COMPLEXITY THEORY

Lecture 18: Polynomial Hierarchy / Circuit Complexity

Markus Krötzsch
Knowledge-Based Systems

TU Dresden, 17th Dec 2018

More about the Polynomial Hierarchy

The Polynomial Hierarchy Three Ways

We discovered a hierarchy of complexity classes between P and PSpace, with NP and coNP on the first level, and infinitely many further levels above:

Definition by ATM: Classes $\Sigma_i^{\mathsf{P}}/\Pi_i^{\mathsf{P}}$ are defined by polytime ATMs with bounded types of alternation, starting computation with existential/universal states.

Definition by Verifier: Classes $\Sigma_i^{\mathsf{P}}/\Pi_i^{\mathsf{P}}$ are given as projections of certain verifier languages in P, requiring existence/universality of polynomial witnesses.

Definition by Oracle: Classes $\Sigma_i^{\mathsf{P}}/\Pi_i^{\mathsf{P}}$ are defined as languages of NP/coNP oracle TMs with $\Sigma_{i-1}^{\mathsf{P}}$ (or, equivalently, Π_{i-1}^{P}) oracle.

Using such oracles with deterministic TMs, we can also define classes Δ_i^{P}.

More Classes in PH

We defined Σ_k^{P} and Π_k^{P} by relativising NP and coNP with oracles.

What happens if we start from P instead?

Definition 18.1: $\Delta_0^{\mathsf{P}} := \mathsf{P}$ and $\Delta_{k+1}^{\mathsf{P}} := \mathsf{P}^{\Sigma_k^{\mathsf{P}}}$.

Some immediate observations:

- $\Delta_1^{\mathsf{P}} = \mathsf{P}^{\mathsf{P}} = \mathsf{P}$
- $\Delta_2^{\mathsf{P}} = \mathsf{P}^{\mathsf{NP}} = \mathsf{P}^{\mathsf{coNP}}$
- $\Delta_k^{\mathsf{P}} \subseteq \Sigma_k^{\mathsf{P}}$ (since $\mathsf{P} \subseteq \mathsf{NP}$) and $\Delta_k^{\mathsf{P}} \subseteq \Pi_k^{\mathsf{P}}$ (since $\mathsf{P} \subseteq \mathsf{coNP}$)
- $\Sigma_k^{\mathsf{P}} \subseteq \Delta_{k+1}^{\mathsf{P}}$ and $\Pi_k^{\mathsf{P}} \subseteq \Delta_{k+1}^{\mathsf{P}}$

Problems for Δ_k^{P}?

Δ_k^{P} seems to be less common in practice, but there are some known complete problems for $\mathrm{P}^{\mathrm{NP}} = \Delta_2^{\mathrm{P}}$:

Uniquely Optimal TSP [Papadimitriou, JACM 1984]

Input:	Undirected graph G with edge weights (distances).
Problem:	Is there exactly one shortest travelling salesman tour on G?

Divisible TSP [Krentel, JCSS 1988]

Input:	Undirected graph G with edge weights; number k.
Problem:	Is the shortest travelling salesman tour on G divisible by k?

Odd Final SAT [Krentel, JCSS 1988]

Input:	Propositional formula φ with n variables.
Problem:	Is X_n true in the lexicographically last assignment satisfying φ?

Is the Polynomial Hierarchy Real?

Questions:

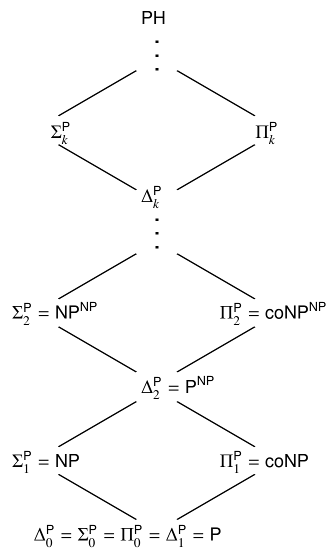
Are all of these classes really distinct?
Nobody knows.

Are any of these classes really distinct?
Nobody knows.

Are any of these classes distinct from P?
Nobody knows.

Are any of these classes distinct from PSpace?
Nobody knows.

What do we know then?



What We Know (Excerpt)

Theorem 18.2: If there is any k such that $\Sigma_k^{\mathrm{P}} = \Sigma_{k+1}^{\mathrm{P}}$ then $\Sigma_j^{\mathrm{P}} = \Pi_j^{\mathrm{P}} = \Sigma_k^{\mathrm{P}}$ for all $j > k$, and therefore $\mathrm{PH} = \Sigma_k^{\mathrm{P}}$.
In this case, we say that the polynomial hierarchy collapses at level k.

Proof: Left as exercise (not too hard to get from definitions). □

Corollary 18.3: If PH ≠ P then NP ≠ P.

Intuitively speaking: "The polynomial hierarchy is built upon the assumption that NP has some additional power over P. If this is not the case, the whole hierarchy collapses."

What We Know (Excerpt)

Theorem 18.4: PH ⊆ PSpace.

Proof: Left as exercise (induction over PH levels, using that $\mathrm{PSpace}^{\mathrm{PSpace}} = \mathrm{PSpace}$). □

Theorem 18.5: If PH = PSpace then there is some k with $\mathrm{PH} = \Sigma_k^{\mathrm{P}}$.

Proof: If PH = PSpace then **True QBF** ∈ PH. Hence **True QBF** $\in \Sigma_k^{\mathrm{P}}$ for some k. Since **True QBF** is PSpace-hard, this implies $\Sigma_k^{\mathrm{P}} = \mathrm{PSpace}$. □

What We Believe (Excerpt)

"Most experts" think that:

- The polynomial hierarchy does not collapse completely (same as P ≠ NP)
- The polynomial hierarchy does not collapse on any level
  (in particular PH ≠ PSpace and there is no PH-complete problem)

But there can always be surprises . . .

Computing with Circuits

Motivation

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

Some questions:

- Even if it is hard to find a universal algorithm for solving all instances of a problem, couldn't it still be that there is a simple algorithm for every fixed problem size?
- What can complexity theory tell us about parallel computation?
- Are there any meaningful complexity classes below LogSpace? Do they contain relevant problems?

$\leadsto$ circuit complexity provides some answers

Intuition: use circuits with logical gates to model computation

Boolean Circuits

Definition 18.6: A Boolean circuit is a finite, directed, acyclic graph where

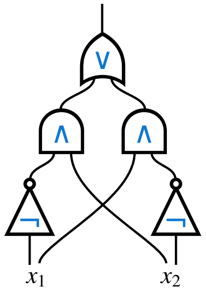
- each node that has no predecessor is an input node
- each node that is not an input node is one of the following types of logical gate:
  - AND with two input wires
  - OR with two input wires
  - NOT with one input wire
- one or more nodes are designated output nodes

The outputs of a Boolean circuit are computed in the obvious way from the inputs.
$\leadsto$ circuits with k inputs and ℓ outputs represent functions $\{0,1\}^k \to \{0,1\}^\ell$

We often consider circuits with only one output.

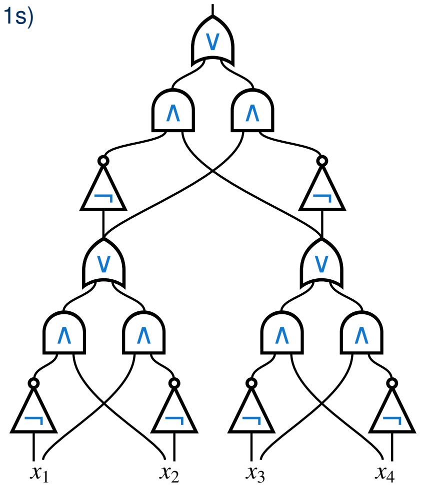
Example 1

XOR function:



Example 2

Parity function with four inputs:
(true for odd number of 1s)

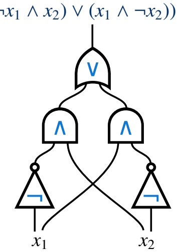


Alternative Ways of Viewing Circuits (1)

Propositional formulae

- propositional formulae are special circuits:
  each non-input node has only one outgoing wire
- each variable corresponds to one input node
- each logical operator corresponds to a gate
- each sub-formula corresponds to a wire

$((\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2))$

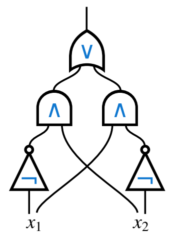


Alternative Ways of Viewing Circuits (2)

Straight-line programs

- are programs without loops and branching (if, goto, for, while, etc.)
- that only have Boolean variables
- and where each line can only be an assignment with a single Boolean operator

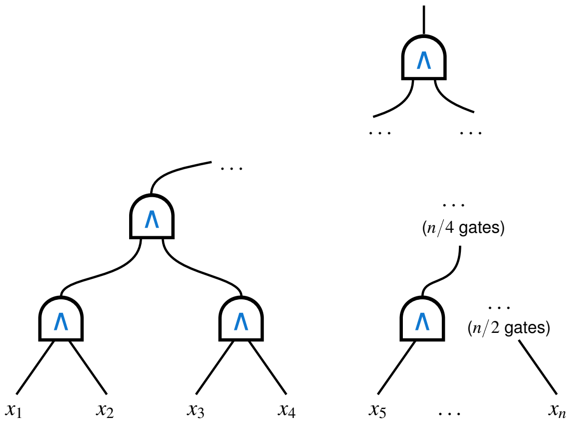
$\leadsto$ n-line programs correspond to n-gate circuits



01 z_1 := $\neg x_1$
02 z_2 := $\neg x_2$
03 z_3 := $z_1 \wedge x_2$
04 z_4 := $z_2 \wedge x_1$
05 return $z_3 \vee z_4$

Example: Generalised AND

The function that tests if all inputs are 1 can be encoded by combining binary AND gates:



- works similarly for OR gates
- number of gates: $n - 1$
- we can use n-way AND and OR (keeping the real size in mind)

Solving Problems with Circuits

Circuits are not universal: they have a fixed number of inputs!
How can they solve arbitrary problems?

Definition 18.7: A circuit family is an infinite list $C = C_1, C_2, C_3, \ldots$ where each C_i is a Boolean circuit with i inputs and one output.
We say that C decides a language $\mathbf{L}$ (over $\{0, 1\}$) if

$$w \in \mathbf{L} \quad \text{if and only if} \quad C_n(w) = 1 \text{ for } n = |w|.$$

Example 18.8: The circuits we gave for generalised AND are a circuit family that decides the language $\{1^n \mid n \geq 1\}$.

Circuit Complexity

To measure difficulty of problems solved by circuits, we can count the number of gates needed:

Definition 18.9: The size of a circuit is its number of gates.

Let $f : \mathbb{N} \to \mathbb{R}^+$ be a function. A circuit family C is f-size bounded if each of its circuits C_n is of size at most $f(n)$.

Size$(f(n))$ is the class of all languages that can be decided by an $O(f(n))$-size bounded circuit family.

Example 18.10: Our circuits for generalised AND show that $\{1^n \mid n \geq 1\} \in$ Size(n).

Examples

Many simple operations can be performed by circuits of polynomial size:

- Boolean functions such as parity (=sum modulo 2), sum modulo n, or majority
- Arithmetic operations such as addition, subtraction, multiplication, division (taking two fixed-arity binary numbers as inputs)
- Many matrix operations

See exercise for some more examples

Polynomial Circuits

Polynomial Circuits

A natural class of problems to consider are those that have polynomial circuit families:

Definition 18.11: $\mathsf{P}_{/\mathrm{poly}} = \bigcup_{d\geq 1} \mathsf{Size}(n^d)$.

Note: A language is in $\mathsf{P}_{/\mathrm{poly}}$ if it is solved by some polynomial-sized circuit family. There may not be a way to compute (or even finitely represent) this family.

How does $\mathsf{P}_{/\mathrm{poly}}$ relate to other classes?

Quadratic Circuits for Deterministic Time

Theorem 18.12: For $f(n) \geq n$, we have $\mathsf{DTime}(f) \subseteq \mathsf{Size}(f^2)$.

Proof sketch (see also Sipser, Theorem 9.30)

- We can represent the DTime computation as in the proof of Theorem 16.10: as a list of configurations encoded as words

  $$* \, \sigma_1 \, \cdots \, \sigma_{i-1} \, \langle q, \sigma_i \rangle \, \sigma_{i+1} \, \cdots \, \sigma_m \, *$$

  of symbols from the set $\Omega = \{*\} \cup \Gamma \cup (Q \times \Gamma)$.
  $\leadsto$ Tableau (i.e., grid) with $O(f^2)$ cells.
- We can describe each cell with a list of bits (wires in a circuit).
- We can compute one configuration from its predecessor by $O(f)$ circuits (idea: compute the value of each cell from its three upper neighbours as in Theorem 16.10)
- Acceptance can be checked by assuming that the TM returns to a unique configuration position/state when accepting $\square$

From Polynomial Time to Polynomial Size

From $\mathsf{DTime}(f) \subseteq \mathsf{Size}(f^2)$ we get:

Corollary 18.13: $\mathsf{P} \subseteq \mathsf{P}_{/\mathrm{poly}}$.

This suggests another way of approaching the P vs. NP question:

If any language in NP is not in $\mathsf{P}_{/\mathrm{poly}}$, then $\mathsf{P} \neq \mathsf{NP}$.
(but nobody has found any such language yet)

Circuit-Sat

Input:	A Boolean Circuit C with one output.
Problem:	Is there any input for which C returns 1?

Theorem 18.14: Circuit-Sat is NP-complete.

Proof: Inclusion in NP is easy (just guess the input).

For NP-hardness, we use that NP problems are those with a P-verifier:

- The DTM simulation of Theorem 18.12 can be used to implement a verifier (input: $(w\#c)$ in binary)
- We can hard-wire the w-inputs to use a fixed word instead (remaining inputs: c)
- The circuit is satisfiable iff there is a certificate for which the verifier accepts w □

Note: It would also be easy to reduce **Sat** to **Circuit-Sat**, but the above yields a proof from first principles.

A New Proof for Cook-Levin

Theorem 18.15: 3Sat is NP-complete.

Proof: Membership in NP is again easy (as before).

For NP-hardness, we express the circuit that was used to implement the verifier in Theorem 18.14 as propositional logic formula in 3-CNF:

- Create a propositional variable X for every wire in the circuit
- Add clauses to relate input wires to output wires, e.g., for AND gate with inputs X_1 and X_2 and output X_3, we encode $(X_1 \wedge X_2) \leftrightarrow X_3$ as:

$$(\neg X_1 \vee \neg X_2 \vee X_3) \wedge (X_1 \vee \neg X_3) \wedge (X_2 \vee \neg X_3)$$

- Fixed number of clauses per gate = constant factor size increase
- Add a clause (X) for the output wire X □

Summary and Outlook

We do not know if the Polynomial Hierarchy is real or collapses

Circuits provide an alternative model of computation

$\mathsf{P} \subseteq \mathsf{P}_{/\mathrm{poly}}$

Circuit-Sat is NP-complete.

What's next?

- Circuits for parallelism
- Complexity classes (strictly!) below P
- Randomness