Computational Complexity; slides 11, HT 2022
Circuit complexity, NC, AC, P-completeness

Prof. Paul W. Goldberg (Dept. of Computer Science,
University of Oxford)

HT 2022

Overview

We "dissect" the class P in more detail, eventually identifying a non-trivial proper subset of it.

Boolean Circuits: another model of computation

A standard mathematical model of "digital circuit"

A Boolean circuit is a DAG:

- *Inputs* : nodes without incoming edges labeled with 0 or 1.
- *Gates* : nodes with (one or two) incoming edges and one outgoing edge labeled AND, OR, or NOT.
- A single node is labeled as *output*.

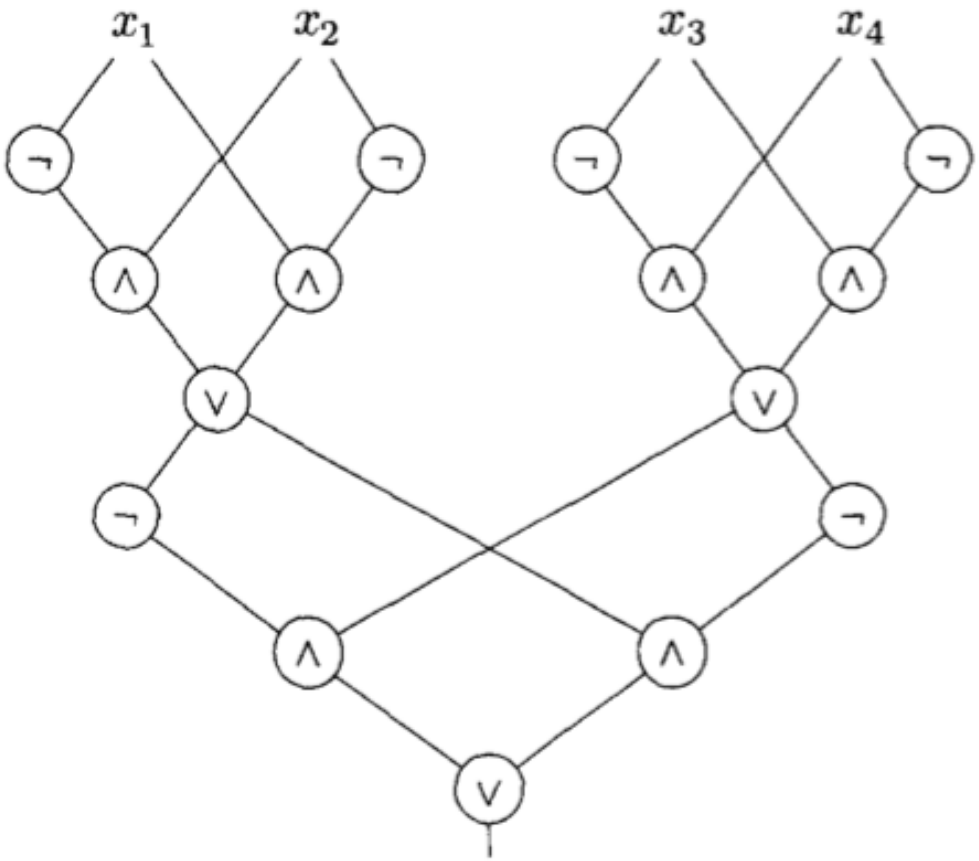
side-note: here we focus on AND, OR, or NOT (the "standard basis"); circuit classes with other operations are of interest, e.g. XOR with multiple inputs (counting mod 2 the number of 1's in inputs), and versions for counting mod k, for other values of k

Boolean Circuits

Input-output behaviour described using *Boolean functions*

To each circuit C with n inputs is associated $f_C : \{0,1\}^n \to \{0,1\}$

Example: parity function with 4 variables (returns 1 if and only if the number of 1's in the input is odd)



Minimal Circuits

Some basic definitions:

Circuit Size: number of gates contained in the circuit

Circuit depth: Length of the longest path from an input to the output gate

Size-minimal circuits: no circuit with fewer gates computes the same function.

Depth-minimal circuits: no circuit with smaller depth computes the same function.

Minimisation (given a circuit, find a smallest equivalent one) is a hard problem in practice

Not known to be in P or even in NP.

Problem of current research interest: *Minimum Circuit Size Problem* (MCSP):

Input: boolean function f presented as truth table; number s
Question: is there a circuit of size s computing f?

Families of Circuits

test membership in language $\mathcal{L}$ using circuits...

$\mathcal{L}$ may have strings of different lengths but circuits have fixed inputs

Circuit family

An infinite list of circuits $C = (C_0, C_1, C_2, \ldots)$ where C_n has n inputs. Family C decides a binary language $\mathcal{L}$ if

$w \in \mathcal{L}$ if and only if $C_k(w) = 1$ (for every string w of length k)

Size (Depth) complexity of a circuit family $C = (C_0, C_1, \ldots)$

Function $f : \mathcal{N} \to \mathcal{N}$ with $f(n)$ size (depth) of C_n

Circuit-size (Circuit-depth) complexity of a language

Size (Depth) complexity of a circuit family for that language where every component circuit C_i is size-minimal (depth-minimal).

Circuit Complexity vs Time Complexity

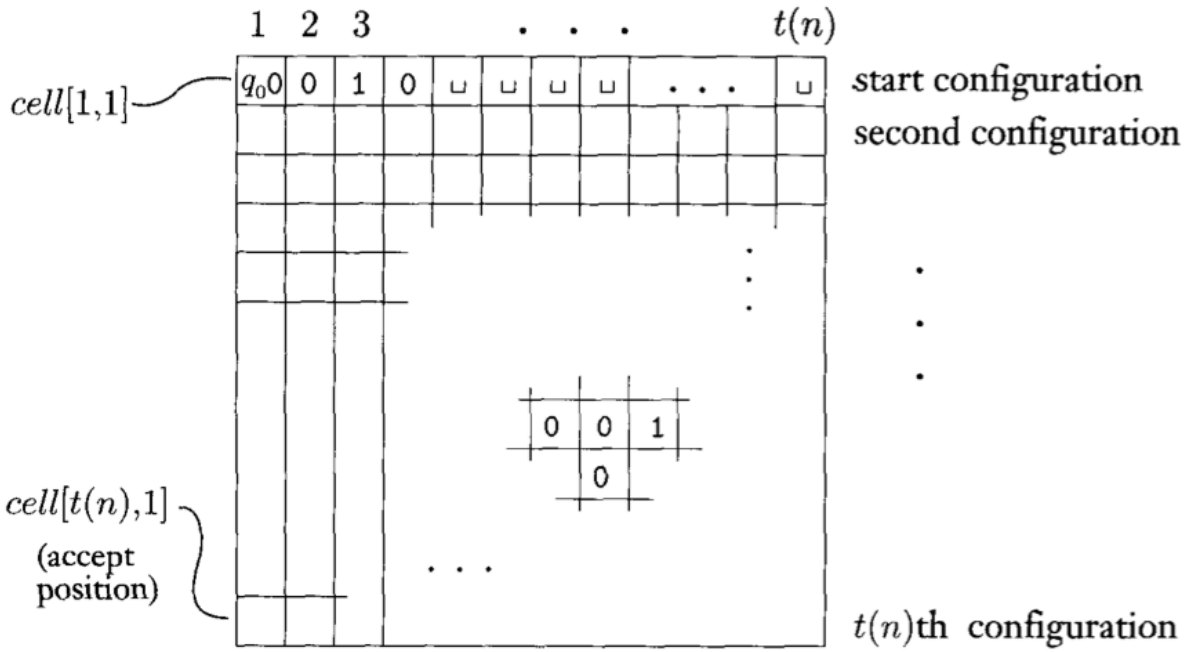
Small time complexity $\Rightarrow$ small circuit complexity

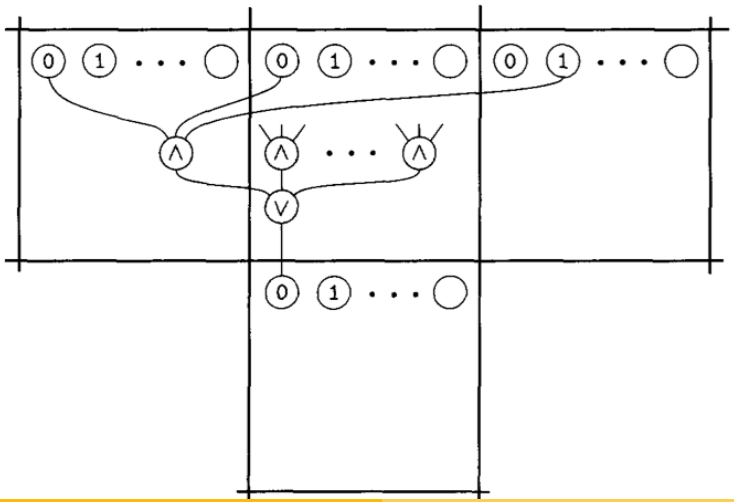
Theorem. If $\mathcal{L} \in \mathrm{DTIME}(t(n))$ with $t(n) \geq n$ then $\mathcal{L}$ has circuit-size complexity $O(t^2(n))$

Proof idea

1. Take a TM M that decides $\mathcal{L}$ in $t(n)$
2. For each n construct C_n that simulates M on inputs of length n
3. Gates of C_n are organised in $t(n)$ rows (one per configuration)
4. Wire each to the previous one to calculate the new configuration from the previous row's configuration as in the transition function.

Circuit Complexity vs. Time Complexity





Consequences

This theorem and its proof yield surprisingly deep consequences.

1. It sheds some light on the P versus NP issue:

   If we can find a language in NP that has super-polynomial circuit complexity then P $\neq$ NP.
2. It allows us to identify a natural P-complete problem.
3. It provides an alternative proof for Cook-Levin theorem.

Yet another complexity class: P/poly — problems that can be solved with polynomial-size circuit families

From the theorem, P$\subseteq$P/poly

P-completeness

Definition. A language $\mathcal{L}$ is P-complete (or PTIME-complete) if

- it is in P and
- every other language in P is LOGSPACE reducible to $\mathcal{L}$.

Circuit Value Problem (CVP) is the problem of checking, given a circuit C and concrete input values, whether C outputs 1.

(Called *MonotoneCVP* if C does not include negation.)

Theorem. CVP is P-complete.

Proof Idea

1. Take the previous construction and some $\mathcal{L} \in$ P.
2. Given x, construct a circuit that simulates a TM M for $\mathcal{L}$ on inputs of length x.
3. The reduction has repetitive structure and is feasible in logarithmic space.

NP-completeness via Circuits; Cook's thm revisited

CIRCUIT-SAT is the problem of checking, given a circuit C, whether C outputs 1 for *some* setting of the inputs.

Theorem. CIRCUIT-SAT is NP-complete.

Proof idea Membership in NP is obvious so take any $\mathcal{L} \in$ NP.

1. There is a verifier $V_{\mathcal{L}}(x, s)$ checking whether s is a solution for x.
   $\Rightarrow$ $V_{\mathcal{L}}$ works in poly time in $|x|$ and $|s|$ is polynomial in $|x|$.
2. $V_{\mathcal{L}}$ can be rendered as a circuit family C whose inputs encode x, s.
   $\Rightarrow$ $C_{|x|+|s|}$ returns 1 iff s is a solution for x.
3. To check $x \in \mathcal{L}$, build $C_{|x|+|s|}$ *leaving the bits for s unknown*
   $\Rightarrow$ the satisfying values for unknowns yield the solutions for x.

Circuit-SAT and SAT are inter-reducible (poly-time equivalent)
$\Rightarrow$ Cook-Levin theorem follows!

The Power of Circuits

A key caveat of circuits. They are not a realistic model of computation!

Theorem. There exist undecidable languages having polynomial size circuits.

1. Consider any undecidable $\mathcal{L} \subseteq \{0,1\}^*$.
2. Let $U = \{1^n \ : \ \text{the binary expansion of } n \text{ is in } \mathcal{L}\ \}$
3. U is undecidable: $\mathcal{L}$ reduces to it via an (exponential) reduction.
4. U has a trivial family of polynomial circuits!
   - If $1^n \in U$ then C_n consists of $n-1$ AND gates.
   - If $1^n \notin U$ then C_n outputs 0.

Uniformity

The catch: Constructing the circuits involves solving an unsolvable problem

Uniform circuit families

Given 1^n as input, C_n can be constructed in LOGSPACE.

$\Rightarrow$ Circuits should be easy to construct!

With uniformity, circuits become a sensible model of computation.

Theorem. A language $\mathcal{L}$ is in P iff it has uniformly polynomial circuits.

Proof

1. Assume $\mathcal{L}$ has uniformly polynomial circuits and let $w \in \mathcal{L}$.
2. Construct $C_{|w|}$ in log. space (and hence in poly. time).
3. Evaluate the circuit (CVP is in P).

Circuits and Parallel Computation

Boolean circuits are genuinely *parallel*

computational activity can happen concurrently at same-level gates.

Parallel time complexity of a circuit related to the circuit's *depth*.

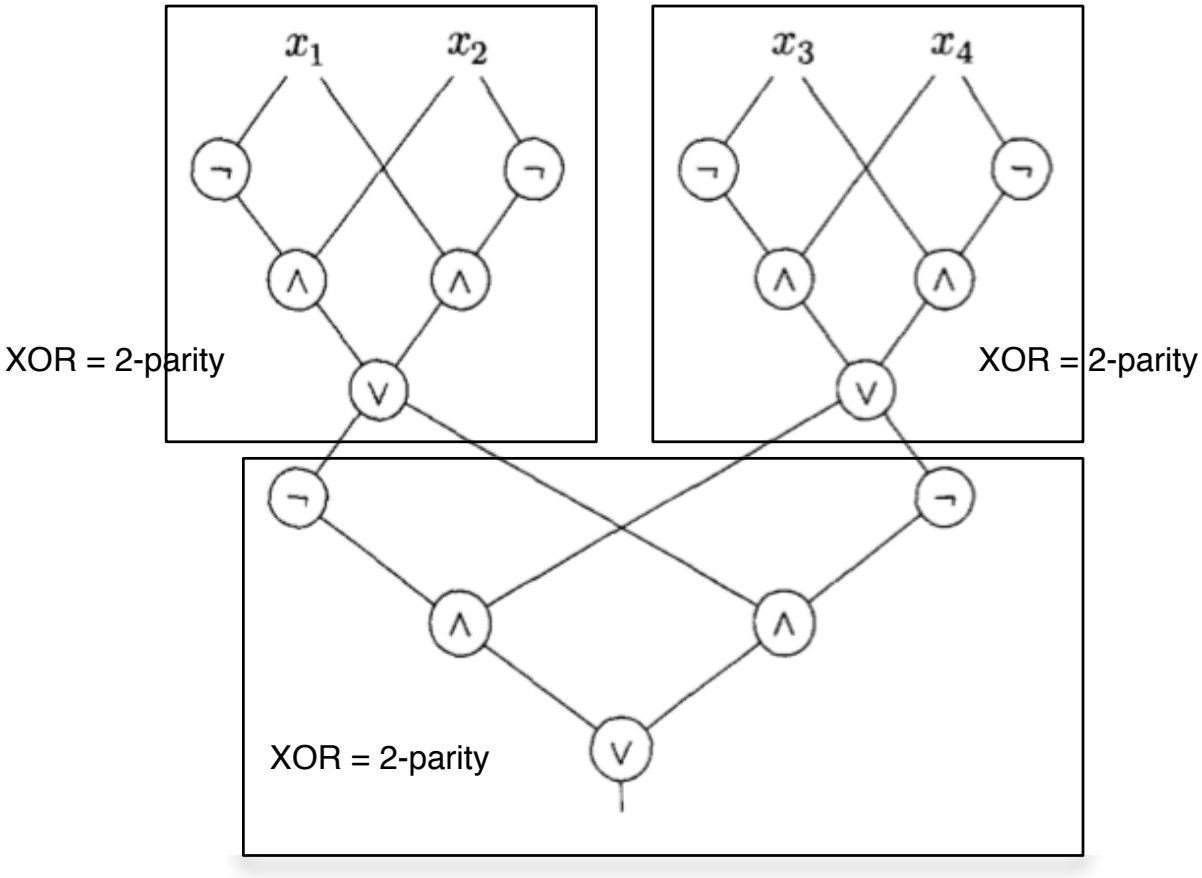
Simultaneous size-depth complexity of a language

$\mathcal{L}$ has simultaneous size-depth complexity $(f(n), g(n))$ if a uniform circuit family exists for $\mathcal{L}$ with

- size complexity $f(n)$ and
- depth complexity $g(n)$.

Parity

Parity is feasible in $(O(n), O(log(n)))$



The class NC

Definition. NC ("Nick's Class", after Nick Pippinger)
For $i \geq 0$, NC^i consists of all languages solvable in $(O(n^k), O(log^i(n)))$ with k an integer. Then, $\mathrm{NC} = \bigcup_i \mathrm{NC}^i$.

"polylogarithmic" depth

Nice features of NC

- Problems in NC are highly parallelisable with moderate amount of processors.
- Contains a wide range of relevant problems (e.g. standard arithmetic and matrix operations)

NC vs. L (or, LOGSPACE)

Theorem. $\mathsf{NC}^1 \subseteq \mathsf{L}$

Proof: Consider $\mathcal{L} \in \mathsf{NC}^1$ and an input w of length n.

General trick: Can construct "on the fly" C_n (and specific gates) from the uniform NC^1 family C deciding $\mathcal{L}$.

1. Evaluate C_n on w in a depth-first manner from the output gate.
   - AND gate: evaluate recursively the first predecessor; if false, then we are done. Otherwise evaluate the second predecessor.
   - OR gate: same principle.
   - NOT: evaluate the unique predecessor and return opposite value.
2. Record only the path to current gate and intermediate results

   Amount we need to remember is logarithmic since the circuit has logarithmic depth!

NC vs. NL (or, NLOGSPACE)

Theorem. $\mathrm{NL} \subseteq \mathrm{NC}^2$

Proof (incomplete, just some ideas): Consider w of length n and a TM M for $\mathcal{L} \in \mathrm{NL}$.

1. Construct (in log. space) the graph G_n of all possible configurations of M for an input of length n.
   - Nodes of G_n are the (polynomially many) configurations of M, i.e.:
     - State
     - Contents of work tape
     - Input tape head position and work tape head position
   - Given nodes c_1 and c_2 with c_1 input tape head position i
     - Add edge (c_1, c_2) labeled w_i if c_1 yields c_2 when $w_i = 1$
     - Add edge (c_1, c_2) labeled $\overline{w_i}$ if c_1 yields c_2 when $w_i = 0$
     - Add edge (c_1, c_2) unlabeled if c_1 yields c_2 regardless of w_i.
2. Build circuit C_n computing reachability over G_n w.r.t. input w
   Can be done in $O(log^2 n)$ depth.

NC vs. P

Theorem. NC $\subseteq$ P

Proof

Let $\mathcal{L} \in$ NC be decided by a uniform circuit family C.
On input w of length n proceed as follows:

1. Construct C_n (using logarithmic space)
2. Evaluate (in polynomial time) the circuit on input w
   - C_n has n^k gates for some k
   - Circuits can be evaluated in time polynomial in the number of gates

An interesting open question is whether P $\subseteq$ NC

We believe that this is not the case

$\Rightarrow$ not all tractable problems seem highly parallelizable!

The Class AC^0

So far we have restricted AND and OR gates to have 2 inputs.

Definition: The class AC^i
analogous to NC^i for circuits with arbitrary fan-in gates.

We have the following hierarchy:

$$NC^0 \subseteq AC^0 \subseteq NC^1 \subseteq AC^1 \subseteq \ldots$$

NC^0: functions that depend on $O(1)$ input bits ("juntas") — very limited!

But AC^0 is interesting:

- Arbitrary fan-in AND and OR gates
- Polynomial number of gates
- *Constant depth*

The power of AC^0

$$AC^0 \subseteq NC^1 \subseteq L \subseteq NL \subseteq NC^2 \subseteq P$$

However, a great deal can be accomplished within AC^0

- Integer addition
- Integer subtraction
- Even the evaluation of a (fixed) Relational Algebra query.

Addition in AC^0

Construct a circuit $C(x_n, \ldots, x_1, y_n, \ldots, y_1)$

- Input are binary numbers $x_n, \ldots, x_1$ and $y_n, \ldots, y_1$
- We have $n+1$ outputs $z_{n+1}, z_n, \ldots, z_1$ (a minor relaxation)

Notation:

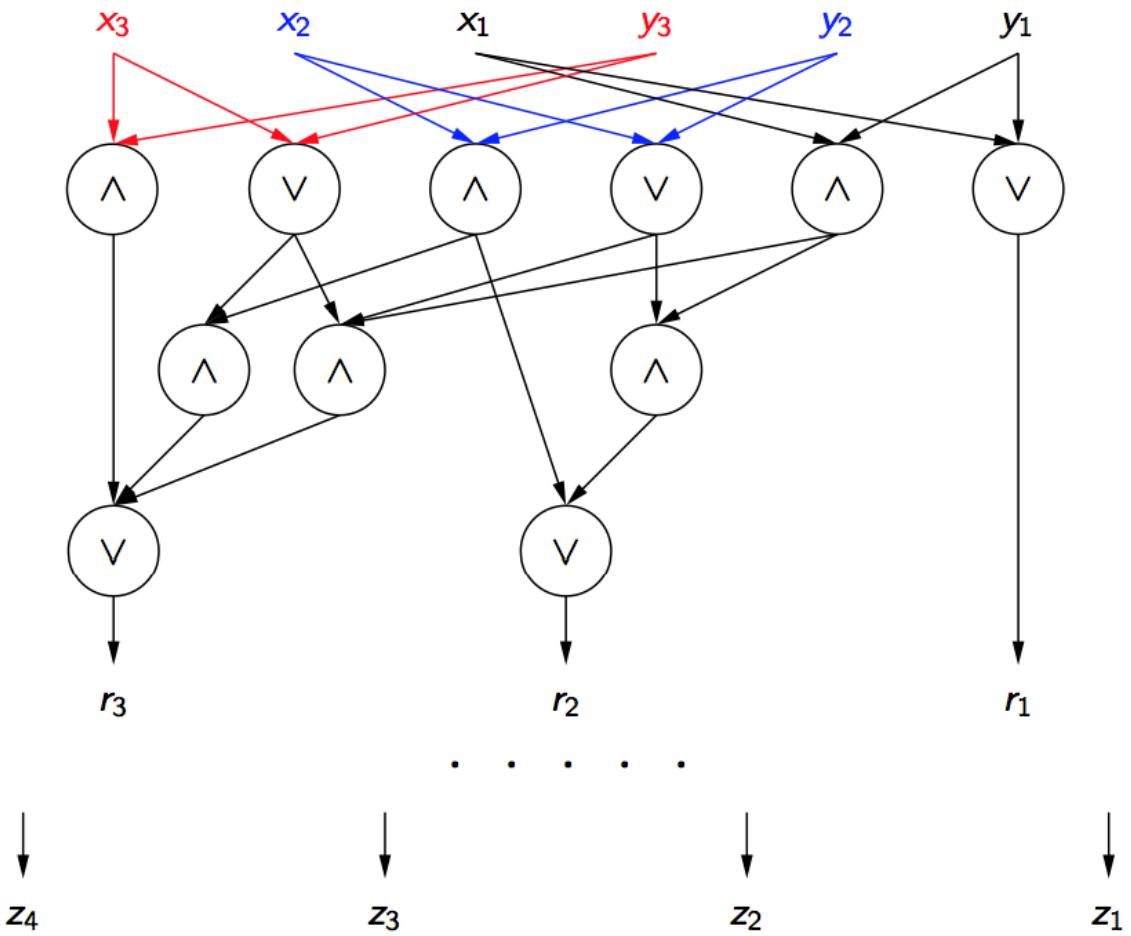
$$\begin{aligned} \mathrm{AND}_i &= x_i \wedge y_i \\ \mathrm{OR}_i &= x_i \vee y_i \\ \mathrm{XOR}_i &= (x_i \wedge \neg y_i) \vee (\neg x_i \wedge y_i) \end{aligned}$$

Then, the "carried-over bit" c_i and result z_i are as follows (take $c_0 = 0$):

$$\begin{aligned} c_i &= \mathrm{AND}_i \vee (\mathrm{OR}_i \wedge c_{i-1}) \\ z_i &= (\neg \mathrm{OR}_i \wedge c_{i-1}) \vee (\mathrm{XOR}_i \wedge \neg c_{i-1}) \vee (\mathrm{AND}_i \wedge c_{i-1}) \end{aligned}$$

Note that $c_1 = \mathrm{AND}_1$, $z_1 = \mathrm{XOR}_1$ and $z_{n+1} = c_n$

Sum in AC^0



The limits of AC^0

Most interestingly, AC^0 has provable limitations!

Theorem. Parity is not feasible in AC^0

As a consequence $AC^0 \subset NC^1$

$$AC^0 \subset NC^1 \subseteq L \subseteq NL \subseteq NC^2 \subseteq P$$