TECHNISCHE UNIVERSITÄT DRESDEN

COMPLEXITY THEORY

Lecture 18: Questions and Answers

Markus Krötzsch

Knowledge-Based Systems

TU Dresden, 20th Dec 2017

The Power of Circuits

Review

What we learned in the previous lecture:

- Circuits provide an alternative model of computation
- $\mathsf{P} \subseteq \mathsf{P}_{/\mathrm{poly}}$
- **Circuit-Sat** is NP-complete

Is $\mathsf{P} = \mathsf{P}_{/\mathrm{poly}}$?

We showed $\mathsf{P} \subseteq \mathsf{P}_{/\mathrm{poly}}$. Does the converse also hold?

Is $\mathsf{P} = \mathsf{P}_{/\mathrm{poly}}$?

We showed $\mathsf{P} \subseteq \mathsf{P}_{/\mathrm{poly}}$. Does the converse also hold?

No!

Theorem 18.1: $\mathsf{P}_{/\mathrm{poly}}$ contains undecidable problems.

Is P = $P_{/\mathrm{poly}}$?

We showed P ⊆ $P_{/\mathrm{poly}}$. Does the converse also hold?

No!

Theorem 18.1: $P_{/\mathrm{poly}}$ contains undecidable problems.

Proof: We define the unary Halting problem as the (undecidable) language:

$$\textsc{UHalt} := \{1^n \mid \text{the binary encoding of } n \text{ encodes a pair } \langle \mathcal{M}, w\rangle \text{ where } \mathcal{M} \text{ is a TM that halts on word } w\}$$

For a number $1^n \in$ **UHalt**, let C_n be the circuit that computes a generalised AND of all inputs. For all other numbers, let C_n be a circuit that always returns 0. The circuit family $C_1, C_2, C_3, \ldots$ accepts **UHalt**. □

Uniform Circuit Families

$\mathsf{P}_{/\mathrm{poly}}$ is too powerful, since we do not require the circuits to be computable.
We can add this requirement:

Definition 18.2: A circuit family $C_1, C_2, C_3, \ldots$ is log-space-uniform if there is a log-space computable function that maps words 1^n to (an encoding of) C_n.

Note: We could also define similar notions of uniformity for other complexity classes.

Uniform Circuit Families

$P_{/\text{poly}}$ is too powerful, since we do not require the circuits to be computable.
We can add this requirement:

Definition 18.2: A circuit family $C_1, C_2, C_3, \ldots$ is log-space-uniform if there is a log-space computable function that maps words 1^n to (an encoding of) C_n.

Note: We could also define similar notions of uniformity for other complexity classes.

Theorem 18.3: The class of all languages that are accepted by a log-space-uniform circuit family of polynomial size is exactly P.

Proof sketch: A detailed analysis shows that our earlier reduction of polytime DTMs to circuits is log-space-uniform. Conversely, a polynomial-time procedure can be obtained by first computing a suitable circuit (in log-space) and then evaluating it (in polynomial time). □

Turing Machines That Take Advice

One can also describe $\mathsf{P}_{/\mathrm{poly}}$ using TMs that take “advice”:

Definition 18.4: Consider a function $a : \mathbb{N} \to \mathbb{N}$. A language **L** is accepted by a Turing Machine $\mathcal{M}$ with a bits of advice if there is a sequence of advice strings $\alpha_0, \alpha_1, \alpha_2, \ldots$ of length $|\alpha_i| = a(i)$ and $\mathcal{M}$ accepts inputs of the form $(w\#\alpha_{|w|})$ if and only if $w \in$ **L**.

$\mathsf{P}_{/\mathrm{poly}}$ is equivalent to the class of problems that can be solved by a PTime TM that takes a polynomial amount of “advice” (where the advice can be a description of a suitable circuit).

(This is where the notation $\mathsf{P}_{/\mathrm{poly}}$ comes from.)

$\mathsf{P}_{/\mathrm{poly}}$ and NP

We showed $\mathsf{P} \subseteq \mathsf{P}_{/\mathrm{poly}}$. Does $\mathsf{NP} \subseteq \mathsf{P}_{/\mathrm{poly}}$ also hold?

$\mathsf{P}_{/\mathrm{poly}}$ and NP

We showed $\mathsf{P} \subseteq \mathsf{P}_{/\mathrm{poly}}$. Does $\mathsf{NP} \subseteq \mathsf{P}_{/\mathrm{poly}}$ also hold?
Nobody knows.

Theorem 18.5 (Karp-Lipton Theorem): If $\mathsf{NP} \subseteq \mathsf{P}_{/\mathrm{poly}}$ then $\mathsf{PH} = \Sigma_2^p$.

$\mathsf{P}_{/\mathrm{poly}}$ and NP

We showed $\mathsf{P} \subseteq \mathsf{P}_{/\mathrm{poly}}$. Does $\mathsf{NP} \subseteq \mathsf{P}_{/\mathrm{poly}}$ also hold?
Nobody knows.

Theorem 18.5 (Karp-Lipton Theorem): If $\mathsf{NP} \subseteq \mathsf{P}_{/\mathrm{poly}}$ then $\mathsf{PH} = \Sigma_2^p$.

Proof sketch (see Arora/Barak Theorem 6.19):

- if $\mathsf{NP} \subseteq \mathsf{P}_{/\mathrm{poly}}$ then there is a polysize circuit family solving `Sat`
- Using this, one can argue that there is also a polysize circuit family that computes the lexicographically first satisfying assignment (k output bits for k variables)
- A Π_2-QBF formula $\forall\vec{X}.\exists\vec{Y}.\varphi$ is true if, for all values of $\vec{X}$, $\varphi(\vec{X})$ is satisfiable.
- In Σ_2^{P}, we can: (1) guess the polysize circuit for SAT, (2) check for all values of $\vec{X}$ if its output is really a satisfying assignment (to verify the guess)
- This solves Π_2^{P}-hard problems in Σ_2^{P}
- But then the Polynomial Hierarchy collapses at Σ_2^{P}, as claimed. $\square$

$\mathsf{P}_{/\mathrm{poly}}$ and ExpTime

We showed $\mathsf{P} \subseteq \mathsf{P}_{/\mathrm{poly}}$. Does ExpTime $\subseteq \mathsf{P}_{/\mathrm{poly}}$ also hold?

$\mathsf{P}_{/\mathrm{poly}}$ and ExpTime

We showed $\mathsf{P} \subseteq \mathsf{P}_{/\mathrm{poly}}$. Does ExpTime $\subseteq \mathsf{P}_{/\mathrm{poly}}$ also hold?
Nobody knows.

Theorem 18.6 (Meyer's Theorem):
If ExpTime $\subseteq \mathsf{P}_{/\mathrm{poly}}$ then ExpTime $=$ PH $= \Sigma_2^p$.

See [Arora/Barak, Theorem 6.20] for a proof sketch.

$\mathsf{P}_{/\mathrm{poly}}$ and ExpTime

We showed $\mathsf{P} \subseteq \mathsf{P}_{/\mathrm{poly}}$. Does ExpTime $\subseteq \mathsf{P}_{/\mathrm{poly}}$ also hold?
Nobody knows.

Theorem 18.6 (Meyer's Theorem):
If ExpTime $\subseteq \mathsf{P}_{/\mathrm{poly}}$ then ExpTime $=$ PH $= \Sigma_2^p$.

See [Arora/Barak, Theorem 6.20] for a proof sketch.

Corollary 18.7: If ExpTime $\subseteq \mathsf{P}_{/\mathrm{poly}}$ then P $\neq$ NP.

Proof: If ExpTime $\subseteq \mathsf{P}_{/\mathrm{poly}}$ then ExpTime $= \Sigma_2^p$ (Meyer's Theorem).
By the Time Hierarchy Theorem, P $\neq$ ExpTime, so P $\neq \Sigma_2^p$.
So the Polynomial Hierarchy doesn't collapse completely, and P $\neq$ NP. □

How Big a Circuit Could We Need?

We should not be surprised that $\mathsf{P}_{/\mathrm{poly}}$ is so powerful:
exponential circuit families are already enough to accept any language

Exercise: show that every Boolean function over n variables can be expressed by a circuit of size $\leq n2^n$.

How Big a Circuit Could We Need?

We should not be surprised that $\mathsf{P}_{/\mathrm{poly}}$ is so powerful:
exponential circuit families are already enough to accept any language

Exercise: show that every Boolean function over n variables can be expressed by a circuit of size $\leq n2^n$.

It turns out that these exponential circuits are really needed:

Theorem 18.8 (Shannon 1949 (!)): For every n, there is a function $\{0,1\}^n \to \{0,1\}$ that cannot be computed by any circuit of size $2^n/(10n)$.

In fact, one can even show: almost every Boolean function requires circuits of size $> 2^n/(10n)$ – and is therefore not in $\mathsf{P}_{/\mathrm{poly}}$

How Big a Circuit Could We Need?

We should not be surprised that $\mathsf{P}_{/\mathrm{poly}}$ is so powerful:
exponential circuit families are already enough to accept any language

Exercise: show that every Boolean function over n variables can be expressed by a circuit of size $\leq n2^n$.

It turns out that these exponential circuits are really needed:

Theorem 18.8 (Shannon 1949 (!)): For every n, there is a function $\{0,1\}^n \rightarrow \{0,1\}$ that cannot be computed by any circuit of size $2^n/(10n)$.

In fact, one can even show: almost every Boolean function requires circuits of size $> 2^n/(10n)$ – and is therefore not in $\mathsf{P}_{/\mathrm{poly}}$

Is any of these functions in NP? Or at least in Exp? Or at least in NExp?

How Big a Circuit Could We Need?

We should not be surprised that $\mathsf{P}_{/\mathrm{poly}}$ is so powerful:
exponential circuit families are already enough to accept any language

Exercise: show that every Boolean function over n variables can be expressed by a circuit of size $\leq n2^n$.

It turns out that these exponential circuits are really needed:

Theorem 18.8 (Shannon 1949 (!)): For every n, there is a function $\{0,1\}^n \to \{0,1\}$ that cannot be computed by any circuit of size $2^n/(10n)$.

In fact, one can even show: almost every Boolean function requires circuits of size $> 2^n/(10n)$ – and is therefore not in $\mathsf{P}_{/\mathrm{poly}}$

Is any of these functions in NP? Or at least in Exp? Or at least in NExp?
Nobody knows.

Question 1: The Logarithmic Hierarchy

Q1: The Logarithmic Hierarchy

The Polynomial Hierarchy is based on polynomially time-bounded TMs

It would also be interesting to study the Logarithmic Hierarchy
obtained by considering logarithmically space-bounded TMs instead

Q1: The Logarithmic Hierarchy

The Polynomial Hierarchy is based on polynomially time-bounded TMs

It would also be interesting to study the Logarithmic Hierarchy obtained by considering logarithmically space-bounded TMs instead, wouldn't it?

Q1: The Logarithmic Hierarchy

The Polynomial Hierarchy is based on polynomially time-bounded TMs

It would also be interesting to study the Logarithmic Hierarchy obtained by considering logarithmically space-bounded TMs instead, wouldn't it?

In detail, we can define:

- $\Sigma_0^{\mathsf{L}} = \Pi_0^{\mathsf{L}} = \mathsf{L}$
- $\Sigma_{i+1}^{\mathsf{L}} = \mathsf{NL}^{\Sigma_i^{\mathsf{L}}}$ alternatively: languages of log-space bounded Σ_{i+1} ATMs
- $\Pi_{i+1}^{\mathsf{L}} = \mathsf{coNL}^{\Sigma_i^{\mathsf{L}}}$ alternatively: languages of log-space bounded Π_{i+1} ATMs

Q1: What is the Logarithmic Hierarchy?

How do the levels of this hierarchy look?

Q1: What is the Logarithmic Hierarchy?

How do the levels of this hierarchy look?

- $\Sigma_0^{\mathsf{L}} = \Pi_0^{\mathsf{L}} = \mathsf{L}$
- $\Sigma_1^{\mathsf{L}} = \mathsf{NL}^{\mathsf{L}} =$

Q1: What is the Logarithmic Hierarchy?

How do the levels of this hierarchy look?

- $\Sigma_0^{\mathsf{L}} = \Pi_0^{\mathsf{L}} = \mathsf{L}$
- $\Sigma_1^{\mathsf{L}} = \mathsf{NL}^{\mathsf{L}} = \mathsf{NL}$

Q1: What is the Logarithmic Hierarchy?

How do the levels of this hierarchy look?

- $\Sigma_0^{\mathsf{L}} = \Pi_0^{\mathsf{L}} = \mathsf{L}$
- $\Sigma_1^{\mathsf{L}} = \mathsf{NL}^{\mathsf{L}} = \mathsf{NL}$
- $\Pi_1^{\mathsf{L}} = \mathsf{coNL}^{\mathsf{L}} =$

Q1: What is the Logarithmic Hierarchy?

How do the levels of this hierarchy look?

- $\Sigma_0^{\mathsf{L}} = \Pi_0^{\mathsf{L}} = \mathsf{L}$
- $\Sigma_1^{\mathsf{L}} = \mathsf{NL}^{\mathsf{L}} = \mathsf{NL}$
- $\Pi_1^{\mathsf{L}} = \mathsf{coNL}^{\mathsf{L}} = \mathsf{coNL}$

Q1: What is the Logarithmic Hierarchy?

How do the levels of this hierarchy look?

- $\Sigma_0^{\mathsf{L}} = \Pi_0^{\mathsf{L}} = \mathsf{L}$
- $\Sigma_1^{\mathsf{L}} = \mathsf{NL}^{\mathsf{L}} = \mathsf{NL}$
- $\Pi_1^{\mathsf{L}} = \mathsf{coNL}^{\mathsf{L}} = \mathsf{coNL} = \mathsf{NL}$ (why?)

Q1: What is the Logarithmic Hierarchy?

How do the levels of this hierarchy look?

- $\Sigma_0^{\mathsf{L}} = \Pi_0^{\mathsf{L}} = \mathsf{L}$
- $\Sigma_1^{\mathsf{L}} = \mathsf{NL}^{\mathsf{L}} = \mathsf{NL}$
- $\Pi_1^{\mathsf{L}} = \mathsf{coNL}^{\mathsf{L}} = \mathsf{coNL} = \mathsf{NL}$ (why?)
- $\Sigma_2^{\mathsf{L}} = \mathsf{NL}^{\Sigma_1^{\mathsf{L}}} = \mathsf{NL}^{\mathsf{NL}}$

Q1: What is the Logarithmic Hierarchy?

How do the levels of this hierarchy look?

- $\Sigma_0^{\mathsf{L}} = \Pi_0^{\mathsf{L}} = \mathsf{L}$
- $\Sigma_1^{\mathsf{L}} = \mathsf{NL}^{\mathsf{L}} = \mathsf{NL}$
- $\Pi_1^{\mathsf{L}} = \mathsf{coNL}^{\mathsf{L}} = \mathsf{coNL} = \mathsf{NL}$ (why?)
- $\Sigma_2^{\mathsf{L}} = \mathsf{NL}^{\Sigma_1^{\mathsf{L}}} = \mathsf{NL}^{\mathsf{NL}} = \mathsf{NL}$ (why?)

Q1: What is the Logarithmic Hierarchy?

How do the levels of this hierarchy look?

- $\Sigma_0^{\mathsf{L}} = \Pi_0^{\mathsf{L}} = \mathsf{L}$
- $\Sigma_1^{\mathsf{L}} = \mathsf{NL}^{\mathsf{L}} = \mathsf{NL}$
- $\Pi_1^{\mathsf{L}} = \mathsf{coNL}^{\mathsf{L}} = \mathsf{coNL} = \mathsf{NL}$ (why?)
- $\Sigma_2^{\mathsf{L}} = \mathsf{NL}^{\Sigma_1^{\mathsf{L}}} = \mathsf{NL}^{\mathsf{NL}} = \mathsf{NL}$ (why?)
- $\Pi_2^{\mathsf{L}} = \mathsf{coNL}^{\Sigma_1^{\mathsf{L}}} = \mathsf{coNL}^{\mathsf{NL}}$

Q1: What is the Logarithmic Hierarchy?

How do the levels of this hierarchy look?

- $\Sigma_0^{\mathsf{L}} = \Pi_0^{\mathsf{L}} = \mathsf{L}$
- $\Sigma_1^{\mathsf{L}} = \mathsf{NL}^{\mathsf{L}} = \mathsf{NL}$
- $\Pi_1^{\mathsf{L}} = \mathsf{coNL}^{\mathsf{L}} = \mathsf{coNL} = \mathsf{NL}$ (why?)
- $\Sigma_2^{\mathsf{L}} = \mathsf{NL}^{\Sigma_1^{\mathsf{L}}} = \mathsf{NL}^{\mathsf{NL}} = \mathsf{NL}$ (why?)
- $\Pi_2^{\mathsf{L}} = \mathsf{coNL}^{\Sigma_1^{\mathsf{L}}} = \mathsf{coNL}^{\mathsf{NL}} = \mathsf{NL}$ (why?)

Q1: What is the Logarithmic Hierarchy?

How do the levels of this hierarchy look?

- $\Sigma_0^{\mathsf{L}} = \Pi_0^{\mathsf{L}} = \mathsf{L}$
- $\Sigma_1^{\mathsf{L}} = \mathsf{NL}^{\mathsf{L}} = \mathsf{NL}$
- $\Pi_1^{\mathsf{L}} = \mathsf{coNL}^{\mathsf{L}} = \mathsf{coNL} = \mathsf{NL}$ (why?)
- $\Sigma_2^{\mathsf{L}} = \mathsf{NL}^{\Sigma_1^{\mathsf{L}}} = \mathsf{NL}^{\mathsf{NL}} = \mathsf{NL}$ (why?)
- $\Pi_2^{\mathsf{L}} = \mathsf{coNL}^{\Sigma_1^{\mathsf{L}}} = \mathsf{coNL}^{\mathsf{NL}} = \mathsf{NL}$ (why?)

Therefore $\Sigma_i^{\mathsf{L}} = \Pi_i^{\mathsf{L}} = \mathsf{NL}$ for all $i \geq 1$.

The Logarithmic Hierarchy collapses on the first level.

Historic note: In 1987, just before the Immerman-Szelepcsényi Theorem was published, Klaus-Jörn Lange, Birgit Jenner, and Bernd Kirsig showed that the Logarithmic Hierarchy collapses on the second level [ICALP 1987].

Question 2: The Hardest Problems in P

Q2: The hardest problems in P

What we know about P and NP:

- We don't know if any problem in NP is really harder than any problem in P.
- But we do know that NP is at least as challenging as P, i.e., $P \subseteq NP$.

Q2: The hardest problems in P

What we know about P and NP:

- We don't know if any problem in NP is really harder than any problem in P.
- But we do know that NP is at least as challenging as P, i.e., P $\subseteq$ NP.

So all problems that are hard for NP are also hard for P

Q2: The hardest problems in P

What we know about P and NP:

- We don't know if any problem in NP is really harder than any problem in P.
- But we do know that NP is at least as challenging as P, i.e., P $\subseteq$ NP.

So all problems that are hard for NP are also hard for P, aren't they?

Q2: Is NP-hard as hard as P-hard?

Let's first recall the definitions:

Definition: A problem **L** is NP-hard if, for all problems **M** $\in$ NP, there is a polynomial many-one reduction **M** $\leq_m$ **L**.

Q2: Is NP-hard as hard as P-hard?

Let's first recall the definitions:

Definition: A problem **L** is NP-hard if, for all problems **M** $\in$ NP, there is a polynomial many-one reduction **M** $\leq_m$ **L**.

Definition: A problem **L** is P-hard if, for all problems **M** $\in$ P, there is a log-space reduction **M** $\leq_L$ **L**.

Q2: Is NP-hard as hard as P-hard?

Let's first recall the definitions:

Definition: A problem **L** is NP-hard if, for all problems **M** $\in$ NP, there is a polynomial many-one reduction **M** $\leq_m$ **L**.

Definition: A problem **L** is P-hard if, for all problems **M** $\in$ P, there is a log-space reduction **M** $\leq_L$ **L**.

How to show "NP-hard implies P-hard"?

Q2: Is NP-hard as hard as P-hard?

Let's first recall the definitions:

Definition: A problem **L** is NP-hard if, for all problems **M** $\in$ NP, there is a polynomial many-one reduction **M** $\leq_m$ **L**.

Definition: A problem **L** is P-hard if, for all problems **M** $\in$ P, there is a log-space reduction **M** $\leq_L$ **L**.

How to show "NP-hard implies P-hard"?

- Assume that **L** is NP-hard.

Q2: Is NP-hard as hard as P-hard?

Let's first recall the definitions:

Definition: A problem **L** is NP-hard if, for all problems **M** $\in$ NP, there is a polynomial many-one reduction **M** $\leq_m$ **L**.

Definition: A problem **L** is P-hard if, for all problems **M** $\in$ P, there is a log-space reduction **M** $\leq_L$ **L**.

How to show "NP-hard implies P-hard"?

- Assume that **L** is NP-hard.
- Consider any language **M** $\in$ P.

Q2: Is NP-hard as hard as P-hard?

Let's first recall the definitions:

Definition: A problem **L** is NP-hard if, for all problems **M** $\in$ NP, there is a polynomial many-one reduction **M** $\leq_m$ **L**.

Definition: A problem **L** is P-hard if, for all problems **M** $\in$ P, there is a log-space reduction **M** $\leq_L$ **L**.

How to show "NP-hard implies P-hard"?

- Assume that **L** is NP-hard.
- Consider any language **M** $\in$ P.
- Then **M** $\in$ NP.

Q2: Is NP-hard as hard as P-hard?

Let's first recall the definitions:

Definition: A problem **L** is NP-hard if, for all problems **M** $\in$ NP, there is a polynomial many-one reduction **M** $\leq_m$ **L**.

Definition: A problem **L** is P-hard if, for all problems **M** $\in$ P, there is a log-space reduction **M** $\leq_L$ **L**.

How to show "NP-hard implies P-hard"?

- Assume that **L** is NP-hard.
- Consider any language **M** $\in$ P.
- Then **M** $\in$ NP.
- So there is a polynomial many-one reduction f from **M** to **L**

Q2: Is NP-hard as hard as P-hard?

Let's first recall the definitions:

Definition: A problem **L** is NP-hard if, for all problems **M** $\in$ NP, there is a polynomial many-one reduction **M** $\leq_m$ **L**.

Definition: A problem **L** is P-hard if, for all problems **M** $\in$ P, there is a log-space reduction **M** $\leq_L$ **L**.

How to show "NP-hard implies P-hard"?

- Assume that **L** is NP-hard.
- Consider any language **M** $\in$ P.
- Then **M** $\in$ NP.
- So there is a polynomial many-one reduction f from **M** to **L**
- Hence, . . .

Q2: Is NP-hard as hard as P-hard?

Let's first recall the definitions:

Definition: A problem **L** is NP-hard if, for all problems **M** $\in$ NP, there is a polynomial many-one reduction **M** $\leq_m$ **L**.

Definition: A problem **L** is P-hard if, for all problems **M** $\in$ P, there is a log-space reduction **M** $\leq_L$ **L**.

How to show "NP-hard implies P-hard"?

- Assume that **L** is NP-hard.
- Consider any language **M** $\in$ P.
- Then **M** $\in$ NP.
- So there is a polynomial many-one reduction f from **M** to **L**
- Hence, ... well...,

Q2: Is NP-hard as hard as P-hard?

Let's first recall the definitions:

Definition: A problem **L** is NP-hard if, for all problems **M** $\in$ NP, there is a polynomial many-one reduction **M** $\leq_m$ **L**.

Definition: A problem **L** is P-hard if, for all problems **M** $\in$ P, there is a log-space reduction **M** $\leq_L$ **L**.

How to show "NP-hard implies P-hard"?

- Assume that **L** is NP-hard.
- Consider any language **M** $\in$ P.
- Then **M** $\in$ NP.
- So there is a polynomial many-one reduction f from **M** to **L**
- Hence, . . . well. . . , nothing much, really.

Q2: Is NP-hard as hard as P-hard?

Let's first recall the definitions:

Definition: A problem **L** is NP-hard if, for all problems **M** $\in$ NP, there is a polynomial many-one reduction **M** $\leq_m$ **L**.

Definition: A problem **L** is P-hard if, for all problems **M** $\in$ P, there is a log-space reduction **M** $\leq_L$ **L**.

How to show "NP-hard implies P-hard"?

- Assume that **L** is NP-hard.
- Consider any language **M** $\in$ P.
- Then **M** $\in$ NP.
- So there is a polynomial many-one reduction f from **M** to **L**
- Hence, . . . well. . . , nothing much, really.

Q2: Is NP-hard as hard as P-hard?

For all we know today, it is perfectly possible that
there are NP-hard problems that are not P-hard.

Q2: Is NP-hard as hard as P-hard?

For all we know today, it is perfectly possible that there are NP-hard problems that are not P-hard.

Example 18.9: We know that $L \subseteq P \subseteq NP$ but we do not know if any of these subsumptions are proper. Suppose that the truth actually looks like this: $L \subsetneq P = NP$. Then all non-trivial problems in P are NP-hard (why?), but not every problem would be P-hard (why?).

Note: This is really about the different notions of reduction used to define hardness. If we used log-space reductions for P-hardness and NP-hardness, the claim would follow.

Question 3: Problems Harder than P

Q3: Problems harder than P

Polynomial time is an approximation of "practically tractable" problems:

- Many practical problems are in P, including many very simple ones (e.g., $\emptyset$)
- P-hard problems are as hard as any other problem in P, and
  P-complete problems therefore are the hardest problems in P
- However, there are even harder problems that are no longer in P

Q3: Problems harder than P

Polynomial time is an approximation of “practically tractable” problems:

- Many practical problems are in P, including many very simple ones (e.g., $\emptyset$)
- P-hard problems are as hard as any other problem in P, and P-complete problems therefore are the hardest problems in P
- However, there are even harder problems that are no longer in P

So, clearly, problems that are not even in P must be P-hard

Q3: Problems harder than P

Polynomial time is an approximation of “practically tractable” problems:

- Many practical problems are in P, including many very simple ones (e.g., $\emptyset$)
- P-hard problems are as hard as any other problem in P, and
  P-complete problems therefore are the hardest problems in P
- However, there are even harder problems that are no longer in P

So, clearly, problems that are not even in P must be P-hard, right?

Q3: Are problems harder than P also hard for P?

Can we find any problem that is surely harder than P?

Q3: Are problems harder than P also hard for P?

Can we find any problem that is surely harder than P? Yes, easily:

- The Halting Problem is undecidable and therefore not in P
- Any ExpTime-complete problem is not in P (Time Hierarchy Theorem); e.g., the Word Problem for exponentially time-bounded DTMs

Q3: Are problems harder than P also hard for P?

Can we find any problem that is surely harder than P? Yes, easily:

- The Halting Problem is undecidable and therefore not in P
- Any ExpTime-complete problem is not in P (Time Hierarchy Theorem); e.g., the Word Problem for exponentially time-bounded DTMs

These concrete examples both are hard for P

Q3: Are problems harder than P also hard for P?

Can we find any problem that is surely harder than P? Yes, easily:

- The Halting Problem is undecidable and therefore not in P
- Any ExpTime-complete problem is not in P (Time Hierarchy Theorem); e.g., the Word Problem for exponentially time-bounded DTMs

These concrete examples both are hard for P:

- The Word Problem for polynomially time-bounded DTMs is hard for P
- This polytime Word Problem log-space reduces to the Word Problem for exponential TMs (reduction: the identity function)
- It also log-space reduces to the Halting problem: a reduction merely has to modify the TM so that every rejecting halting configuration leads into an infinite loop

Q3: Are problems harder than P also hard for P?

Rephrasing the question: Are there problems that are not in P, yet not hard for P?

Q3: Are problems harder than P also hard for P?

Rephrasing the question: Are there problems that are not in P, yet not hard for P?

Some observations:

Q3: Are problems harder than P also hard for P?

Rephrasing the question: Are there problems that are not in P, yet not hard for P?

Some observations:

- $\emptyset$ is not P-hard (why?)

Q3: Are problems harder than P also hard for P?

Rephrasing the question: Are there problems that are not in P, yet not hard for P?

Some observations:

- $\emptyset$ is not P-hard (why?)
- Any ExpTime-complete problem **L** is not in P (why?)

Q3: Are problems harder than P also hard for P?

Rephrasing the question: Are there problems that are not in P, yet not hard for P?

Some observations:

- $\emptyset$ is not P-hard (why?)
- Any ExpTime-complete problem **L** is not in P (why?)
- We can enumerate DTMs for all languages in P (how?)

Q3: Are problems harder than P also hard for P?

Rephrasing the question: Are there problems that are not in P, yet not hard for P?

Some observations:

- $\emptyset$ is not P-hard (why?)
- Any ExpTime-complete problem **L** is not in P (why?)
- We can enumerate DTMs for all languages in P (how?)
- We can enumerate DTMs for all P-hard languages in ExpTime (how?)

Q3: Are problems harder than P also hard for P?

Rephrasing the question: Are there problems that are not in P, yet not hard for P?

Some observations:

- $\emptyset$ is not P-hard (why?)
- Any ExpTime-complete problem **L** is not in P (why?)
- We can enumerate DTMs for all languages in P (how?)
- We can enumerate DTMs for all P-hard languages in ExpTime (how?)

So, it's clear what we have to do now ...

Q3: Are problems harder than P also hard for P?

Schöning to the rescue (see Theorem 15.2):

Corollary 18.10: Consider the classes C_1 = ExpPHard (P-hard problems in ExpTime) and C_2 = P. Both are classes of decidable languages. We find that for either class C_k:

- We can effectively enumerate TMs $\mathcal{M}_0^k, \mathcal{M}_1^k, \ldots$ such that $C_k = \{\mathbf{L}(\mathcal{M}_i^k) \mid i \geq 0)\}$.
- If $\mathbf{L} \in C_k$ and $\mathbf{L'}$ differs from $\mathbf{L}$ on only a finite number of words, then $\mathbf{L'} \in C_k$

Let $\mathbf{L_1} = \emptyset$, and let $\mathbf{L_2}$ be some ExpTime-complete problem. Clearly, $\mathbf{L_1} \notin$ ExpPHard and $\mathbf{L_2} \notin$ P (Time Hierarchy), hence there is a decidable language $\mathbf{L_d} \notin$ ExpPHard $\cup$ P.
Moreover, as $\emptyset \in$ P and $\mathbf{L_2}$ is not trivial, $\mathbf{L_d} \leq_p \mathbf{L_2}$ and hence $\mathbf{L_d} \in$ ExpTime. Therefore $\mathbf{L_d} \notin$ ExpPHard implies that $\mathbf{L_d}$ is not P-hard.

This idea of using Schöning's Theorem has been put forward by Ryan Williams (link). Our version is a modification requiring $C_1 \subseteq$ ExpTime.

Q3: Are problems harder than P also hard for P?

No, there are problems in ExpTime that are neither in P nor hard for P.

(Other arguments can even show the existence of undecidable sets that are not P-hard[1])

[1] Related note: the undecidable **UHalt** is not NP-hard, since it is a so-called sparse language.

Q3: Are problems harder than P also hard for P?

No, there are problems in ExpTime that are neither in P nor hard for P.

(Other arguments can even show the existence of undecidable sets that are not P-hard[1])

Discussion:

- Considering Questions 2 and 3, the use of the word hard is misleading, since we interpret it as difficult
- However, the actual meaning difficult would be "not in a given class" (e.g., problems not in P are clearly more difficult than those in P)
- Our formal notion of hard also implies that a problem is difficult in some sense, but it also requires it to be universal in the sense that many other problems can be solved through it

What we have seen is that there are difficult problems that are not universal.

[1] Related note: the undecidable **UHalt** is not NP-hard, since it is a so-called sparse language.

Your Questions

Summary and Outlook

Nonuniform circuit families are very powerful, and even polynomial circuits can solve undecidable problems

Log-space-uniform polynomial circuits capture P.

Most boolean functions cannot be expressed by polynomial circuits, yet we don't know of any such function that is even in NExp

Answer 1: The Logarithmic Hierarchy collapses.

Answer 2: We don't know that NP-hard implies P-hard.

Answer 3: Being outside of P does not make a problem P-hard.

What's next?

- Holidays
- More on circuits
- Randomness

Here's wishing you

a Merry Christmas, a Happy Hanukkah,

a Joyous Yalda, a Cheerful Dōngzhì,

a Great Feast of Juul,

and a Wonderful Winter Solstice,

respectively!