אוניברסיטת
אריאל
בשומרון

Computational Complexity

Lecturer: Edward A. Hirsch

https://edwardahirsch.github.io/edwardahirsch

edward.a.hirsch@gmail.com

December 17, 2025

Lecture plan (including next lecture(s))

- Space-bounded computation.
  - Logspace-bounded computation *continued*.
  - Parallel computation.
  - The space hierarchy theorem.
    - Bonus track: The time hierarchy theorem.
    - Bonus bonus track: The nondeterministic time hierarchy theorem.

- Randomized Computation.
  - Definitions ($\mathbf{ZPP}, \mathbf{RP}, \mathbf{BPP}, \mathbf{PP}$). Examples. Equivalence. Probability amplification.
  - Understanding $\mathbf{P/poly}$ through non-uniform advice.
  - $\mathbf{BPP} \subseteq \mathbf{P/poly}$.
  - $\mathbf{BPP} \subseteq \Sigma_2^p$.
  - Discussion on probabilistic classes.

- Interactive protocols.
  - ...

More to learn about Space Complexity

Small-space classes
Logarithmic space, nondeterministic log. space

Parallel computation
How is it related to logarithmic space?

The space hierarchy theorem
More space $\Rightarrow$ more languages

Small space classes: Reminders

Definition

DSpace$[f(n)] = \{L \mid L \text{ is decided by a DTM using space } O(f(n))\}$,
where the DTM has a dedicated read-only input tape.

NSpace$[f(n)] = \{L \mid L \text{ is decided by a NTM using space } O(f(n))\}$,
where the NTM has a dedicated read-only input tape.

$$\mathbf{L} = \mathbf{DL} = \mathbf{DSpace}[\log n]$$

$$\mathbf{NL} = \mathbf{NSpace}[\log n]$$

A logspace many-one reduction is computed by a polynomial-time DTM that uses log space on its working tapes, while its input tape is read-only and its output tape is write-once and unidirectional.

Classes **L**, **NL**, other space-bounded classes below **P** are closed under logspace reductions.

Appetizer:

- **NSpace** classes are closed under complement (non-trivial!).
- Small **DSpace** and **NSpace** classes are interleaved (**NL** $\subseteq$ **DSpace**$[\log^2 n]$).
- Parallel computation classes are also related to them.

The reachability problem: STCON

$$\texttt{STCON} = \{(G, s, t) \mid G = (V, E) \text{ is a directed graph, } s \rightsquigarrow t \text{ in } G\}.$$

NB!!! *We will use it for also implicitly defined graphs — graphs of configurations of TMs.*

Lemma

$\texttt{STCON} \in \mathbf{DSpace}[\log^2 n]$.

$$\begin{aligned} \mathrm{PATH}(x, y, i) &= \text{“there is a path } x \rightsquigarrow y \text{ of length } \leqslant 2^i\text{”} \\ \mathrm{PATH}(x, y, i) &= \bigvee_d (\mathrm{PATH}(x, d, i-1) \wedge \mathrm{PATH}(d, y, i-1)) \end{aligned}$$

Induction base:

$$\mathrm{PATH}(x, y, 0) = \text{“there is an edge } (x, y) \in E, \text{ or } x = y\text{”}$$

Interested in:

$$\mathrm{PATH}(s, t, \lceil \log |V| \rceil)$$

The reachability problem: STCON

$\texttt{STCON} = \{(G, s, t) \mid G = (V, E) \text{ is a directed graph, } s \rightsquigarrow t \text{ in } G\}.$

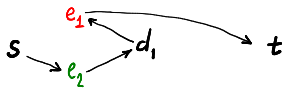
NB!!! *We will use it for also implicitly defined graphs — graphs of configurations of TMs.*

Lemma

$\texttt{STCON} \in \mathbf{DSpace}[\log^2 n]$.

```
Procedure PATH(x, y, i):
-- if x = y then return True;                    // recursion base
-- if i = 0 then return ((x, y) ∈ E);            // recursion base
-- for each d ∈ V do:
-- -- if PATH(x, d, i − 1)                       // 1st recursive call
-- -- then if PATH(d, y, i − 1)                  // 2nd recursive call
-- -- then return True;
-- return False;
```

Interested in: $\text{PATH}(s, t, \lceil \log |V| \rceil)$



Implement it as a recursive procedure (using stack):

$(s, t, 2)$				

The reachability problem: STCON

$\texttt{STCON} = \{(G, s, t) \mid G = (V, E) \text{ is a directed graph, } s \rightsquigarrow t \text{ in } G\}.$

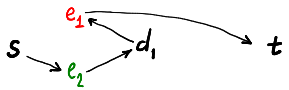
NB!!! *We will use it for also implicitly defined graphs — graphs of configurations of TMs.*

Lemma

$\texttt{STCON} \in \mathbf{DSpace}[\log^2 n]$.

```
Procedure PATH(x, y, i):
-- if x = y then return True;                    // recursion base
-- if i = 0 then return ((x, y) ∈ E);            // recursion base
-- for each d ∈ V do:
-- -- if PATH(x, d, i − 1)                       // 1st recursive call
-- -- then if PATH(d, y, i − 1)                  // 2nd recursive call
-- -- then return True;
-- return False;
```

Interested in: $\text{PATH}(s, t, \lceil \log |V| \rceil)$



Implement it as a recursive procedure (using stack):

$(s, t, 2)$	$(s, d_1, 1)$			

The reachability problem: STCON

$\texttt{STCON} = \{(G, s, t) \mid G = (V, E) \text{ is a directed graph, } s \rightsquigarrow t \text{ in } G\}.$

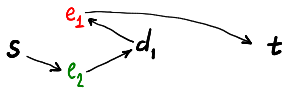
NB!!! *We will use it for also implicitly defined graphs — graphs of configurations of TMs.*

Lemma

$\texttt{STCON} \in \mathbf{DSpace}[\log^2 n]$.

```
Procedure PATH(x, y, i):
-- if x = y then return True;                    // recursion base
-- if i = 0 then return ((x, y) ∈ E);            // recursion base
-- for each d ∈ V do:
-- -- if PATH(x, d, i − 1)                       // 1st recursive call
-- -- then if PATH(d, y, i − 1)                  // 2nd recursive call
-- -- then return True;
-- return False;
```

Interested in: $\text{PATH}(s, t, \lceil \log |V| \rceil)$



Implement it as a recursive procedure (using stack):

$(s, t, 2)$	$(s, d_1, 1)$	$(s, e_1, 0)$		

The reachability problem: STCON

$\texttt{STCON} = \{(G, s, t) \mid G = (V, E) \text{ is a directed graph, } s \rightsquigarrow t \text{ in } G\}.$

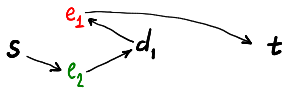
NB!!! *We will use it for also implicitly defined graphs — graphs of configurations of TMs.*

Lemma

$\texttt{STCON} \in \mathbf{DSpace}[\log^2 n]$.

```
Procedure PATH(x, y, i):
-- if x = y then return True;                    //recursion base
-- if i = 0 then return ((x, y) ∈ E);            //recursion base
-- for each d ∈ V do:
-- -- if PATH(x, d, i − 1)                       //1st recursive call
-- -- then if PATH(d, y, i − 1)                  //2nd recursive call
-- -- then return True;
-- return False;
```

Interested in: $\text{PATH}(s, t, \lceil \log |V| \rceil)$



Implement it as a recursive procedure (using stack):

$(s, t, 2)$	$(s, d_1, 1)$	$(s, e_1, 0)\times$		

The reachability problem: STCON

$\texttt{STCON} = \{(G, s, t) \mid G = (V, E) \text{ is a directed graph, } s \rightsquigarrow t \text{ in } G\}.$

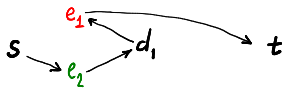
NB!!! *We will use it for also implicitly defined graphs — graphs of configurations of TMs.*

Lemma

$\texttt{STCON} \in \mathbf{DSpace}[\log^2 n]$.

```
Procedure PATH(x, y, i):
-- if x = y then return True;                    //recursion base
-- if i = 0 then return ((x, y) ∈ E);            //recursion base
-- for each d ∈ V do:
-- -- if PATH(x, d, i − 1)                       //1st recursive call
-- -- then if PATH(d, y, i − 1)                  //2nd recursive call
-- -- then return True;
-- return False;
```

Interested in: $\text{PATH}(s, t, \lceil \log |V| \rceil)$



Implement it as a recursive procedure (using stack):

$(s, t, 2)$	$(s, d_1, 1)$	$(s, e_2, 0)$		

The reachability problem: STCON

$\texttt{STCON} = \{(G, s, t) \mid G = (V, E) \text{ is a directed graph, } s \rightsquigarrow t \text{ in } G\}.$

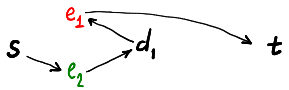
NB!!! *We will use it for also implicitly defined graphs — graphs of configurations of TMs.*

Lemma

$\texttt{STCON} \in$ **DSpace**$[\log^2 n]$.

```
Procedure PATH(x, y, i):
-- if x = y then return True;                    //recursion base
-- if i = 0 then return ((x, y) ∈ E);            //recursion base
-- for each d ∈ V do:
-- -- if PATH(x, d, i − 1)                       //1st recursive call
-- -- then if PATH(d, y, i − 1)                  //2nd recursive call
-- -- then return True;
-- return False;
```

Interested in: $\text{PATH}(s, t, \lceil \log |V| \rceil)$



Implement it as a recursive procedure (using stack):

$(s, t, 2)$	$(s, d_1, 1)$	$(s, e_2, 0)\surd$		

The reachability problem: STCON

$\texttt{STCON} = \{(G, s, t) \mid G = (V, E) \text{ is a directed graph, } s \rightsquigarrow t \text{ in } G\}.$

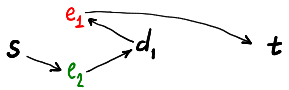
NB!!! *We will use it for also implicitly defined graphs — graphs of configurations of TMs.*

Lemma

$\texttt{STCON} \in \mathbf{DSpace}[\log^2 n]$.

```
Procedure PATH(x, y, i):
-- if x = y then return True;                    //recursion base
-- if i = 0 then return ((x, y) ∈ E);            //recursion base
-- for each d ∈ V do:
-- -- if PATH(x, d, i − 1)                       //1st recursive call
-- -- then if PATH(d, y, i − 1)                  //2nd recursive call
-- -- then return True;
-- return False;
```

Interested in: $\text{PATH}(s, t, \lceil \log |V| \rceil)$



Implement it as a recursive procedure (using stack):

$(s, t, 2)$	$(s, d_1, 1)$	$(e_2, d_1, 0)$		

The reachability problem: STCON

$\texttt{STCON} = \{(G, s, t) \mid G = (V, E) \text{ is a directed graph, } s \rightsquigarrow t \text{ in } G\}.$

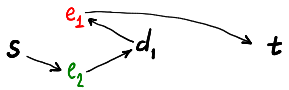
NB!!! *We will use it for also implicitly defined graphs — graphs of configurations of TMs.*

Lemma

$\texttt{STCON} \in \mathbf{DSpace}[\log^2 n]$.

```
Procedure PATH(x, y, i):
-- if x = y then return True;                    //recursion base
-- if i = 0 then return ((x, y) ∈ E);            //recursion base
-- for each d ∈ V do:
-- -- if PATH(x, d, i − 1)                       //1st recursive call
-- -- then if PATH(d, y, i − 1)                  //2nd recursive call
-- -- then return True;
-- return False;
```

Interested in: $\text{PATH}(s, t, \lceil \log |V| \rceil)$



Implement it as a recursive procedure (using stack):

$(s, t, 2)$	$(s, d_1, 1)$	$(e_2, d_1, 0)\surd$		

The reachability problem: STCON

$\texttt{STCON} = \{(G, s, t) \mid G = (V, E) \text{ is a directed graph, } s \rightsquigarrow t \text{ in } G\}.$

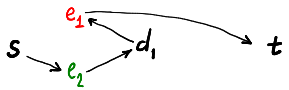
NB!!! *We will use it for also implicitly defined graphs — graphs of configurations of TMs.*

Lemma

$\texttt{STCON} \in \mathbf{DSpace}[\log^2 n]$.

```
Procedure PATH(x, y, i):
-- if x = y then return True;                    // recursion base
-- if i = 0 then return ((x, y) ∈ E);            // recursion base
-- for each d ∈ V do:
-- -- if PATH(x, d, i − 1)                       // 1st recursive call
-- -- then if PATH(d, y, i − 1)                  // 2nd recursive call
-- -- then return True;
-- return False;
```

Interested in: $\text{PATH}(s, t, \lceil \log |V| \rceil)$



Implement it as a recursive procedure (using stack):

$(s, t, 2)$	$(s, d_1, 1)\surd$			

The reachability problem: STCON

$\texttt{STCON} = \{(G, s, t) \mid G = (V, E) \text{ is a directed graph, } s \rightsquigarrow t \text{ in } G\}.$

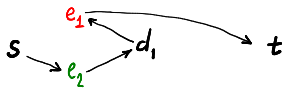
NB!!! *We will use it for also implicitly defined graphs — graphs of configurations of TMs.*

Lemma

$\texttt{STCON} \in$ **DSpace**$[\log^2 n]$.

```
Procedure PATH(x, y, i):
-- if x = y then return True;                    //recursion base
-- if i = 0 then return ((x, y) ∈ E);            //recursion base
-- for each d ∈ V do:
-- -- if PATH(x, d, i − 1)                       //1st recursive call
-- -- then if PATH(d, y, i − 1)                  //2nd recursive call
-- -- then return True;
-- return False;
```

Interested in: $\text{PATH}(s, t, \lceil \log |V| \rceil)$



Implement it as a recursive procedure (using stack):

$(s, t, 2)$	$(d_1, t, 1)$			

Application: Savitch's Theorem

Lemma

STCON is **NL**-complete.

Proof: Consider $L \in \mathbf{NL}$ and the graph of configurations of its machine.
We counted earlier that for each input there are only $\mathrm{poly}(n)$ vertices.

Corollary $\mathbf{NL} \subseteq \mathbf{DSpace}[\log^2 n]$.

Caveat $\mathbf{NL} \subseteq \mathbf{P}$ follows from the Alg, not the Cor.

Application: Savitch's Theorem

Lemma

STCON is **NL**-complete.

Proof: Consider $L \in \mathbf{NL}$ and the graph of configurations of its machine.
We counted earlier that for each input there are only $\mathrm{poly}(n)$ vertices.

Corollary $\mathbf{NL} \subseteq \mathbf{DSpace}[\log^2 n]$.

Caveat $\mathbf{NL} \subseteq \mathbf{P}$ follows from the Alg, not the Cor.

Even more (Savitch's Theorem) $\mathbf{NSpace}[s(n)] \subseteq \mathbf{DSpace}[s(n)^2]$ for "nice" $s(n) \geqslant \log n$.

Corollary (already learned) $\mathbf{PSPACE} = \mathbf{NPSPACE}$.

Application: Savitch's Theorem

Lemma

STCON is **NL**-complete.

Proof: Consider $L \in \mathbf{NL}$ and the graph of configurations of its machine.
We counted earlier that for each input there are only $\mathrm{poly}(n)$ vertices.

Corollary $\mathbf{NL} \subseteq \mathbf{DSpace}[\log^2 n]$.

Caveat $\mathbf{NL} \subseteq \mathbf{P}$ follows from the Alg, not the Cor.

Even more (Savitch's Theorem) $\mathbf{NSpace}[s(n)] \subseteq \mathbf{DSpace}[s(n)^2]$ for "nice" $s(n) \geqslant \log n$.

Corollary (already learned) $\mathbf{PSPACE} = \mathbf{NPSPACE}$.

Fact: the same problem for undirected graphs: USTCON $\in \mathbf{L}$ [Omer Reingold, 2004].

Exercise

2-SAT is **NL**-complete.

Immerman–Szelepcsényi: $\mathbf{NSpace}(f) = \mathbf{co}\text{-}\mathbf{NSpace}(f)$ (not f^2 !)

- We proved that $\mathbf{NSpace}[s(n)] \subseteq \mathbf{DSpace}[s(n)^2]$.
- It means that $\mathbf{co}\text{-}\mathbf{NSpace}[s(n)] \subseteq \mathbf{NSpace}[s(n)^2]$.

- We want to prove that $\mathbf{co}\text{-}\mathbf{NSpace}[s(n)] \subseteq \mathbf{NSpace}[s(n)]$.
- (In particular, $\mathbf{NL} = \mathbf{co}\text{-}\mathbf{NL}$.)

- We proved that $\mathtt{STCON}$, $\overline{\mathtt{STCON}} \in \mathbf{DSpace}[\log^2 n]$.
- It is enough to prove that $\overline{\mathtt{STCON}} \in \mathbf{NSpace}[\log n]$ $(= \mathbf{NL})$ — we will do it now.

Lemma

Given $G = (V, E)$ with all loops $\{u, u\}$, and vertex x, the vertices reachable from $x \in V$ can be visited (in particular, counted) nondeterministically using space $O(\log n)$, where $n = |v|$.

$S(k) := \{v : v \text{ there is a path } x \rightsquigarrow v \text{ of length} \leqslant k\}$ *// Looking for $t \in S(n)$*

Visit $S(k)$ using $|S(\boldsymbol{k-1})|$.

```
-- counter_new:=0;
-- For every u ∈ V do:
-- -- counter_old:=0;
-- -- For every v ∈ V do:                    // Find the previous vertex x ⇝ v → u
-- -- -- Guess and verify a path x = v_0 → v_1 → ... → v_{k-1} = v (allow v_i = v_{i+1});
-- -- -- If the path is correct, counter_old++;
-- -- -- If (v, u) ∈ E then
-- -- -- -- { counter_new++;                                                    }
-- -- If counter_old ≠ |S(k-1)|, reject.     // Failed nondeterministic branch
-- Otherwise we can assume |S(k)| = counter_new.
```

Lemma

Given $G = (V, E)$ with all loops $\{u, u\}$, and vertex x, the vertices reachable from $x \in V$ can be visited (in particular, counted) nondeterministically using space $O(\log n)$, where $n = |v|$.
In particular, we can check whether a given vertex t is reachable.

$S(k) := \{v : v \text{ there is a path } x \rightsquigarrow v \text{ of length} \leqslant k\}$ // *Looking for $t \in S(n)$*
Visit $S(k)$ using $|S(\boldsymbol{k}-1)|$.

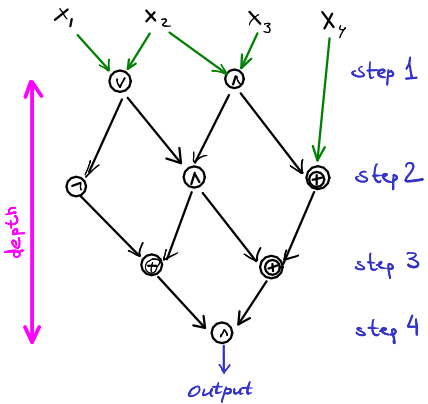
```
For k := 1 to n − 1 do:                                          // |S(0)| = 1
-- counter_new:=0;
-- For every u ∈ V do:
-- -- counter_old:=0;
-- -- For every v ∈ V do:                                       // Find the previous vertex x ⇝ v → u
-- -- -- Guess and verify a path x = v_0 → v_1 → ... → v_{k−1} = v (allow v_i = v_{i+1});
-- -- -- If the path is correct, counter_old++;
-- -- -- If (v, u) ∈ E then
-- -- -- -- { counter_new++; If u = t, reject, else break "For v" and go to the next u;}
-- -- If counter_old ≠ |S(k − 1)|, reject.                      // Failed nondeterministic branch
-- Otherwise we can assume |S(k)| = counter_new.
Accept.                                                          // No path to t is found
```

Introduction to Parallel Computations

How is logspace related to parallel computation?

Consider a circuit. Compute it using parallel processes. Then

$$\text{time} = \text{depth}$$



The ultimate goal: parallelize all poly-time computations to achieve $\log^i n$ time !

Examples of parallel algorithms

Big $\wedge$, big $\vee$, matrix multiplication, matrix power, reachability — to be added on the whiteboard.

(More in an elective course perhaps next year.)

Uniform families and **NC** classes

Consider a family of circuits $\{C_n\}_{n\in\mathbb{N}}$.

Definition

It is called polynomial-time uniform family if there is a polynomial-time DTM T printing each circuit ($T(1^n) = C_n$ in time $\mathrm{poly}(n)$)

Theorem

$L \in \mathbf{P} \iff$ there is polynomial-time uniform circuit family for L.

Uniform families and **NC** classes

Consider a family of circuits $\{C_n\}_{n\in\mathbb{N}}$.

Definition

It is called polynomial-time uniform family if there is a polynomial-time DTM T printing each circuit ($T(1^n) = C_n$ in time $\mathrm{poly}(n)$)

Theorem

$L \in \mathbf{P} \iff$ there is polynomial-time uniform circuit family for L.

Definition

It is called logspace-uniform family if there is a logspace DTM T printing each circuit ($T(1^n) = C_n$ in time $\mathrm{poly}(n)$)

Definition

$\mathbf{NC}^i = \{L :$ there is a logspace-uniform family of $O(\log^i n)$-depth circuits for $L\}$.

$\mathbf{NC} = \bigcup_{i\in\mathbb{N}} \mathbf{NC}^i$ ($\subseteq \mathbf{P}$, because generated in logspace).

So how logspace is related to parallel computations?

Theorem

$\mathbf{NC}^1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{NC}^2$.

Proof (sketch):

$\subseteq$ 1: Evaluate a log-depth circuit in logspace — **Exercise**.
$\subseteq$ 2: Trivial.
$\subseteq$ 3: Powering the adjacency matrix in parallel.

Theorem

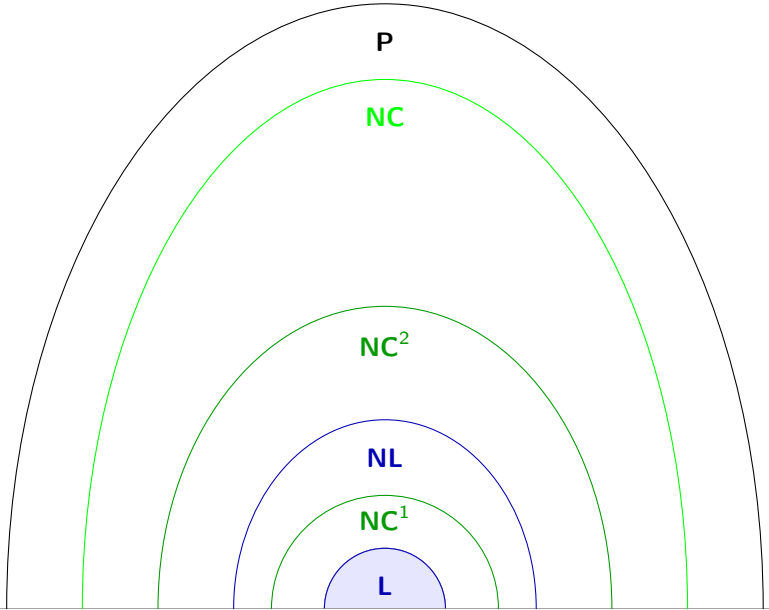
NC is closed under logspace reductions.

In particular, if C is **P**-complete, then $C \in \mathbf{NC} \iff \mathbf{P} = \mathbf{NC}$.

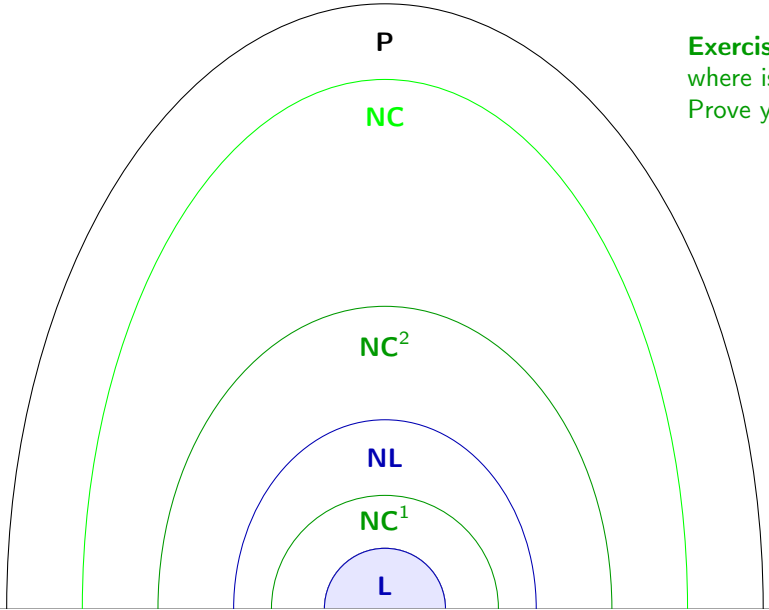
Proof (sketch):

- transform a logspace reduction into a $\log^2$-depth circuit (recall the exercise on implicit reductions),
- then connect the two circuits.

Inside P



Inside P



Exercise:
where is **DSpace**[$\log^2 n$]?
Prove your answer.

At last, we can prove some classes are different!

Space and Time hierarchy theorems

A few words about UTM

Theorem

There is a universal DTM U such that

- $\forall x \; \forall$ DTM $M \quad U(\langle M, x \rangle) = M(x)$,
- U uses $O($ space used by $M(x)$ $)$,
- U uses time $O(T(|x|) \log T(|x|))$, where $T(n) \geqslant n$ is the running time of $M(x)$.

The space hierarchy theorem

More space $\Rightarrow$ more languages

Definition

Space-constructible function $f(n)$: $\geqslant \log n$, non-decreasing, and computable within space $O(f(n))$.

Consider **DSpace**$[f(n)]$ only for space-constructible f !

The space hierarchy theorem

More space $\Rightarrow$ more languages

Definition

Space-constructible function $f(n)$: $\geqslant \log n$, non-decreasing, and computable within space $O(f(n))$.

Consider **DSpace**$[f(n)]$ only for space-constructible f !

Diagonalization: Simulate a machine, then invert its answer — can't be done with fewer resources.

The space hierarchy theorem

More space $\Rightarrow$ more languages

Definition

Space-constructible function $f(n)$: $\geqslant \log n$, non-decreasing, and computable within space $O(f(n))$.

Consider **DSpace**$[f(n)]$ only for space-constructible f !

Diagonalization: Simulate a machine, then invert its answer — can't be done with fewer resources.

Theorem (The Space Hierarchy Theorem, Nothing to do with **PH** !!!)

DSpace$[s(n)] \neq$ **DSpace**$[S(n)]$, where $s(n) = o(S(n))$, where $S(n) \geqslant \log n$.

$$L = \left\{ x = \langle M \rangle 01^k \;\middle|\; M \text{ rejects } x \text{ using space } \leq S(|x|) \right\} \in \textbf{DSpace}[S(|x|)] \qquad \text{(using UTM)}.$$

Assume to the contrary: Let $\boldsymbol{M_*}$ decide L using space $\leq \boldsymbol{s_*}(|\boldsymbol{x}|) = O(s(|x|)) = o(S(|x|))$.

The space hierarchy theorem

More space $\Rightarrow$ more languages

Definition

Space-constructible function $f(n)$: $\geqslant \log n$, non-decreasing, and computable within space $O(f(n))$.

Consider **DSpace**$[f(n)]$ only for space-constructible f !

Diagonalization: Simulate a machine, then invert its answer — can't be done with fewer resources.

Theorem (The Space Hierarchy Theorem, Nothing to do with **PH** !!!)

DSpace$[s(n)] \neq$ **DSpace**$[S(n)]$, where $s(n) = o(S(n))$, where $S(n) \geqslant \log n$.

$$L = \left\{ x = \langle M \rangle 01^k \;\middle|\; \begin{array}{l} k \in \mathbb{N} \cup \{0\}, \\ |\langle M \rangle| < S(|x|), \\ M \text{ rejects } x \text{ using space } \leq S(|x|) \end{array} \right\} \in \textbf{DSpace}[S(|x|)] \qquad \text{(using UTM)}.$$

Assume to the contrary: Let $\boldsymbol{M_*}$ decide L using space $\leq \boldsymbol{s_*}(|\boldsymbol{x}|) = O(s(|x|)) = o(S(|x|))$.

"small $o(\ldots)$" $\Rightarrow \exists N_1\ \forall n > N_1 \quad s_*(n) < S(n)$.

Consider $N_* > \max\{N_1, 2^{|\langle M_* \rangle|}\}$, so that M_* in the condition of L has enough space to run.

If $M_*(\langle M_* \rangle 01^{N_* - |\langle M_* \rangle| - 1}) = 1 \Rightarrow$ then its input $\notin L$, and vice versa.

Bonus track: The time hierarchy theorem

Definition

Time-constructible function $f(n)$: $\geqslant n$, non-decreasing, and computable within time $O(f(n))$.

Consider **DTime**$[f(n)]$ only for time-constructible f !

Theorem

DTime$[t(n)] \neq$ **DTime**$[T(n)]$, where $\frac{t(n)\log t(n)}{T(n)} \to 0$.

$$L = \left\{ x = \langle M \rangle 01^k \;\middle|\; \begin{array}{l} k \in \mathbb{N} \cup \{0\}, \\ |M| < T(|x|), \\ \text{UTM rejects } \langle M, x \rangle \text{ using time} \leq T(|x|) \end{array} \right\} \in \textbf{DTime}[T(|x|)].$$

Let M_* decide L in time $\leq t_*(|x|) = O(t(|x|))$.

c = the const in UTM's overhead

$\exists N_1\ \forall n > N_1 \quad c t_*(n) \log t_*(n) < T(n)$

Consider $N_* > \max\{N_1, 2^{|\langle M_* \rangle|}\}$, so that M_* in the condition of L has enough time to be run by by UTM.

If $M_*(M_* 01^{N_* - |M_*| - 1}) = 1 \Rightarrow$ then its input $\notin L$, and vice versa.

Superbonus track: The nondeterministic time hierarchy theorem

(Will be proved if there is enough time.)

Theorem

NTime$[t(n)] \neq$ **NTime**$[T(n)]$, where $t(n+1) = o(T(n))$, both time-constructible.

Corollaries

- **L** $\neq$ **PSPACE**.
- **P** $\neq$ **EXP**.
- **NP** $\neq$ **NEXP**.
- (etc – consider any [“nice”] functions you want)

Padding

Now as we started to talk about such classes as **EXP** and **NEXP**.

$$\mathbf{P} \stackrel{?}{=} \mathbf{NP} \quad \text{vs} \quad \mathbf{EXP} \stackrel{?}{=} \mathbf{NEXP}$$

How are they related?

Padding

Now as we started to talk about such classes as **EXP** and **NEXP**.

$$\mathbf{P} = \mathbf{NP} \quad \Rightarrow \quad \mathbf{EXP} = \mathbf{NEXP}$$

How are they related?

Assume $\mathbf{P} = \mathbf{NP}$. Take $L \in \mathbf{NEXP}$, decided by $2^{p(n)}$-time NTM N.

Consider $L_{pad} = \{\underbrace{x01^{2^{p(|x|)}-|x|-1}}_{2^{p(n)} \text{ bits}} : x \in L\}$ $\ldots$ $\subseteq \mathbf{NP}$. // *Ignore the ~~garbage~~ padding.*

Under our assumption $L_{pad} \in \mathbf{P}$, decided by a $q(n)$-time DTM M;

Padding

Now as we started to talk about such classes as **EXP** and **NEXP**.

$$\mathbf{P} = \mathbf{NP} \quad \Rightarrow \quad \mathbf{EXP} = \mathbf{NEXP}$$

How are they related?

Assume $\mathbf{P} = \mathbf{NP}$. Take $L \in \mathbf{NEXP}$, decided by $2^{p(n)}$-time NTM N.

Consider $L_{pad} = \{\underbrace{x01^{2^{p(|x|)}-|x|-1}}_{2^{p(n)} \text{ bits}} : x \in L\}$ $\quad \ldots \quad$ $\subseteq \mathbf{NP}$. // *Ignore the ~~garbage~~ padding.*

Under our assumption $L_{pad} \in \mathbf{P}$, decided by a $q(n)$-time DTM M;
Then we can show that $L \in \mathbf{EXP}$:

Algorithm for L:
`On input` x`,`
`Run` $M(x01^{2^{p(|x|)}-|x|-1})$ `and return its answer.`

The running time is $q(2^{p(|x|)})$, and this is exponential in $|x|$. // $\left(2^{n^\ell}\right)^k = 2^{kn^\ell}$

Padding

Now as we started to talk about such classes as **EXP** and **NEXP**.

$$\mathbf{P} \neq \mathbf{NP} \quad \Leftarrow \quad \mathbf{EXP} \neq \mathbf{NEXP}$$

How are they related?

Assume $\mathbf{P} = \mathbf{NP}$. Take $L \in \mathbf{NEXP}$, decided by $2^{p(n)}$-time NTM N.

Consider $L_{pad} = \{\underbrace{x01^{2^{p(|x|)}-|x|-1}}_{2^{p(n)} \text{ bits}} : x \in L\}$ $\quad \ldots \quad \subseteq \mathbf{NP}$. // *Ignore the ~~garbage~~ padding.*

Under our assumption $L_{pad} \in \mathbf{P}$, decided by a $q(n)$-time DTM M;
Then we can show that $L \in \mathbf{EXP}$:

Algorithm for L:
`On input` x`,`
`Run` $M(x01^{2^{p(|x|)}-|x|-1})$ `and return its answer.`

The running time is $q(2^{p(|x|)})$, and this is exponential in $|x|$. // $\left(2^{n^\ell}\right)^k = 2^{kn^\ell}$

Next lecture: **Randomized (probabilistic) algorithms**