $TQBF = \{\langle\Phi\rangle : \Phi$ is a fully quantified Boolean formula that is true$\}$

quantifiers may alternate

e.g. $\exists x_1 \forall x_2 \exists x_3 ((x_1 \to x_2) \wedge (x_2 \to x_3) \wedge (x_3 \to x_2))$

true. $x_1 = 0$, $x_3 = x_2$

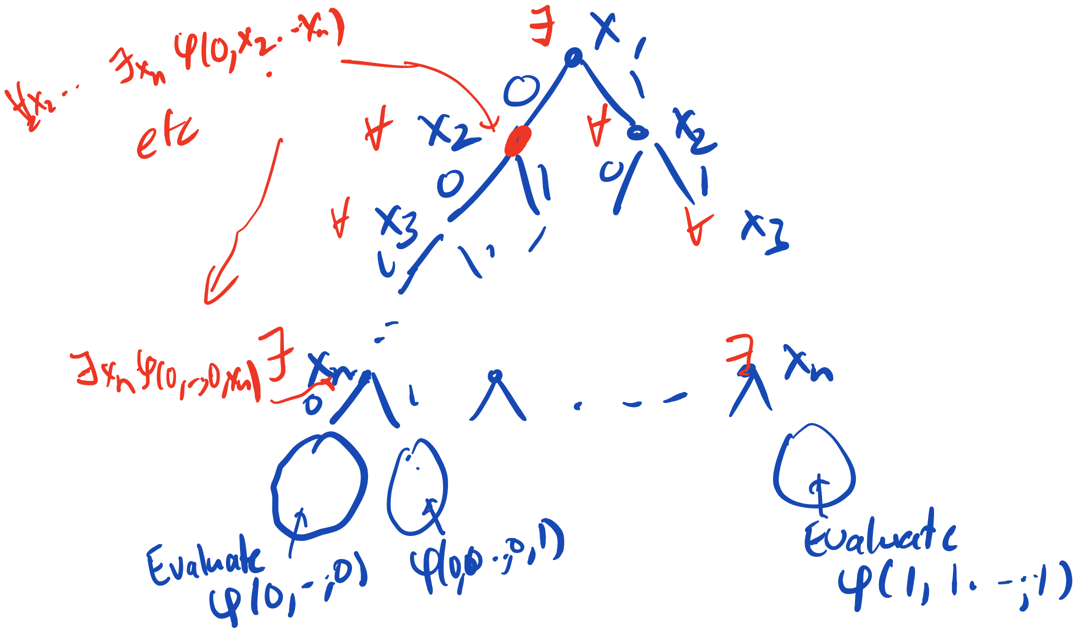
Thm TQBF is PSPACE-complete

Proof 1. Claim: TQBF $\in$ PSPACE

Write $\Phi = Q_1 x_1 \cdots Q_n x_n\, \underbrace{\varphi(x_1 \cdots x_n)}_{\text{Boolean formula on } x_1 \cdots x_n}$

quantifier $Q_i = \exists$ or $\forall$

Imagine a full binary tree on the assignments to $x_1 \cdots x_n$



Consider an alg that does a (recursive) DFS on this tree evaluating the formula:

The value at the leaf is easy poly time to compute.
We can evaluate each node as we backtrack from the DFS

If x_i is labelled by $\exists$:
evaluate left-child
if left-child's value is 1 return 1
else evaluate right child and return its value

If x_i is labelled by $\forall$:
evaluate left-child
if left-child's value is 0 return 0
else evaluate right-child and returns
its value

What storage is required:
DFS stack: height n
Enough to evaluate φ at a leaf
$\langle\varphi\rangle$
Total $n + |\langle\varphi\rangle| \Rightarrow$ linear space

2) <u>TQBF is PSPACE-hard:</u>

Let $A \in$ PSPACE
$\therefore A$ is decided by some TM M using
space $S = cn^k$ for some constant c, k

Recall: $x \in A \Leftrightarrow \exists$ path from C_0 to C_{accept}
in $G_{M,x}$
(Configuration graph of M on
input x)

- $G_{M,x}$ has at most
$T = 2^{dS}$ nodes
- each node of $G_{M,x}$ is a
configuration of M on input x
and can be described
by $\underbrace{O(S)}_{O(n^k)}$ bits.

Recall $\text{CANYIELD}_t(C,D)$ configuration of M on input x

$\equiv$ there is a path from C to D in $G_{M,x}$ of length $\leq t$.

$\text{CANYIELD}_0(C,D) \equiv$ "$C=D$"

$\text{CANYIELD}_1(C,D) \equiv$ "$C=D$" or "$C \vdash_M D$" "yields in one step"

$$\text{CANYIELD}_t(C,D) \equiv \exists C_{mid}.(\text{CANYIELD}_{t/2}(C,C_{mid}) \wedge \text{CANYIELD}_{t/2}(C_{mid},D))$$

We prove $A \leq_m^P \text{TQBF}$

<u>Goal:</u> $x \in A \xrightarrow{f} \langle \Phi_{M,x} \rangle$

where $\Phi_{M,x} = 1$ iff $\text{CANYIELD}_T(C_0, C_{accept})$

We will define formulas $\Phi_t(\vec{C},\vec{D})$ s.t.

$\Phi_t(\vec{C},\vec{D})$ iff $\text{CANYIELD}_t(C,D)$

where $\vec{C},\vec{D}$ are binary vectors of variables corresponding to configs C,D

Since space is $\leq S$, $\vec{C},\vec{D}$ take $O(S) = O(n^k)$ bits.

We will set $\Phi_{M,x} = \Phi_T(C_0, C_{accept})$ constant bit-vectors each representing specific configuration.

$\Phi_0(\vec{C},\vec{D})$ is an $\wedge$ of $O(S)$ conditions of the form $(\vec{C})_i = (\vec{D})_i$

$\Phi_1(\vec{C},\vec{D}) = \Phi_0(\vec{C},\vec{D}) \cup$ "$C \vdash_M D$"

easy to express in logic with δ function

(just like adjacent rows in Cook-Levin tableau)

Assume wlog that we only define Φ_t when t is a power of 2.

Obvious attempt based on CANYIELD$_t(C, D)$

$$\cancel{\Phi_t(\vec{C},\vec{D}) = \exists \vec{C}_{mid} \left(\Phi_{t/2}(\vec{C},\vec{C}_{mid}) \wedge \Phi_{t/2}(\vec{C}_{mid},\vec{D}) \right)}$$

$O(S)$ $\exists$ quantifier in a row for the bits of C_{mid}

When we unwind this recursion we realize that Φ_t

size$(\Phi_t) > 2$ size$(\Phi_{t/2})$

So size$(\Phi_t) > t$ which will be bad for Φ_T since T is exponential and we need to computable in polytime

But we haven't used any $\forall$ in this!

Our new idea will be to write $\Phi_{t/2}$ just once and use the $\forall$ quantifier to cover the two cases:

Define $\Phi_t(\vec{C},\vec{D}) = \exists \overrightarrow{C_{mid}}.\ \forall \vec{E},\vec{F}$

the two cases we care about

$$\left[\left(\left((\vec{E}=\vec{C}) \wedge (\vec{F}=\overrightarrow{C_{mid}}) \right) \vee \left((\vec{E}=\overrightarrow{C_{mid}}) \wedge (\vec{F}=\vec{D}) \right) \right) \rightarrow \Phi_{t/2}(\vec{E},\vec{F}) \right)$$

Now $\text{size}(\Phi_t) = cn^k + \text{size}(\Phi_{t/2})$

$\therefore \text{size}(\Phi_t) = (cn^k) \cdot \underbrace{\log T}_{O(n^k)} + cn^k$

$\therefore \text{size}(\Phi_t)$ is $O(n^{2k})$ which is polynomial

Φ_t is very easy to write down
- everything but Φ_1 doesn't even depend on the details of M

$\therefore f$ is polynomial

By construction it satisfies correctness ∎

Nope: complexity classes inside P.
Is every problem in P solvable in small space?

Logarithmic Space

Consider the following non-regular language

$$A = \{0^n 1^n : n \geq 0\}$$

Thm $A \in \text{SPACE}(\log n)$

TM deciding A: On input x

Space: two counter up to length of input

$O(\log n)$ bits

- Count # of 0's at start before first 1
- Count # of 1's next.
- If counts differ or there are more characters before 1st blank reject
- else accept.

Let $L = SPACE(\log n)$ $\therefore A \in L$ $\blacksquare$

$NL = NSPACE(\log n)$

$$L \subseteq NL \subseteq TIME(2^{O(\log n)}) = P \subseteq NP$$

Open: Power of nondeterminism.

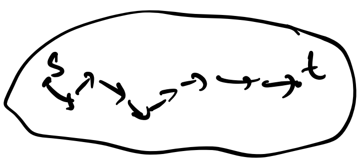
- Is $L = NL$?
- Is $L = P$ or $L = NP$?

Recall $PATH = \{\langle G, s, t\rangle : G$ is a directed graph with a path from s to $t\}$

Thm $PATH \in NL$

Proof idea: "Guess and verify a path of length $\leq n$ (# vertices) from s to t, one vertex at a time"

not enough space to actually write down the path.

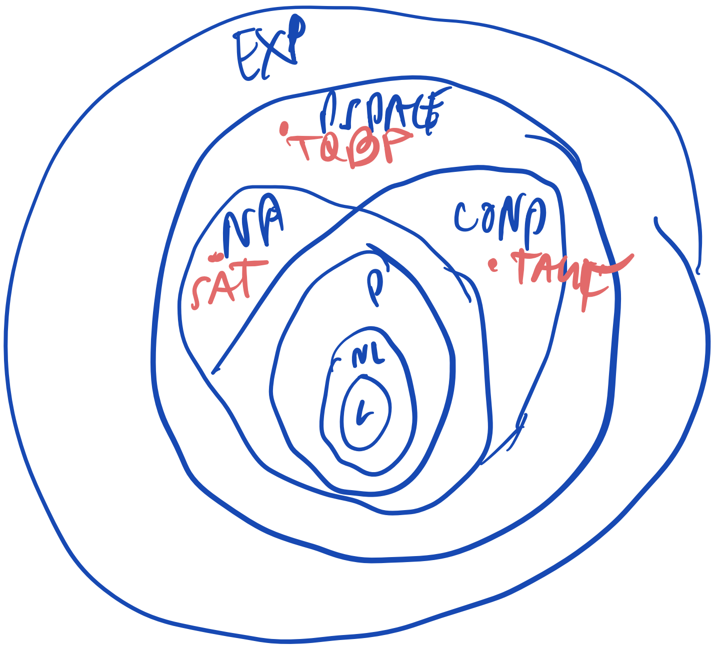


Keep track of: counter for the length — $\log_2 n$ bits

current vertex (and next vertex) — $O(\log_2 n)$ bits

NTM:

count $\leftarrow 0$
curr $\leftarrow s$
while count $\le n$ and curr $\ne t$ do {
  Guess next vertex (neighbour) v of curr
  Check if (curr, v) is an edge
  If not then reject
  else curr $\leftarrow v$. }
If curr $= t$ then accept
  else reject



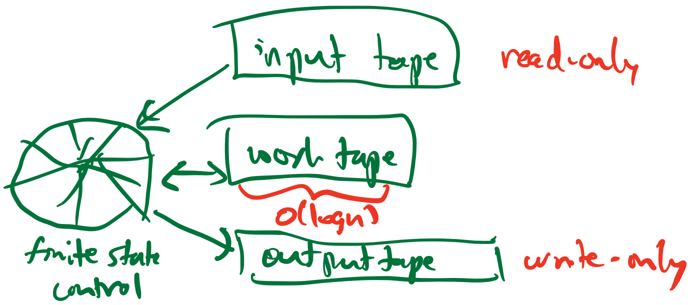
To study these questions we need a finer notion of reduction than $\le_m^P$ which allows polynomial slack

For this we need a notion of log-space computable functions

We modify our 2-tape space bounded notion of TM to a 3-tape TM like this:

Def^n f is <u>logspace-computable</u> iff f is computable by a TM of the following form

"logspace transducer"



Def^n $A \le_m^L B$ iff $A \le_m B$ via reduction f that is logspace-computable

Def^n B is <u>NL-hard</u> iff $\forall A \in NL$, $A \le_m^L B$

Def^n B is <u>NL-complete</u> iff (1) $B \in NL$ (2) B is NL-complete

Def^n PATH is NL-complete

Proof (1) $PATH \in NL$

(2) Let $A \in NL$. <u>Claim</u> $A \le_m^L PATH$

$$\begin{array}{ccc} A & & PATH \\ x & \xrightarrow{f} & \langle G_{M,x}, C_0, C_{accept} \rangle \end{array}$$

where M is logspace NTM deciding A

$C_0 = (q_0 x, \sqcup)$

$C_{accept} = (q_{accept}\, x, -)$

$x \in A \iff$ there is a path in $G_{M,x}$ from C_0 to C_{accept}.

$G_{M,x}$ is size $2^{O(\log n)}$ which is polynomial

Why is f logspace-computable?

- each configuration/vertex of $G_{M,x}$ takes $O(\log n)$ space so C_0, C_{accept} easy

Producing $G_{M,x}$:

Adjacency list form:

For all configurations C (in lexicographic order, not necessarily reachable)

Output C followed by all next configurations D_i s.t. $C \vdash_M D_i$ (i.e. $C \to D$)

based on δ function of M (builtin)

i.e. $C : D_1, \ldots, D_b$;

vertex out-neighbors

only need to store a constant # of configurations.

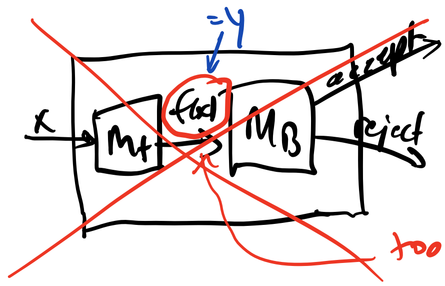
$\therefore O(\log n)$ space $\blacksquare$

We also need nice properties of $\leq_m^L$ to make this useful:

We still need to prove properties of $\leq_m^L$ that were easy for $\leq_m$ and $\leq_m^P$ but are tricky for $\leq_m^L$:

Thm
- If $A \leq_m^L B$ and $B \in L$ then $A \in L$
- If $A \leq_m^L B$ and $B \in NL$ then $A \in NL$
- If $A \leq_m^L B$ and $B \leq_m^L C$ then $A \leq_m^L C$

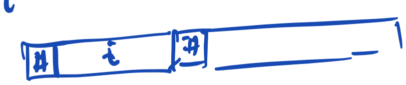
Proof Usual method



too long to write down for O(log n) space

Instead:

Modify M_B: If M_B is looking at y_i we have M_B also keep track of the input head position i



Change M_f by removing its output tape
New machine for A will "call" M_f with index i (x is still on input tape, i is on the work tape)
Each time it does it will run M_f ignoring its output except for the ith bit of output

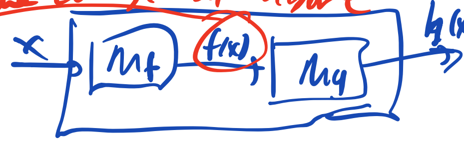
M_f will need to keep track of the # of bits output so far, j.

Re-run M_f each time step of M_B to find out the value of y_i

Total space: Space for M_f
Space for M_B
$+O(\log n)$

Note: $|f(x)|$ is $n^{O(1)}$ if $|x|=n$
$\therefore \log|f(x)|$ is $O(\log n)$
so still $O(\log n)$ space total

Note: • same construction works for NL case.
• For $A \le_m^L B$ and $B \le_m^L C \Rightarrow A \le_m^L C$
do the same except M_B replaced by M_g
do same change as above



$h(x)=g(f(x))$

$\boxtimes$

Cor $\text{PATH} \le_m^L B \Rightarrow B$ is NL-hard

The following is very surprising

Thm $\overline{\text{PATH}} \in \text{NL}$

$\overline{\text{PATH}} \approx \{\langle G,s,t\rangle : G \text{ does } \underline{\text{not}} \text{ have a path from } s \text{ to } t\}$

key proof idea:

Imagine that we have the value

Count = # of vertices of G reachable from s

NoPath(s, t, n, Count)

Reach ← 0

For all vertices v ≠ t, v ∈ G

Guess whether v is reachable from s

if Guess is yes then

Guess & verify a path of length ≤ n from s to v, one vertex at a time

if path found Reach ← Reach + 1

else reject

end for

if reach = count then accept

else reject

// Reach paths from s to vertices other than t

If ≥ Count such paths, t is not reachable

Then we could decide $\overline{PATH}$ using space O(log n).