

# Theory of Computer Science

## E6. Beyond NP

Gabriele Röger

University of Basel

May 15, 2019

# Complexity Theory: What we already have seen

- Complexity theory investigates which problems are “easy” to solve and which ones are “hard”.
- two important problem classes:
  - P: problems that are solvable in polynomial time by “normal” computation mechanisms
  - NP: problems that are solvable in polynomial time with the help of nondeterminism
- We know that  $P \subseteq NP$ , but we do not know whether  $P = NP$ .
- Many practically relevant problems are NP-complete:
  - They belong to NP.
  - All problems in NP can be polynomially reduced to them.
- If there is an efficient algorithm for one NP-complete problem, then there are efficient algorithms for all problems in NP.

coNP  
●○○○

Time and Space Complexity  
○○○○

Polynomial Hierarchy  
○○○○

Counting  
○○

End of Part E  
○○○

# coNP

# Complexity Class coNP

## Definition (coNP)

coNP is the set of all languages  $L$  for which  $\bar{L} \in \text{NP}$ .

Example: The complement of SAT is in coNP.

# Hardness and Completeness

## Definition (Hardness and Completeness)

Let  $C$  be a complexity class.

A problem  $Y$  is called  **$C$ -hard** if  $X \leq_p Y$  for **all** problems  $X \in C$ .

$Y$  is called  **$C$ -complete** if  $Y \in C$  and  $Y$  is  $C$ -hard.

## Example (TAUTOLOGY)

The following problem **TAUTOLOGY** is coNP-complete:

**Given:** a propositional logic formula  $\varphi$

**Question:** Is  $\varphi$  valid?

# Known Results and Open Questions

## Open

- $NP \stackrel{?}{=} coNP$

## Known

- $P \subseteq coNP$
- If  $X$  is NP-complete then  $\bar{L}$  is coNP-complete.
- If  $NP \neq coNP$  then  $P \neq NP$ .
- If a coNP-complete problem is in NP, then  $NP = coNP$ .
- If a coNP-complete problem is in P, then  $P = coNP = NP$ .

# Time and Space Complexity

# Time

## Definition (Reminder: Accepting a Language in Time $f$ )

Let  $M$  be a DTM or NTM with input alphabet  $\Sigma$ ,  
 $L \subseteq \Sigma^*$  a language and  $f : \mathbb{N}_0 \rightarrow \mathbb{N}_0$  a function.

$M$  accepts  $L$  in time  $f$  if:

- ① for all words  $w \in L$ :  $M$  accepts  $w$  in time  $f(|w|)$
- ② for all words  $w \notin L$ :  $M$  does not accept  $w$

- **TIME( $f$ )**: all languages accepted by a **DTM** in time  $f$ .
- **NTIME( $f$ )**: all languages accepted by a **NTM** in time  $f$ .
- $P = \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k)$
- $NP = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$

# Space

- Analogously: A TM accepts a language  $L$  in **space  $f$**  if every word  $w \in L$  gets accepted using at most of  $f(|w|)$  space besides its input on the tape and no  $w \notin L$  gets accepted.
- **SPACE( $f$ )**: all languages accepted by a **DTM** in space  $f$ .
- **NSPACE( $f$ )**: all languages accepted by a **NTM** in space  $f$ .

# Important Complexity Classes Beyond NP

- $\text{PSPACE} = \bigcup_{k \in \mathbb{N}} \text{SPACE}(n^k)$
- $\text{NPSPACE} = \bigcup_{k \in \mathbb{N}} \text{NSPACE}(n^k)$
- $\text{EXPTIME} = \bigcup_{k \in \mathbb{N}} \text{TIME}(2^{n^k})$
- $\text{EXPSPACE} = \bigcup_{k \in \mathbb{N}} \text{SPACE}(2^{n^k})$

Some known results:

- $\text{PSPACE} = \text{NPSPACE}$  (from Savitch's theorem)
- $\text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{EXPSPACE}$   
(at least one relationship strict)
- $\text{P} \neq \text{EXPTIME}$ ,  $\text{PSPACE} \neq \text{EXPSPACE}$
- $\text{P} \subseteq \text{NP} \subseteq \text{PSPACE}$

# Polynomial Hierarchy

# Oracle Machines

An **oracle machine** is like a Turing machine that has access to an **oracle** which can solve some decision problem in constant time.

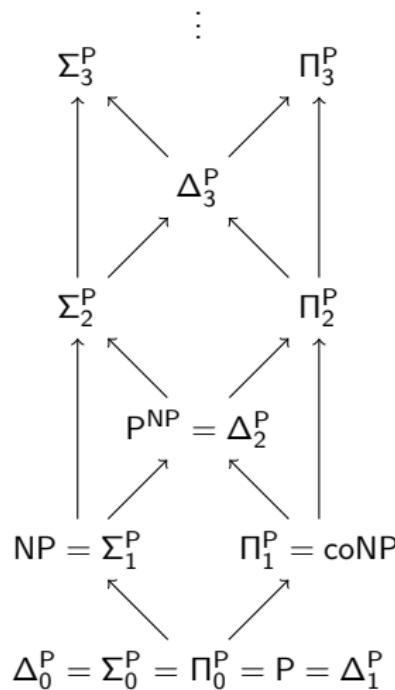
Example oracle classes:

- $P^{NP} = \{L \mid L \text{ can get accepted in polynomial time by a DTM with an oracle that decides some problem in } NP\}$
- $NP^{NP} = \{L \mid L \text{ can get accepted in pol. time by a NTM with an oracle deciding some problem in } NP\}$

# Polynomial Hierarchy

Inductively defined:

- $\Delta_0^P := \Sigma_0^P := \Pi_0^P := P$
- $\Delta_{i+1}^P := P^{\Sigma_i^P}$
- $\Sigma_{i+1}^P := NP^{\Sigma_i^P}$
- $\Pi_{i+1}^P := coNP^{\Sigma_i^P}$
- $PH := \bigcup_k \Sigma_k^P$



## Polynomial Hierarchy: Results

- $\text{PH} \subseteq \text{PSPACE}$  ( $\text{PH} \stackrel{?}{=} \text{PSPACE}$  is open)
- There are complete problems for each level.
- If there is a PH-complete problem, then the polynomial hierarchy collapses to some finite level.
- If  $P = NP$ , the polynomial hierarchy collapses to the first level.

# Counting

## #P

Complexity class **#P**

- Set of functions  $f : \{0, 1\} \rightarrow \mathbb{N}_0$ , where  $f(n)$  is the number of accepting paths of a polynomial-time NTM

## Example (#SAT)

The following problem **#SAT** is **#P**-complete:

Given: a propositional logic formula  $\varphi$

Question: How many models does  $\varphi$  have?

coNP  
oooo

Time and Space Complexity  
oooo

Polynomial Hierarchy  
oooo

Counting  
oo

End of Part E  
●oo

# End of Part E

# What's Next?

contents of this course:

- A. **background ✓**
  - ▷ mathematical foundations and proof techniques
- B. **logic ✓**
  - ▷ How can knowledge be represented?
  - ▷ How can reasoning be automated?
- C. **automata theory and formal languages ✓**
  - ▷ What is a computation?
- D. **Turing computability ✓**
  - ▷ What can be computed at all?
- E. **complexity theory**
  - ▷ What can be computed efficiently?
- F. **more computability theory**
  - ▷ Other models of computability

# What's Next?

contents of this course:

- A. **background ✓**
  - ▷ mathematical foundations and proof techniques
- B. **logic ✓**
  - ▷ How can knowledge be represented?
  - ▷ How can reasoning be automated?
- C. **automata theory and formal languages ✓**
  - ▷ What is a computation?
- D. **Turing computability ✓**
  - ▷ What can be computed at all?
- E. **complexity theory ✓**
  - ▷ What can be computed efficiently?
- F. **more computability theory**
  - ▷ Other models of computability

# Quiz



kahoot.it