Computational Complexity Theory

Lecture 14: Karp-Lipton theorem; Class AC and NC

Department of Computer Science,
Indian Institute of Science

Recap: Algorithm per input length?

- *"One might imagine that P ≠ NP, but SAT is tractable in the following sense: for every ℓ there is a very short program that runs in time ℓ^2 and correctly treats all instances of size ℓ."* --- Karp and Lipton (1982).

- P ≠ NP rules out the existence of a <u>single</u> efficient algorithm for SAT that handles <u>all</u> input lengths. But, it doesn't rule out the possibility of having <u>a sequence of</u> efficient SAT algorithms – one <u>for each input length</u>.

Recap: Lesson from Cook-Levin

- Locality of computation implies that an algorithm A working on inputs of some fixed length n and running in time $T(n)$ can be viewed as a Boolean circuit ϕ of size $O(T(n)^2)$ s.t. $A(x) = \phi(x)$ for every $x \in \{0,1\}^n$.
- On the other hand, a circuit on inputs of length n and of size S can be viewed as an algorithm working on length n inputs and running in time S.

- To rule the existence of a sequence of algorithms – one for each input length – we need to rule out the existence of a sequence of <u>(i.e., a family of) circuits</u>.

Recap: Boolean circuits

- A <u>Boolean circuit</u> is a directed acyclic graph whose nodes/gates are labelled as follows:
  - A node with in-degree zero is labelled by an input variable, and it outputs the value of the variable.
  - Any other node is labelled by one of the three operations $\wedge$, $\vee$, $\neg$, and it outputs the value of the operation on its input.

  Nodes with out-degree zero are the output gates.

- <u>**Size**</u> of circuit is the no. of edges in it. <u>**Depth**</u> is the length of the longest path from an i/p to o/p node.

Recap: Boolean circuits

- A <u>Boolean circuit</u> is a directed acyclic graph whose nodes/gates are labelled as follows:
  - A node with in-degree zero is labelled by an input variable, and it outputs the value of the variable.
  - Any other node is labelled by one of the three operations $\wedge$, $\vee$, $\neg$, and it outputs the value of the operation on its input.

  Nodes with out-degree zero are the output gates.

- <u>**Size**</u> corresponds to "sequential time complexity". <u>**Depth**</u> corresponds to "parallel time complexity".

Recap: Class P/poly

- Let $T\colon \mathbb{N} \rightarrow \mathbb{N}$ be some function.
- Definition: A $T(n)$-size circuit family is a set of circuits $\{C_n\}_{n\in\mathbb{N}}$ such that C_n has n inputs and $|C_n| \leq T(n)$.
- Definition: A language L is in $\mathrm{SIZE}(T(n))$ if there's a $T(n)$-size circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that

$$x \in L \quad \Leftrightarrow C_n(x) = 1, \text{ where } n = |x|.$$

- Defintion: Class $\mathrm{P/poly} = \bigcup_{c\geq 1} \mathrm{SIZE}(n^c)$.

Recap: Class P/poly

- Observation: $P \subseteq P/poly$.
- Is $P = P/poly$? No! P/poly contains undecidable languages.

- Let $UHALT = \{1^{\#(M,y)} : (M,y) \in HALT\}$. Then, UHALT is also an undecidable language.

- Obs. Any unary language is in P/poly.
- Hence, $P \subsetneq P/poly$.

Recap: Class P/poly

- What makes P/poly contain undecidable languages? *Ans:* $L \in$ P/poly implies that L is decided by a circuit family $\{C_n\}$, where $|C_n| = n^{O(1)}$. <u>We don't require that C_n is poly-time computable from 1^n</u>.
- P/poly is a <u>*non-uniform class*</u> as a language in this class is allowed to have different algorithms/circuits for different input lengths.
- P is a <u>*uniform class*</u> as a language in this class has one algorithm for all inputs.
- Is SAT $\in$ P/poly? In other words, is NP $\subsetneq$ P/poly?

Karp-Lipton theorem

- Theorem (*Karp & Lipton 1982*). If $\mathrm{NP} \subsetneq \mathrm{P/poly}$ then $\mathrm{PH} = \Sigma_2$.
- Proof. We'll show that $\mathrm{NP} \subsetneq \mathrm{P/poly}$ implies $\Pi_2 = \Sigma_2$. It's sufficient to show that $\Pi_2 \subseteq \Sigma_2$.

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let $L \in \Pi_2$. There's a polynomial function $q(.)$ and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \exists u_2 \in \{0,1\}^{q(|x|)}\ M(x,u_1,u_2) = 1$.

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let L $\in \Pi_2$. There's a polynomial function q(.) and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \exists u_2 \in \{0,1\}^{q(|x|)}\ M(x,u_1, u_2) = 1.$

- Goal. Come up with a polynomial function p(.) and a poly-time TM N s.t.

  $x \in L \Leftrightarrow \exists v_1 \in \{0,1\}^{p(|x|)}\ \forall v_2 \in \{0,1\}^{p(|x|)}\ N(x,v_1, v_2) = 1.$

- Think about designing such a TM N.

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let L $\in \Pi_2$. There's a polynomial function q(.) and a poly-time TM M s.t.

by Cook-Levin

$$x \in L \iff \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \phi(x,u_1, u_2) = 1.$$

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If $NP \subsetneq P/poly$ then $PH = \Sigma_2$.
- Proof. Let $L \in \Pi_2$. There's a polynomial function $q(.)$ and a poly-time TM M s.t.

by Cook-Levin

$$x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)} \; \exists u_2 \in \{0,1\}^{q(|x|)} \; \phi(x, u_1, u_2) = 1.$$

- If M runs in time $T(n) = n^{O(1)}$ on (x, u_1, u_2), where $|x| = n$, then $|\phi| = O(T(n)^2)$. Let m = #(bits to write ϕ).
- N can compute ϕ from M in $poly(|x|)$ time.

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If $NP \subsetneq P/poly$ then $PH = \Sigma_2$.
- Proof. Let $L \in \Pi_2$. There's a polynomial function $q(.)$ and a poly-time TM M s.t.

by Cook-Levin

$$x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \exists u_2 \in \{0,1\}^{q(|x|)}\ \phi(x,u_1,u_2) = 1.$$

- If M runs in time $T(n) = n^{O(1)}$ on (x,u_1,u_2), where $|x| = n$, then $|\phi| = O(T(n)^2)$. Let m = length of ϕ.
- N can compute ϕ from M in poly(|x|) time.

Karp-Lipton theorem

- Theorem (*Karp & Lipton 1982*). If $NP \subsetneq P/poly$ then $PH = \Sigma_2$.
- Proof. Let $L \in \Pi_2$. There's a polynomial function $q(.)$ and a poly-time TM M s.t.

$$x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \phi(x,u_1, u_2) = 1 .$$

$\phi(x,u_1, u_2)$ as a function of u_2 is satisfiable. Wlog ϕ is a CNF (why?).

Karp-Lipton theorem

- Theorem (*Karp & Lipton 1982*). If $NP \subsetneq P/poly$ then $PH = \Sigma_2$.
- Proof. Let $L \in \Pi_2$. There's a polynomial function $q(.)$ and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \phi(x,u_1, u_2) \in SAT.$
- By assumption, $SAT \in P/poly$, i.e., there's a circuit C_m of size $p(m) = m^{O(1)}$ that correctly decides satifiability of all input circuits ϕ of length m.

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let L $\in \Pi_2$. There's a polynomial function q(.) and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \phi(x,u_1, u_2) \in SAT.$
- First attempt. A Σ_2 statement to capture membership of strings in L.

  $x \in L \Leftrightarrow \exists C_m \in \{0,1\}^{p(m)}\ \forall u_1 \in \{0,1\}^{q(|x|)}\ C_m(\phi(x,u_1, u_2))=1.$

Karp-Lipton theorem

- Theorem (*Karp & Lipton 1982*). If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let L $\in \Pi_2$. There's a polynomial function q(.) and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)} \ \phi(x,u_1, u_2) \in SAT.$
- First attempt. A Σ_2 statement to capture membership of strings in L.

  $x \in L \Leftrightarrow \exists C_m \in \{0,1\}^{p(m)} \ \forall u_1 \in \{0,1\}^{q(|x|)} \ C_m(\phi(x,u_1, u_2))=1.$

- Wrong! Think about a C_m that always outputs 1.

Karp-Lipton theorem

- Theorem (*Karp & Lipton 1982*). If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let L $\in \Pi_2$. There's a polynomial function q(.) and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \phi(x,u_1, u_2) \in SAT.$
- First attempt. A Σ_2 statement to capture membership of strings in L.

  $x \in L \Leftrightarrow \exists C_m \in \{0,1\}^{p(m)}\ \forall u_1 \in \{0,1\}^{q(|x|)}\ C_m(\phi(x,u_1, u_2))=1.$

- Need to be sure that C_m is the right circuit.

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = $\sum_2$.
- Proof. Let $L \in \prod_2$. There's a polynomial function $q(.)$ and a poly-time TM M s.t.

  $x \in L \iff \forall u_1 \in \{0,1\}^{q(|x|)} \ \phi(x,u_1, u_2) \in SAT.$
- If there's a circuit C_m of size $m^{O(1)}$ that correctly decides satifiability of all input circuits ϕ of length m, then <u>by self-reducibility of SAT</u>, there's a <u>multi-output</u> circuit D_m of size $r(m) = m^{O(1)}$ that <u>outputs a *satisfying assignment*</u> for input ϕ if $\phi \in SAT$. *(Homework)*

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let $L \in \Pi_2$. There's a polynomial function $q(.)$ and a poly-time TM M s.t.

$$x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)} \ \phi(x,u_1, u_2) \in SAT.$$

- A Σ_2 statement to capture membership in L.

$$x \in L \Leftrightarrow$$

$$\exists D_m \in \{0,1\}^{r(m)} \ \forall u_1 \in \{0,1\}^{q(|x|)} \ \phi(x,u_1, \underbrace{D_m(\phi(x,u_1, u_2))}_{\text{assignment to the } u_2 \text{ variables}}) = 1.$$

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let L $\in \Pi_2$. There's a polynomial function q(.) and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)} \;\; \phi(x,u_1, u_2) \in SAT.$

- A Σ_2 statement to capture membership in L.

  $x \in L \Leftrightarrow$

  $\exists D_m \in \{0,1\}^{r(m)} \; \forall u_1 \in \{0,1\}^{q(|x|)} \;\; \underbrace{\phi(x,u_1, D_m(\phi(x,u_1, u_2)) = 1}_{\text{Can be checked by a poly-time TM N.}}.$

Karp-Lipton theorem

- Theorem (*Karp & Lipton 1982*). If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let $L \in \Pi_2$. There's a polynomial function $q(.)$ and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \phi(x,u_1, u_2) \in SAT.$

- A Σ_2 statement to capture membership in L.

  $x \in L \Leftrightarrow$

  $\exists D_m \in \{0,1\}^{r(m)}\ \forall u_1 \in \{0,1\}^{q(|x|)}\ N(x, D_m, u_1) = 1.$

Karp-Lipton theorem

- Theorem (*Karp & Lipton 1982*). If $\mathrm{NP} \subsetneq \mathrm{P/poly}$ then $\mathrm{PH} = \Sigma_2$.

- If we can show $\mathrm{NP} \not\subset \mathrm{P/poly}$ assuming $\mathrm{P} \neq \mathrm{NP}$, then

$$\mathrm{NP} \not\subset \mathrm{P/poly} \Leftrightarrow \mathrm{P} \neq \mathrm{NP}.$$

- Karp-Lipton theorem shows $\mathrm{NP} \not\subset \mathrm{P/poly}$ assuming the stronger statement $\mathrm{PH} \neq \Sigma_2$.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly?

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1-\exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Follows from a counting argument.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.
- Number of bits required to write the adjacency lists it at most $s(\log s + 3) + 4(s + n) \leq 9s.\log s$.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.
- Number of circuits of size s is at most $3^s.2^{9s.\log s}$.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.
- Number of circuits of size s is at most $2^{11 s.\log s}$.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.
- Number of circuits of size s is at most $\exp(2^{n-1})$.
- Number of functions in n variables is $\exp(2^n)$.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. 1- $\exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.
- So, circuits of size s can compute at most $\exp(-2^{n-1})$ fraction of all Boolean functions on n variables.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many.
- Is one out of so many functions outside P/poly in NP?

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many.
- Is one out of so many functions outside P/poly in NP? We don't know even after ~40 yrs of research!

- Theorem. *(Iwama, Lachish, Morizumi & Raz 2002)* There is a language $L \in NP$ such that any circuit C_n that decides $L \cap \{0,1\}^n$ requires $5n - o(n)$ many $\wedge$ and $\vee$ gates.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many.
- Is one out of so many functions outside P/poly in NP? We don't know even after ~40 yrs of research!

- Theorem. *(Iwama, Lachish, Morizumi & Raz 2002)* There is a language $L \in NP$ such that any circuit C_n that decides $L \cap \{0,1\}^n$ requires $5n - o(n)$ many $\wedge$ and $\vee$ gates.

Results of this kind are known as circuit lower bound.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many.
- Is one out of so many functions outside P/poly in NP? We don't know even after ~40 yrs of research!

- Open problem. Prove that NEXP $\not\subset$ P/poly .

Lower bounds for restricted circuits

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

Lower bounds for restricted circuits

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

- Fact. PARITY(x_1, x_2, …, x_n) can be computed by a circuit of size $O(n)$ and a formula of size $O(n^2)$.

Homework

Lower bound for Boolean formulas

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

- Theorem. *(Khrapchenko 1971)* Any formula computing PARITY($x_1, x_2, \ldots, x_n$) has size $\Omega(n^2)$.

Lower bound for Boolean formulas

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

- Theorem. (*Andreev 1987, Hastad 1998*) There's a f that can be computed by a $O(n)$-size circuit such that any formula computing f has size $\Omega(n^{3-o(1)})$.

Technique: Shrinkage of formulas under random restrictions (*Subbotovskaya 1961*).

Lower bound for Boolean formulas

- Nevertheless, the <u>clean combinatorial structure</u> of a circuit has been used to prove lower bounds for some <u>natural classes of circuits</u>.
- The proofs of these lower bounds introduced and developed some highly <u>interesting techniques</u>.

- Conjecture. <u>*(Circuits more powerful than formulas)*</u> There's a f that can be computed by a $O(n)$-size circuit such that any formula computing f has size $n^{\omega(1)}$.

An interesting approach was given by *Karchmer, Raz & Wigderson (1995)* .

LB for AC^0 and monotone circuits

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.
- We'll discuss a **super-polynomial** lower bound for constant depth circuits later.

Non-uniform size hierarchy

- Shanon's result. There's a constant $c \geq 1$ such that every Boolean function in n variables has a circuit of size at most $c.(2^n/n)$.

- Theorem. There's a constant $d \geq 1$ s.t. if $T_1: \mathbb{N} \rightarrow \mathbb{N}$ & $T_2: \mathbb{N} \rightarrow \mathbb{N}$ and $T_1(n) \leq d^{-1}.T_2(n) \leq T_2(n) \leq c.(2^n/n)$ then

$$\mathrm{SIZE}(T_1(n)) \subsetneq \mathrm{SIZE}(T_2(n)).$$

Non-uniform size hierarchy

- Shanon's result. There's a constant $c \geq 1$ such that every Boolean function in n variables has a circuit of size at most $c.(2^n/n)$.

- Theorem. There's a constant $d \geq 1$ s.t. if $T_1 : \mathbb{N} \rightarrow \mathbb{N}$ & $T_2 : \mathbb{N} \rightarrow \mathbb{N}$ and $T_1(n) \leq d^{-1}.T_2(n) \leq T_2(n) \leq c.(2^n/n)$ then

$$\mathrm{SIZE}(T_1(n)) \subsetneq \mathrm{SIZE}(T_2(n)).$$

- Proof. Uses Shanon's result and a counting argument.

(*Homework*)

Class NC^i and AC^i

Class NC

- NC stands for Nick's Class – named after Nick Pippenger.
- Definition. For $i \in \mathbb{N}$, a language L is in NC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $NC = \bigcup_{i \in \mathbb{N}} NC^i$.

Class NC

- NC stands for <u>Nick's Class</u> – named after Nick Pippenger.
- Definition. For $i \in \mathbb{N}$, a language L is in NC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $NC = \bigcup_{i \in \mathbb{N}} NC^i$.

- *Homework:* PARITY is in NC^1.

Class NC

- NC stands for Nick's Class – named after Nick Pippenger.
- Definition. For $i \in \mathbb{N}$, a language L is in NC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.
- Definition. $NC = \bigcup_{i \in \mathbb{N}} NC^i$.
- $\boxed{NC^1 = \textit{poly(n)}\text{-size Boolean formulas.}}$ (*Assignment*)

Class NC

- NC stands for <u>Nick's Class</u> – named after Nick Pippenger.
- Definition. For $i \in \mathbb{N}$, a language L is in NC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Further, L is in ***<u>log-space uniform</u>*** NC^i if C_n is implicitly log-space computable from 1^n.

Note: Sometimes NC^i is defined as log-space uniform NC^i.

Class NC

- NC stands for <u>Nick's Class</u> – named after Nick Pippenger.
- Definition. For $i \in \mathbb{N}$, a language L is in NC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Further, L is in <u>***log-space uniform***</u> NC^i if C_n is implicitly log-space computable from 1^n.

log-space uniform $NC \subseteq P$.

NC ≡ Efficient parallel computation

- Definition. A language L can be decided *efficiently in parallel* if there's a polynomial function $q(.)$ and constants c & i s.t. $L \cap \{0,1\}^n$ can be decided using $q(n)$ many processors in $c.(\log n)^i$ time.

NC ≡ Efficient parallel computation

- Definition. A language L can be decided *efficiently in parallel* if there's a polynomial function $q(.)$ and constants c & i s.t. $L \cap \{0,1\}^n$ can be decided using $q(n)$ many processors in $c.(\log n)^i$ time.
- Model: **PRAM** (has a central shared memory)
  - A processor can "deliver" a message to any other processor in $O(\log n)$ time.
  - A processor has $O(\log n)$ bits of memory and performs a poly-time computation at every step.
  - Processor computation steps are synchronized.

NC ≡ Efficient parallel computation

- Definition. A language L can be decided *efficiently in parallel* if there's a polynomial function $q(.)$ and constants c & i s.t. $L \cap \{0,1\}^n$ can be decided using $q(n)$ many processors in $c.(\log n)^i$ time.

- Observation. A language L is in NC if and only if L can be decided efficiently in parallel.
- Proof. Almost immediate from the assumptions on the parallel computation model.

Class AC

- Definition. For $i \in \mathbb{N} \cup \{0\}$, a language L is in AC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size **unbounded fan-in** circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $AC = \bigcup_{i \geq 0} AC^i$. (stands for *Alternating Class*)

Class AC

- Definition. For $i \in \mathbb{N} \cup \{0\}$, a language L is in AC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size **unbounded fan-in** circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $AC = \bigcup_{i \geq 0} AC^i$.

- Observation. $AC^i \subseteq NC^{i+1} \subseteq AC^{i+1}$ for all $i \geq 0$.

Replace an unbounded fan-in gate by a binary tree of bounded fan-in gates.

Class AC

- Definition. For $i \in \mathbb{N} \cup \{0\}$, a language L is in AC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size **unbounded fan-in** circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $AC = \bigcup_{i \geq 0} AC^i$.

- Observation. $NC = AC$.

Class AC

- Definition. For $i \in \mathbb{N} \cup \{0\}$, a language L is in AC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size **unbounded fan-in** circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $AC = \bigcup_{i \geq 0} AC^i$.

- In the next lecture, we'll show that PARITY is not in AC^0, i.e., $AC^0 \subsetneq NC^1$.

Class AC

- Definition. For $i \in \mathbb{N} \cup \{0\}$, a language L is in AC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size **unbounded fan-in** circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $AC = \bigcup_{i \geq 0} AC^i$.

- Further, L is in ***log-space uniform*** AC^i if C_n is implicitly log-space computable from 1^n.

log-space uniform $AC \subseteq P$.