Complexity Theory

Jan Křetínský

Technical University of Munich

Summer 2019

May 9, 2019

Lecture 6

coNP

Agenda

- **coNP**
- the importance of **P** vs. **NP** vs. **coNP**
- neither in **P** nor **NP**-complete: Ladner's theorem
- wrap-up Lecture 1-6

coNP

- reminder: $L \subseteq \{0,1\}^* \in$ **coNP** iff $\{0,1\}^* \setminus L \in$ **NP**
- example: $\overline{\text{SAT}}$ contains

coNP

- reminder: $L \subseteq \{0,1\}^* \in$ **coNP** iff $\{0,1\}^* \setminus L \in$ **NP**
- example: $\overline{\text{SAT}}$ contains
  - not well-formed formulas
  - unsatisfiable formulas

coNP

- reminder: $L \subseteq \{0,1\}^* \in$ **coNP** iff $\{0,1\}^* \setminus L \in$ **NP**
- example: $\overline{\text{SAT}}$ contains
  - not well-formed formulas
  - unsatisfiable formulas
- does $\overline{\text{SAT}}$ have polynomial certificates?

coNP

- reminder: $L \subseteq \{0,1\}^* \in$ **coNP** iff $\{0,1\}^* \setminus L \in$ **NP**
- example: $\overline{\text{SAT}}$ contains
  - not well-formed formulas
  - unsatisfiable formulas
- does $\overline{\text{SAT}}$ have polynomial certificates?
- not known: open problem whether **NP** is closed under complement
- note that **P** is closed under complement, compare with NFA vs DFA closure

For all certificates

- like for **NP** there is a characterization in terms of certificates
- for **coNP** it is dual: for all certificates
- $\overline{\text{3SAT}}$: to prove unsatifiability one must check all assignments, for satisfiability only one

For all certificates

- like for **NP** there is a characterization in terms of certificates
- for **coNP** it is dual: for all certificates
- $\overline{\text{3SAT}}$: to prove unsatisfiability one must check all assignments, for satisfiability only one

Theorem (coNP certificates)

A language $L \subseteq \{0,1\}^*$ is in **coNP** *iff there exists a polynomial p and a TM M such that*

$$\forall x \in \{0,1\}^* \; x \in L \Leftrightarrow \forall u \in \{0,1\}^{p(|x|)} \; M(x,u) = 1$$

Completeness

- like for **NP** one can define **coNP**-hardness and completeness
- L is **coNP**-complete iff $L \in$ **coNP** and all problems in **coNP** are polynomial-time Karp-reducible to L
- classical example: Tautology $= \{\varphi \mid$ φ is Boolean formula that is true for every assignment$\}$
- example: $x \vee \overline{x} \in$ Tautology
- proof?

Completeness

- like for **NP** one can define **coNP**-hardness and completeness
- L is **coNP**-complete iff $L \in$ **coNP** and all problems in **coNP** are polynomial-time Karp-reducible to L
- classical example: Tautology $= \{\varphi \mid$ φ is Boolean formula that is true for every assignment$\}$
- example: $x \vee \overline{x} \in$ Tautology
- proof?
  - note that L is **coNP**-complete, if $\overline{L}$ is **NP**-complete

  $\Rightarrow$ $\overline{\text{SAT}}$ is **coNP**-complete

  $\Rightarrow$ Tautology is **coNP**-complete (reduction from $\overline{\text{SAT}}$ by negating formula)

Regular Expression Equivalence

xA regular expression over $\{0, 1\}$ is defined by

$$r ::= 0 \mid 1 \mid rr \mid r|r \mid r \cap r \mid r^*$$

The language defined by r is written $\mathcal{L}(r)$.

Regular Expression Equivalence

xA regular expression over $\{0, 1\}$ is defined by

$$r ::= 0 \mid 1 \mid rr \mid r|r \mid r \cap r \mid r^*$$

The language defined by r is written $\mathcal{L}(r)$.

- let $\varphi = C_1 \wedge \ldots \wedge C_m$ be a Boolean formula in 3CNF over variables $x_1, \ldots, x_n$

Regular Expression Equivalence

xA regular expression over $\{0, 1\}$ is defined by

$$r ::= 0 \mid 1 \mid rr \mid r|r \mid r \cap r \mid r^*$$

The language defined by r is written $\mathcal{L}(r)$.

- let $\varphi = C_1 \wedge \ldots \wedge C_m$ be a Boolean formula in 3CNF over variables $x_1, \ldots, x_n$
- compute from φ a regular expression: $f(\varphi) = (\alpha_1|\alpha_2|\ldots|\alpha_m)$

Regular Expression Equivalence

xA regular expression over $\{0, 1\}$ is defined by

$$r ::= 0 \mid 1 \mid rr \mid r|r \mid r \cap r \mid r^*$$

The language defined by r is written $\mathcal{L}(r)$.

- let $\varphi = C_1 \wedge \ldots \wedge C_m$ be a Boolean formula in 3CNF over variables $x_1, \ldots, x_n$
- compute from φ a regular expression: $f(\varphi)$=$(\alpha_1|\alpha_2|\ldots|\alpha_m)$
- $\alpha_i = \gamma_{i1} \ldots \gamma_{in}$

Regular Expression Equivalence

xA regular expression over $\{0, 1\}$ is defined by

$$r ::= 0 \mid 1 \mid rr \mid r|r \mid r \cap r \mid r^*$$

The language defined by r is written $\mathcal{L}(r)$.

- let $\varphi = C_1 \wedge \ldots \wedge C_m$ be a Boolean formula in 3CNF over variables $x_1, \ldots, x_n$
- compute from φ a regular expression: $f(\varphi)$=$(\alpha_1|\alpha_2|\ldots|\alpha_m)$
- $\alpha_i = \gamma_{i1} \ldots \gamma_{in}$
- $\gamma_{ij} = \begin{cases} 0 & x_j \in C_i \\ 1 & \overline{x_j} \in C_i \\ (0|1) & \text{otherwise} \end{cases}$

Regular Expression Equivalence

xA regular expression over $\{0, 1\}$ is defined by

$$r ::= 0 \mid 1 \mid rr \mid r|r \mid r \cap r \mid r^*$$

The language defined by r is written $\mathcal{L}(r)$.

- let $\varphi = C_1 \wedge \ldots \wedge C_m$ be a Boolean formula in 3CNF over variables $x_1, \ldots, x_n$
- compute from φ a regular expression: $f(\varphi)$=$(\alpha_1|\alpha_2|\ldots|\alpha_m)$
- $\alpha_i = \gamma_{i1} \ldots \gamma_{in}$
- $\gamma_{ij} = \begin{cases} 0 & x_j \in C_i \\ 1 & \overline{x_j} \in C_i \\ (0|1) & \text{otherwise} \end{cases}$
- example: $(x \vee y \vee \overline{z}) \wedge (\overline{y} \vee z \vee w)$ transformed to $(001(0|1)) \mid (0|1)100)$

Regular Expression Equivalence

xA regular expression over $\{0,1\}$ is defined by

$$r ::= 0 \mid 1 \mid rr \mid r|r \mid r \cap r \mid r^*$$

The language defined by r is written $\mathcal{L}(r)$.

- let $\varphi = C_1 \wedge \ldots \wedge C_m$ be a Boolean formula in 3CNF over variables $x_1, \ldots, x_n$
- compute from φ a regular expression: $f(\varphi)$=$(\alpha_1|\alpha_2|\ldots|\alpha_m)$
- $\alpha_i = \gamma_{i1} \ldots \gamma_{in}$
- $\gamma_{ij} = \begin{cases} 0 & x_j \in C_i \\ 1 & \overline{x_j} \in C_i \\ (0|1) & \text{otherwise} \end{cases}$
- example: $(x \vee y \vee \overline{z}) \wedge (\overline{y} \vee z \vee w)$ transformed to $(001(0|1)) \mid (0|1)100)$
- observe: φ is unsatisfiable iff $f(\varphi) = \{0,1\}^n$

Regular expressions and computational complexity

- previous slide establishes: $\overline{\text{3SAT}} \leq_p \text{RegExpEq}_0$
- that is: regular expression equivalence is **coNP**-hard

Regular expressions and computational complexity

- previous slide establishes: $\overline{\text{3SAT}} \leq_p \text{RegExpEq}_0$
- that is: regular expression equivalence is **coNP**-hard
- it is **coNP**-complete for expressions without $*, \cap$
- because one needs to check for all expressions of length *n* whether they are included (test polynomial by NFA transformation)

Regular expressions and computational complexity

- previous slide establishes: $\overline{\text{3SAT}} \leq_p \text{RegExpEq}_0$
- that is: regular expression equivalence is **coNP**-hard
- it is **coNP**-complete for expressions without $*, \cap$
- because one needs to check for all expressions of length n whether they are included (test polynomial by NFA transformation)
- the problem becomes **PSPACE**-complete when $*$ is added
- the problem becomes **EXP**-complete when $*, \cap$ is added

Agenda

- **coNP** ✓
- the importance of **P** vs. **NP** vs. **coNP**
- neither in **P** nor **NP**-complete: Ladner's theorem
- wrap-up Lecture 1-6

Open and known problems

OPEN

- **P** $=$ **NP**?
- **NP** $=$ **coNP**?

Open and known problems

OPEN

- **P** $=$ **NP**?
- **NP** $=$ **coNP**?

KNOWN

- if an **NP**-complete problem is in **P**, then **P** $=$ **NP**
- **P** $\subseteq$ **coNP** $\cap$ **NP**
- if $L \in$ **coNP** and L **NP**-complete then **NP** $=$ **coNP**
- if **P** $=$ **NP** then **P** $=$ **NP** $=$ **coNP**
- if **NP** $\neq$ **coNP** then **P** $\neq$ **NP**
- if **EXP** $\neq$ **NEXP** then **P** $\neq$ **NP** (equalities scale up, inequalities scale down – by padding)

What if $P = NP$?

- one of the most important open problems
- computational utopia
- SAT has polynomial algorithm
- 1000s of other problems, too (due to reductions, completeness)
- finding solutions is as easy as verifying them
- guessing can be done deterministically
- decryption as easy as encryption
- randomization can be de-randomized

What if NP $=$ coNP

Problems have short certificates that don't seem to have any!

- like tautology, unsatisfiability
- like unsatisfiable ILPs
- like regular expression equivalence

How to cope with NP-complete problems?

- ignore (see SAT), it may still work
- modify your problem (2SAT, 2Coloring)
- NP-completeness talks about worst cases and exact solutions
  - → try average cases
  - → try approximations
- randomize
- explore special cases (TSP)

In praise of reductions

- reductions help, when lower bounds are hard to come by
- reductions helped to prove **NP**-completeness for 1000s of natural problems
- in fact, most natural problems (exceptions are Factoring and Iso) are either in **P** or **NP**-complete
- but, unless **P** $=$ **NP**, there exist such problems

Agenda

- **coNP** ✓
- the importance of **P** vs. **NP** vs. **coNP** ✓
- neither in **P** nor **NP**-complete: Ladner's theorem
- wrap-up Lecture 1-6

Ladner's Theorem

P/**NP** intermediate languages exist!

Theorem (Ladner)

If **P** $\neq$ **NP** *then there exists a language* $L \subseteq$ **NP** $\setminus$ **P** *that is not* **NP***-complete.*

Proof

- let $H : \mathbb{N} \rightarrow \mathbb{N}$ be a function
- define $\mathrm{SAT_H}$ to be

$$\{\varphi 0 1^{n^{H(n)}} \mid \varphi \in \mathrm{SAT}, n = |\varphi|\}$$

Proof

- let $H : \mathbb{N} \to \mathbb{N}$ be a function
- define $\mathrm{SAT_H}$ to be

$$\{\varphi 01^{n^{H(n)}} \mid \varphi \in \mathrm{SAT}, n = |\varphi|\}$$

Using the definition of SAT_H one can show

1. $H(n) \in \mathcal{O}(1) \Rightarrow SAT_H \notin$ **P**
2. $\lim_{n\to\infty} H(n) = \infty \Rightarrow SAT_H$ is not **NP**-complete

Proof

- let $H : \mathbb{N} \to \mathbb{N}$ be a function
- define $\mathrm{SAT_H}$ to be

$$\{\varphi 0 1^{n^{H(n)}} \mid \varphi \in \mathrm{SAT}, n = |\varphi|\}$$

Using the definition of SAT_H one can show

1. $H(n) \in \mathcal{O}(1) \Rightarrow SAT_H \notin$ **P**
2. $\lim_{n\to\infty} H(n) = \infty \Rightarrow SAT_H$ is not **NP**-complete

For $H(n)$ at most a constant, padding is polynomial and the SAT_H is **NP**-complete, hence not in **P**.

Proof

- let $H : \mathbb{N} \to \mathbb{N}$ be a function
- define SAT_H to be

$$\{\varphi 0 1^{n^{H(n)}} \mid \varphi \in \mathrm{SAT}, n = |\varphi|\}$$

Using the definition of SAT_H one can show

1. $H(n) \in \mathcal{O}(1) \Rightarrow SAT_H \notin \mathbf{P}$
2. $\lim_{n\to\infty} H(n) = \infty \Rightarrow SAT_H$ is not **NP**-complete

For $H(n)$ at most a constant, padding is polynomial and the SAT_H is **NP**-complete, hence not in **P**.

If SAT_H is **NP**-complete, then there is a reduction from SAT to SAT_H in time $\mathcal{O}(n^i)$ for some constant. For large n it maps SAT instances φ to SAT_H instances $\psi 0 1^{|\psi|^{H(|\psi|)}}$ of size $|\psi| + |\psi|^{H(|\psi|)} = \mathcal{O}(|\varphi|^i)$. This implies $|\psi| \in o(|\varphi|)$ and by repeated application SAT $\in$ **P**. Contradiction!

Proof

Combine the approaches:

- define the function H and fix SAT_H

Proof

Combine the approaches:

- define the function H and fix SAT_H
- $H(n)$ is
  - the smallest $i < \log \log n$ such that

Proof

Combine the approaches:

- define the function H and fix SAT_H
- $H(n)$ is
  - the smallest $i < \log\log n$ such that
    $\forall x \in \{0,1\}^*$ with $|x| \leq \log n$
    M_i (the i-th TM) outputs $SAT_H(x)$
    within $i|x|^i$ steps

Proof

Combine the approaches:

- define the function H and fix SAT_H
- $H(n)$ is
  - the smallest $i < \log\log n$ such that
    $\forall x \in \{0,1\}^*$ with $|x| \leq \log n$
    M_i (the i-th TM) outputs $SAT_H(x)$
    within $i|x|^i$ steps
  - if no such i exists then $H(n) = \log\log n$
- if $SAT_H(x) \in \mathbf{P}$, say computed in kn^k
  then there is $j > k$ such that M_i computes $SAT_H(x)$
  hence for $n > 2^{2^j}$ we have $H(n) \leq j$

Proof

Combine the approaches:

- define the function H and fix SAT_H
- $H(n)$ is
  - the smallest $i < \log\log n$ such that
    $\forall x \in \{0,1\}^*$ with $|x| \leq \log n$
    M_i (the i-th TM) outputs $SAT_H(x)$
    within $i|x|^i$ steps
  - if no such i exists then $H(n) = \log\log n$
- if $SAT_H(x) \in \mathbf{P}$, say computed in kn^k
  then there is $j > k$ such that M_i computes $SAT_H(x)$
  hence for $n > 2^{2^j}$ we have $H(n) \leq j$
- H tends to ∞ since $SAT_H(x)$ cannot be computed in $\mathbf{P}$ and each M_i must be wrong on a long enough input

Agenda

- **coNP** ✓
- the importance of **P** vs. **NP** vs. **coNP** ✓
- neither in **P** nor **NP**-complete: Ladner’s theorem ✓
- wrap-up Lecture 1-6

What you should know by now

- deterministic TMs capture the inuitive notion of algorithms and computability
- universal TM $\sim$ general-purpose computer or an interpreter
- some problems are not computable aka. undecidable, like the halting problem
- this is proved by diagonalization
- complexity class **P** captures tractable problems
- **P** is robust under TM definition tweaks (tapes, alphabet size, obliviousness, universal simulation)
- non-deterministic TMs can be simulated by TM in exponential time
- **NP** $\sim$ non-det. poly. time $\sim$ polynomially checkable certificates

What you should know by now

- **NP** $\sim$ non-det. poly. time $\sim$ polynomially checkable certificates
- reductions allow to define hardness and completeness of problems
- complete problems are the hardest within a class, if they can be solved efficiently the whole class can
- **NP** complete problems: 3SAT (by Cook-Levin); Indset, 3−Coloring, ILP (by reduction from 3SAT)
- SAT is practically useful and feasible
- **coNP** complete problems: Tautology, star-free regular expression equivalence
- probably there are problems neither in **P** nor **NP**-complete (Ladner)

What's next?

- space classes
- space and time hierarchy theorems
- generalization of **NP** and **coNP**: polynomial hierarchy
- probabilistic TMs, randomization
- complexity and proofs