Computational Complexity; slides 4, HT 2023 Polynomial hierarchy, (N)LOGSPACE, Circuit complexity, NC, AC, P-completeness

Paul W. Goldberg (Dept. of CS, Oxford)

HT 2023

The polynomial-time hierarchy

- NP: given an existentially-quantified QBF, is it true?
- co-NP: given a universally-quantified QBF, is it true?
- PSPACE: given an unrestricted QBF, is it true?

The polynomial-time hierarchy

- NP: given an existentially-quantified QBF, is it true?
- co-NP: given a universally-quantified QBF, is it true?
- PSPACE: given an unrestricted QBF, is it true?

"intermediate" problems:

- Evaluate formula of the form $\exists x_1, \ldots, x_n \forall y_1, \ldots, y_n \; \varphi$
- Evaluate formula of the form $\forall x_1, \ldots, x_n \exists y_1, \ldots, y_n \; \varphi$
- Evaluate formula of the form $\exists x_1, \ldots, x_n \forall y_1, \ldots, y_n \exists z_1, \ldots, z_n \; \varphi$
- etc.

$\rightsquigarrow$ yet more complexity classes! (seemingly)

Sipser, chapter 10.3 (brief mention); Arora/Barak Chapter 5

The polynomial-time hierarchy

There are multiple equivalent definitions of the classes of the polynomial hierarchy. — Wikipedia

Model of computation for (say) $\exists x_1, \ldots, x_n \forall y_1, \ldots, y_n \ \varphi$?

The polynomial-time hierarchy

There are multiple equivalent definitions of the classes of the polynomial hierarchy. — Wikipedia

Model of computation for (say) $\exists x_1, \ldots, x_n \forall y_1, \ldots, y_n \ \varphi$?

—Yes, poly-time alternating TM where $\exists$ states must precede $\forall$ states, in any computation.
Any such formula has ATM of this kind that solves it; any ATM can be converted to equivalent $\exists \ldots \forall$-formula.

—Another answer: in terms of oracle machines...

The polynomial-time hierarchy

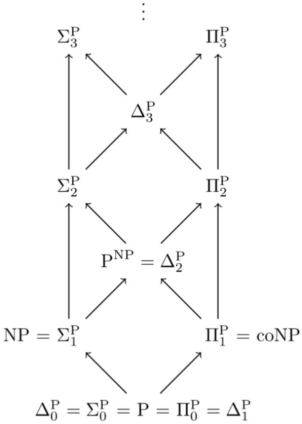


diagram taken from Wikipedia

$\Sigma_{i+1}^{\mathsf{P}} := \mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$

$\Pi_{i+1}^{\mathsf{P}} := \text{co-}\mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$, i.e. $co - \Sigma_{i+1}^{\mathsf{P}}$

$\Delta_{i+1}^{\mathsf{P}} := \mathsf{P}^{\Sigma_i^{\mathsf{P}}}$

A^B: problems solved by A-machine with oracle for B-complete problem

Warm-up: consider P^{P}, NP^{P}, P^{NP},...

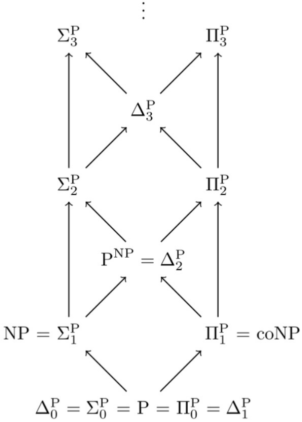


diagram taken from Wikipedia

$\Sigma_{i+1}^{\mathsf{P}} := \mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$
$\Pi_{i+1}^{\mathsf{P}} := \text{co-}\mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$, i.e. $co - \Sigma_{i+1}^{\mathsf{P}}$
$\Delta_{i+1}^{\mathsf{P}} := \mathsf{P}^{\Sigma_i^{\mathsf{P}}}$

A^B: problems solved by A-machine with oracle for B-complete problem

Warm-up: consider P^{P}, NP^{P}, P^{NP},...

P^{NP} seems to be more than just NP; indeed there are classes of interest intermediate between NP and P^{NP}!

Examples

Many diverse problems are complete for low levels of PH (link to compendium on course web page)

Example of a Σ_2^{P}-complete problem: MIN-DNF: consists of a DNF formula φ and integer k.
Question: is there a DNF formula ψ for which $\psi \equiv \varphi$ and ψ has size at most k?

Containment in Σ_2^{P}: note that the problem is of the form

$\exists$ (bit-string describing ψ) $\forall$ (valuations β of boolean variables) φ and ψ agree on β

Hardness requires $\exists x \forall y$(formula over variables x, y) to be efficiently encoded as (φ, k), instance of MIN-DNF...

The polynomial-time hierarchy

PH denotes the union of class in the hierarchy

Some key facts:

- PH lies below PSPACE; if any problem is complete for PH, it must belong to the *k*-th level of the hierarchy, and PH would "collapse" to that level
- Classes in PH are characterised by restricted alternating TMs
- If P is equal to NP, then PH would collapse to P (next slide)
- If NP is equal to co-NP, then PH collapses to NP. (hints that NP $\neq$ co-NP.)

If the *graph isomorphism problem* is NP-complete, then the PH collapses to the second level (Schöning 1987)
...evidence that the problem is not in fact NP-complete.

Theorem: If P is equal to NP, then PH would collapse to P

Proof: If P is equal to NP, it's also the same as co-NP

Recall the expressions

$$\Sigma^{\mathsf{P}}_{i+1} := \mathsf{NP}^{\Sigma^{\mathsf{P}}_i}$$
$$\Pi^{\mathsf{P}}_{i+1} := \text{co-}\mathsf{NP}^{\Sigma^{\mathsf{P}}_i}$$
$$\Delta^{\mathsf{P}}_{i+1} := \mathsf{P}^{\Sigma^{\mathsf{P}}_i}$$

and proceed by induction on i

i.e. $\Sigma^{\mathsf{P}}_2 = \mathsf{NP}^{\Sigma^{\mathsf{P}}_1} = \mathsf{NP}^{\mathsf{NP}}$ (by def, $\Sigma^{\mathsf{P}}_1 = \mathsf{NP}$)
$= \mathsf{P}^{\mathsf{P}}$ (by assumption of the theorem)
$= \mathsf{P}$
etc.

PH is "structure between NP and PSPACE": a sequence of classes that "seem" to all be different.

Next: Logarithmic space: structure within P

Logarithmic Space

Polynomial space: seems more powerful than NP.

Linear space: we noted is similar to polynomial space

Sub-linear space?

To be meaningful, we consider Turing machines with separate input tape and only count working space.

LOGSPACE (or, L)	Problems solvable by logarithmic space bounded TM
NLOGSPACE (or, NL)	Problems solvable by logarithmic space bounded NTM

Not hard to show that L⊆NL⊆P

(Sipser Chapter 8.4, Arora/Barak, p.80)

Problems in L and NL

What sort of problems are in L and NL?

In logarithmic space we can store

- a fixed number of counters (up to length of input)
- a fixed number of pointers to positions in the input string.

Problems in L and NL

What sort of problems are in L and NL?

In logarithmic space we can store

- a fixed number of counters (up to length of input)
- a fixed number of pointers to positions in the input string.

Hence,

- LOGSPACE contains all problems requiring only a constant number of counters/pointers for solving.
- NLOGSPACE contains all problems requiring only a constant number of counters/pointers for verifying solutions.

Example. The language $\{0^n 1^n : n \geq 0\}$

Algorithm.

- Check that no 1 is ever followed by a 0

  Requires no working space. (only movements of the read head)

- Count the number of 0's and 1's.
- Compare the two counters.

Examples: Problems in L

Example. The language $\{0^n 1^n : n \geq 0\}$

Algorithm.

- Check that no 1 is ever followed by a 0

  Requires no working space. (only movements of the read head)
- Count the number of 0's and 1's.
- Compare the two counters.

Example. PALINDROMES $\in$ LOGSPACE
(words that read the same forward and backward)

Algorithm.

- Use two pointers, one to the beginning and one to the end of the input.
- At each step, compare the two symbols pointed to.
- Move the pointers one step inwards.

Example: A Problem in NL

Example. The following problem is in NL:

> **REACHABILITY a.k.a. PATH**
> *Input:* Directed graph G, vertices $s, t \in V(G)$
> *Problem:* Does G contain a path from s to t?

Algorithm.

Set counter $c := |V(G)|$
Let pointer p point to s
while $c \neq 0$ **do**
  if $p = t$ **then** halt and accept
  else
    nondeterministically select a successor p' of p
    set $p := p'$
    $c := c - 1$
reject.

LOGSPACE Reductions

Polynomial-time reductions are too "coarse" to compare poly-time vs. log-space computability.

LOGSPACE Reductions

Polynomial-time reductions are too "coarse" to compare poly-time vs. log-space computability.

Definition. A LOGSPACE-transducer M is a TM with

- a read-only input tape
- a write only, write once output tape
- a memory tape of size $O(\log(n))$

M computes a function $f : \Sigma^* \to \Sigma^*$, where $f(w)$ is the content of the output tape of M running on input w when M halts.

f is called a logarithmic space computable function.

LOGSPACE Reductions

Polynomial-time reductions are too "coarse" to compare poly-time vs. log-space computability.

Definition. A LOGSPACE-transducer M is a TM with

- a read-only input tape
- a write only, write once output tape
- a memory tape of size $O(\log(n))$

M computes a function $f : \Sigma^* \to \Sigma^*$, where $f(w)$ is the content of the output tape of M running on input w when M halts.

f is called a logarithmic space computable function.

Definition.

A LOGSPACE reduction from $\mathcal{L} \subseteq \Sigma^*$ to $\mathcal{L}' \subseteq \Sigma^*$ is a log space computable function $f : \Sigma^* \to \Sigma^*$ such that for all $w \in \Sigma^*$:

$$w \in \mathcal{L} \Longleftrightarrow f(w) \in \mathcal{L}'$$

We write $\mathcal{L} \leq_L \mathcal{L}'$

NLOGSPACE (or, NL)-Completeness

A problem $\mathcal{L} \in \mathsf{NL}$ is complete for NL, if every other language in NL is log space reducible to $\mathcal{L}$.

Theorem. REACHABILITY (or, PATH) is NL-complete.

Proof idea. (details to follow)

Let M be a non-deterministic LOGSPACE TM deciding $\mathcal{L}$.

On input w:

1. construct a graph whose nodes are configurations of M and edges represent possible computational steps of M on w
2. Find a path from the start configuration to an accepting configuration.

NL-Completeness of PATH

some more details.

Construct $\langle G, s, t\rangle$ from M and w using a LOGSPACE-transducer:

1. A configuration $(q, w_2, (p_1, p_2))$ of M can be described in $c \log n$ space for some constant c and $n = |w|$.
2. List the nodes of G by going through all strings of length $c \log n$ and outputting those that correspond to legal configurations.
3. List the edges of G by going through all pairs of strings (C_1, C_2) of length $c \log n$ and outputting those pairs where $C_1 \vdash_M C_2$.
4. s is the starting configuration of G.
5. Assume w.l.o.g. that M has a single accepting configuration t.

$w \in \mathcal{L}$ iff $\langle G, s, t\rangle \in$ REACHABILITY

(see Sipser Thm. 8.25)

co-NLOGSPACE

As for time, we consider complement classes for space.

Recall

If $\mathcal{C}$ is a complexity class, we define

$$\text{co-}\mathcal{C} := \{\mathcal{L} : \overline{\mathcal{L}} \in \mathcal{C}\}.$$

From Savitch's theorem:

PSPACE = NPSPACE and hence co-NPSPACE = PSPACE

NLOGSPACE = co-NLOGSPACE

However, from Savitch's theorem we only know

$$\text{NLOGSPACE} \subseteq \text{DSPACE}(\log^2 n).$$

Theorem. (Immerman and Szelepcsényi '87-8)

NLOGSPACE = co-NLOGSPACE

Proof idea.

Show that $\overline{\text{REACHABILITY}}$ is in NL.

Proof sketch. On input $\langle G, s, t\rangle$, let $m = |V(G)|$.

Define c_i to be number of nodes reachable from s in $\leq i$ steps; compute c_i for increasing $i = 1, 2, \ldots, m$

1. Only node s is reachable in 0 steps, so $c_0 = 1$
2. For each $i = 1, \ldots, m$, set $c_i = 1$, remember c_{i-1}, and for each $v \neq s$ in G
    1. $d := 0$
    2. For each node u in G
        1. guess if reachable from s in $\leq i - 1$ steps, if so do (2,3):
        2. Verify each "yes" guess by guessing an at most $i - 1$ step path from s to u; if so, $d := d + 1$; reject if no such path found
        3. If we guessed that u is reachable, and $(u, v) \in E(G)$, then increment c_i and continue with next v
    3. If total number d of u guessed is not equal to c_{i-1}, then reject

Continued...

NLOGSPACE = co-NLOGSPACE

Proof sketch (continued). On input $\langle G, s, t\rangle$

(at this stage we have c_m)

Then try to guess c_m nodes reachable from s and not equal to t:

1. For each node u in G, guess if reachable from s in m steps
2. Verify each "yes" guess by guessing a $\leq m$ step path from s to u; reject if no such path found
3. If we guessed that u is reachable, and $u = t$, then reject
4. If total number d of u guessed not equal to c_m, then reject
5. Otherwise accept

Algorithm stores (at one time) only 6 counters (u, v, c_{i-1}, c_i, d and i) and a pointer to the head of a path; hence runs in logspace.

(more details in Sipser Theorem 8.27)

To summarise

It's unknown where L is equal to NL, or if NL is equal to P.

$$\mathsf{L} \subseteq \mathsf{NL} = \text{co-}\mathsf{NL} \subseteq \mathsf{P}$$

Still, we have that NL is closed under complement — contrast with NP

By space hierarchy theorem, L $\subsetneq$ PSPACE
Indeed (from s.h.t. and Savitch's theorem) NL $\subsetneq$ PSPACE

Next: more structure within P: Circuit complexity, NC, AC, P-completeness
(also P/poly)

Boolean Circuits: another model of computation

A standard mathematical model of "digital circuit"

A Boolean circuit is a DAG:

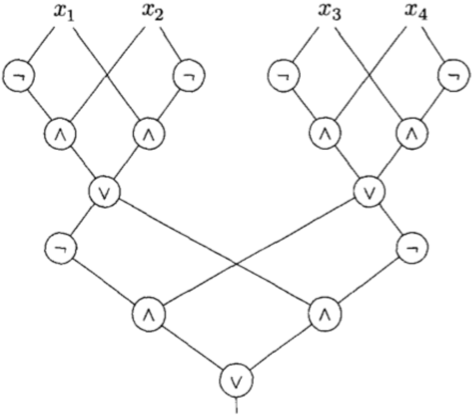
- *Inputs* : nodes without incoming edges labeled with 0 or 1.
- *Gates* : nodes with (one or two) incoming edges and one outgoing edge labeled AND, OR, or NOT.
- A single node is labeled as *output*.

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

Input-output behaviour described using *Boolean functions*

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

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

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

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

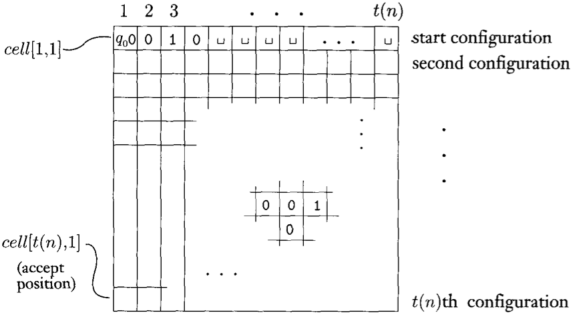
Small time complexity $\Rightarrow$ small circuit complexity

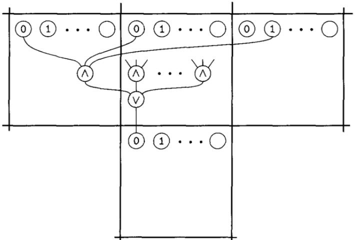
Theorem. If $\mathcal{L} \in \text{DTIME}(t(n))$ with $t(n) \geq n$ then $\mathcal{L}$ has circuit-size complexity $O(t^2(n))$

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 $\mathsf{P} \neq \mathsf{NP}$.

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, $\mathsf{P} \subseteq \mathsf{P/poly}$

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 in P/poly *(i.e. 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).

Boolean circuits are genuinely *parallel*

computational activity can happen concurrently at same-level gates.

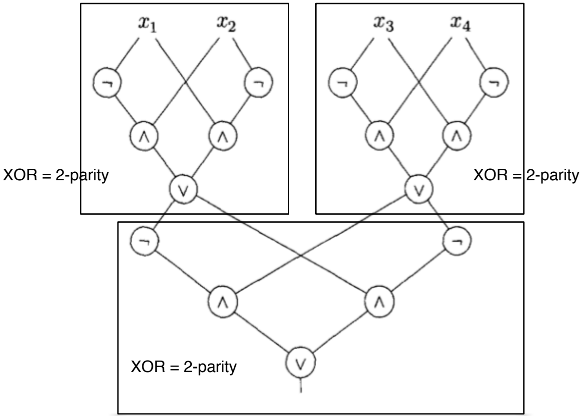
Parallel time complexity of a circuit related to the circuit's *depth*.

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 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, $\mathsf{NC} = \bigcup_i \mathsf{NC}^i$.

"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. NL $\subseteq$ NC^2

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

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

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

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

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:

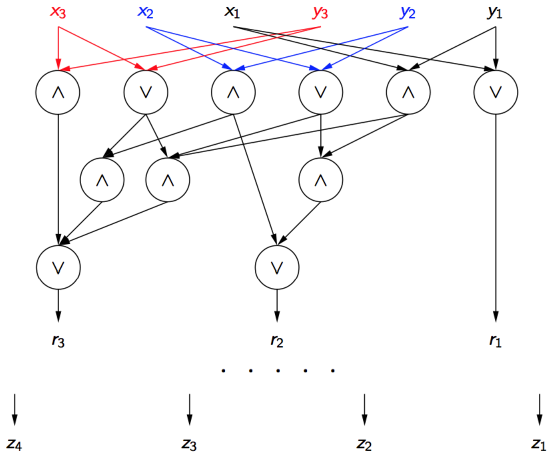
$$\begin{aligned}
\mathrm{AND}_i &= x_i \wedge y_i \\
\mathrm{OR}_i &= x_i \vee y_i \\
\mathrm{XOR}_i &= (x_i \wedge \neg y_i) \vee (\neg x_i \wedge y_i)
\end{aligned}$$

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 $\mathsf{AC}^0 \subset \mathsf{NC}^1$

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