Automata and Grammars (BIE-AAG)

10. Type 1 and 0 languages. Turing machine.

Jan Holub

Department of Theoretical Computer Science
Faculty of Information Technology
Czech Technical University in Prague

Turing machine

Definition

Deterministic Turing machine is a 7-tuple $R = (Q, \Sigma, G, \delta, q_0, B, F)$, where:

- Q is a finite set of states,
- Σ is a finite input alphabet,
- G is a finite work alphabet $(\Sigma \subset G)$,

Moves left, right or stays in the same position of the tape

- δ is a mapping from $(Q \setminus F) \times G$ into $Q \times G \times \{-1, 0, 1\}$,
- $q_0 \in Q$ is the initial state,
- B is a blank symbol $(B \in G \setminus \Sigma)$,

Only for non final states

- $F \subseteq Q$ is a set of final states.

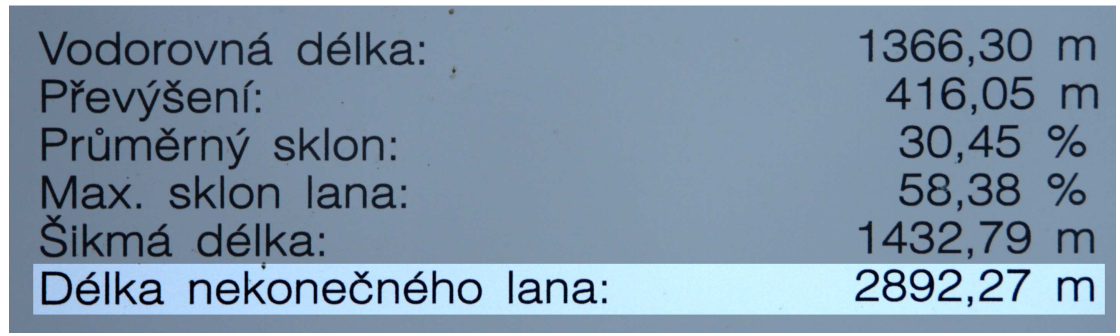
TM can write on the input tape and can move on it freely.

http://aturingmachine.com

Turing machine

Infinite tape?

No problem.



Turing machine

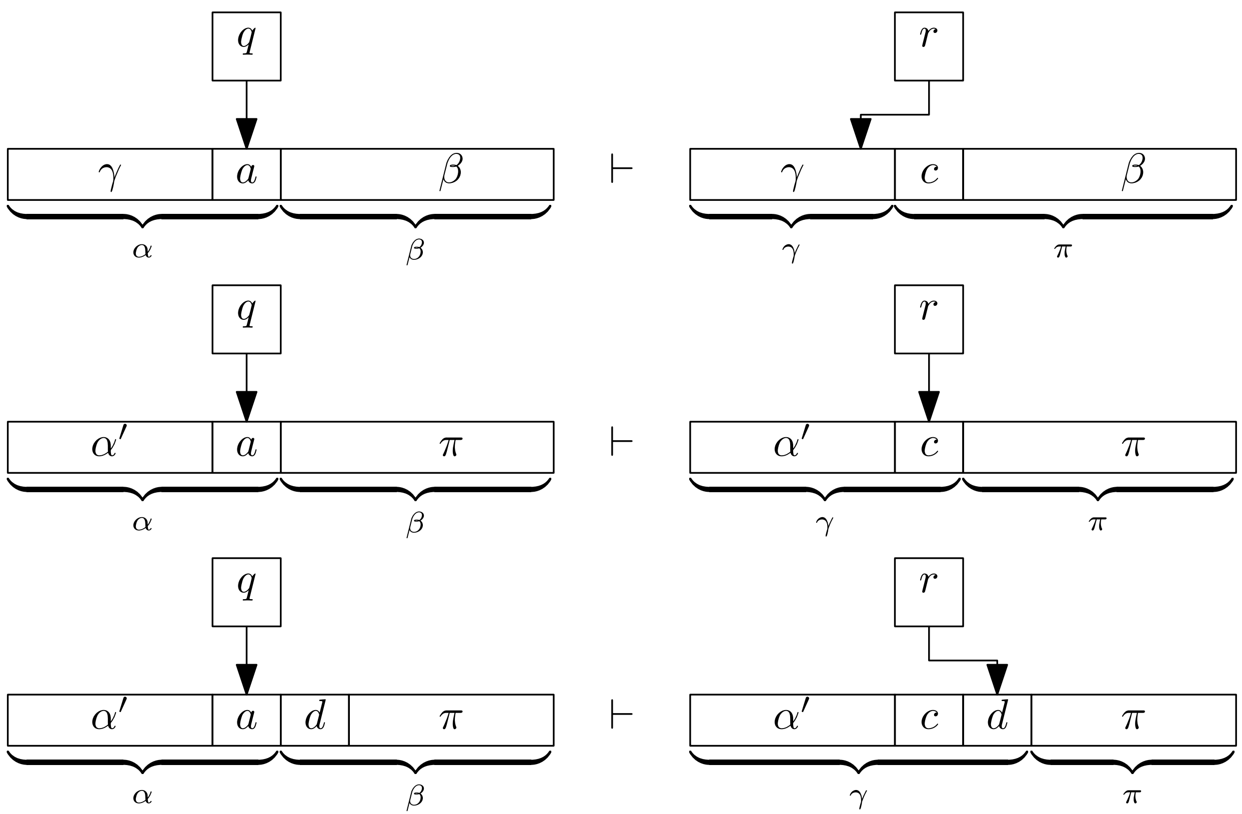
Configuration of TM R: $(\alpha, q, \beta) \in G^* \times Q \times G^*$, where

- q is the machine's current state,
- head R reads position $|\alpha|$ on the input tape,
- i-th letter of string $\alpha\beta$ is on the input tape when $i \leq |\alpha\beta|$, or B is on the input tape if $i > |\alpha\beta|$.

Turing machine

TM makes transition $(\alpha, q, \beta) \vdash (\gamma, r, \pi)$ if:

- $\alpha = \gamma a, c\beta = \pi, \delta(q, a) = (r, c, -1),\ a, c \in G$,
- $\alpha = \alpha' a, \gamma = \alpha' c, \beta = \pi, \delta(q, a) = (r, c, 0),\ a, c \in G$,
- $\alpha = \alpha' a, \gamma = \alpha' cd, \beta = d\pi, \delta(q, a) = (r, c, 1),\ a, c, d \in G$.



Turing machine

Definition

Turing machine $R = (Q, \Sigma, G, \delta, q_0, B, F)$ *accepts a word* $a\alpha \in \Sigma^+$ if $\exists q \in F$, $(a, q_0, \alpha) \vdash^* (B, q, \varepsilon)$.
TM R *accepts* ε if $\exists q \in F$, $(B, q_0, \varepsilon) \vdash^* (B, q, \varepsilon)$.
$L(R)$ is a language of words accepted by TM R.

Definition

Language L is *recursively enumerable* if it is accepted by some TM R $(L = L(R))$.

Theorem

Every language accepted by k-tape TM, $k \geq 1$, is recursively enumerable.

Nondeterministic Turing machine

Definition

Nondeterministic TM is a seven-tuple $R = (Q, \Sigma, G, \delta, q_0, B, F)$, where:

- Q is a finite set of states,
- Σ is a finite input alphabet,
- G is a finite work alphabet ($\Sigma \subset G$),
- δ is a mapping from $(Q \setminus F) \times G$ into $\mathcal{P}(Q \times G \times \{-1, 0, 1\})$,
- $q_0 \in Q$ is the initial state,
- B is the blank symbol ($B \in G \setminus \Sigma$),
- $F \subseteq Q$ is a set of final states.

NTM accepts a word $a\alpha$ if there exists $q \in F$ so that $(a, q_0, \alpha) \vdash^* (B, q, \varepsilon)$.
NTM accepts word ε if there exists $q \in F$ so that $(B, q_0, \varepsilon) \vdash^* (B, q, \varepsilon)$.

Nondeterministic Turing machine

Theorem
If M_N is a nondeterministic TM, then there is a deterministic TM M_D such that $L(M_N) = L(M_D)$.

Corollary
Nondeterministic Turing machines accept exactly recursively enumerable languages.

Linear bounded automaton (LBA)

Linear bounded automaton $=$ Linear bounded Turing machine

Definition

TM is a *linear bounded automaton* if the length of its tape is restricted to a k-multiple of length of the input word for some fixed $k \geq 1$.

Theorem

For every noncontracting grammar G there exists an equivalent context-sensitive grammar.

Theorem

For any grammar G there exists a TM R such that $L(G) = L(R)$. For any noncontracting grammar G there exists a LBA R such that $L(G) = L(R)$.

Linear bounded automaton (LBA)

Corollary

Grammars generate exactly recursively enumerable languages.
Context-sensitive languages are accepted exactly by LBAs.

Theorem

Recursively enumerable languages are closed under operations of union, concatenation, and Kleene star.

Theorem

Context-sensitive languages are closed under operations of union, concatenation, Kleene star, and complement.

Algorithm

Definition

Turing machine R *decides* language L over alphabet Σ if its computation halts for every word and $L(R) = L$.
Language L is recursive if there exists a TM that decides it.

Theorem

L is recursive if and only if L and $\overline{L}$ are recursively enumerable.

Theorem

Every context-sensitive language is recursive.

Church-Turing thesis

Every language that can be described in some manner by a finite expression is recursively enumerable.
There is an equivalent Turing machine to every algorithm.

Universal Turing machine

Definition
Turing machine is *universal* if and only if it accepts all pairs $(\textit{codeof}(R); \alpha)$ such that TM R accepts word α.

Undecidable problems

Halting Problem for TM:
Given a Turing machine T and an input w, does T halt on w?
It is a language of pairs (R, w), where R is a TM and $w \in \Sigma^*$ such that R halts having w as input.

Theorem
The Halting Problem is not recursive.

Proof
By contradiction. Suppose function *Halts*():

$$\textit{Halts}(P; \alpha) = \begin{cases} \textit{yes}, & \text{if } P \text{ halts for input } \alpha, \\ \textit{no}, & \text{if } P \text{ does not halt for input } \alpha. \end{cases}$$

Construct program P for UTM:
P: L: if *Halts*$(P; P)$ then goto L else halt $\square$

Undecidable problems

Post Correspondence Problem:
Given two sequences $U = (u_1, u_2, \ldots, u_m)$ and $V = (v_1, v_2, \ldots, v_m)$ of strings $u_i, v_i \in \Sigma^*, |\Sigma| \geq 2.$ Find whether there is a finite sequence $(i_1, i_2, \ldots, i_p)$, $i_j \in \{1, \ldots, m\}$ so that $u_{i_1} u_{i_2} \ldots u_{i_p} = v_{i_1} v_{i_2} \ldots v_{i_p}$.

Example
$U = (abab, aaabbb, aab, ba, ab, aa)$
$V = (ababaaa, bb, baab, baa, ba, a)$

There is a solution for this instance of the problem:
1234556
$abab\ aaabbb\ aab\ ba\ ab\ ab\ aa = ababaaa\ bb\ baab\ baa\ ba\ ba\ a$.

Classes P and NP

Problems

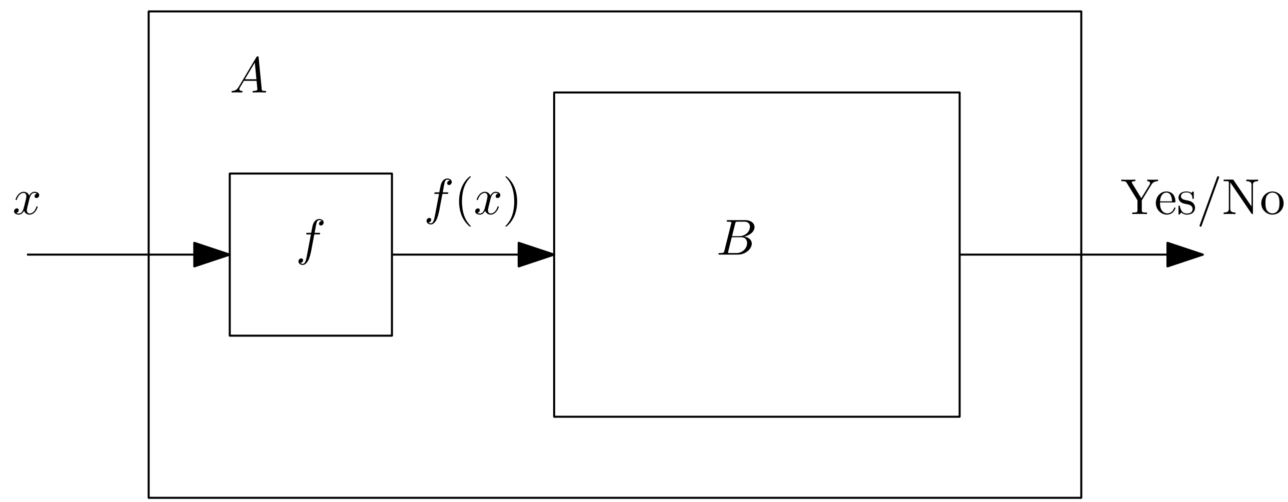
- decision (yes/no)
- optimization (the best solution)

Definition
Class NP (**n**on-deterministic **p**olynomial-time) is a class of problems that can be solved in polynomial time on a non-deterministic Turing machine.

Definition
Class P (**p**olynomial-time) is the class of problems that can be solved in polynomial time using a deterministic Turing machine.

Polynomial-time reduction

Definition (Polynomial-time reduction)
We say that a language $A \subseteq \{0,1\}^*$ is polynomial-time (Karp) reducible to a language $B \subseteq \{0,1\}^*$ denoted by $A \leq_p B$ if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$, $x \in A$ if and only if $f(x) \in B$.



Polynomial-time reduction

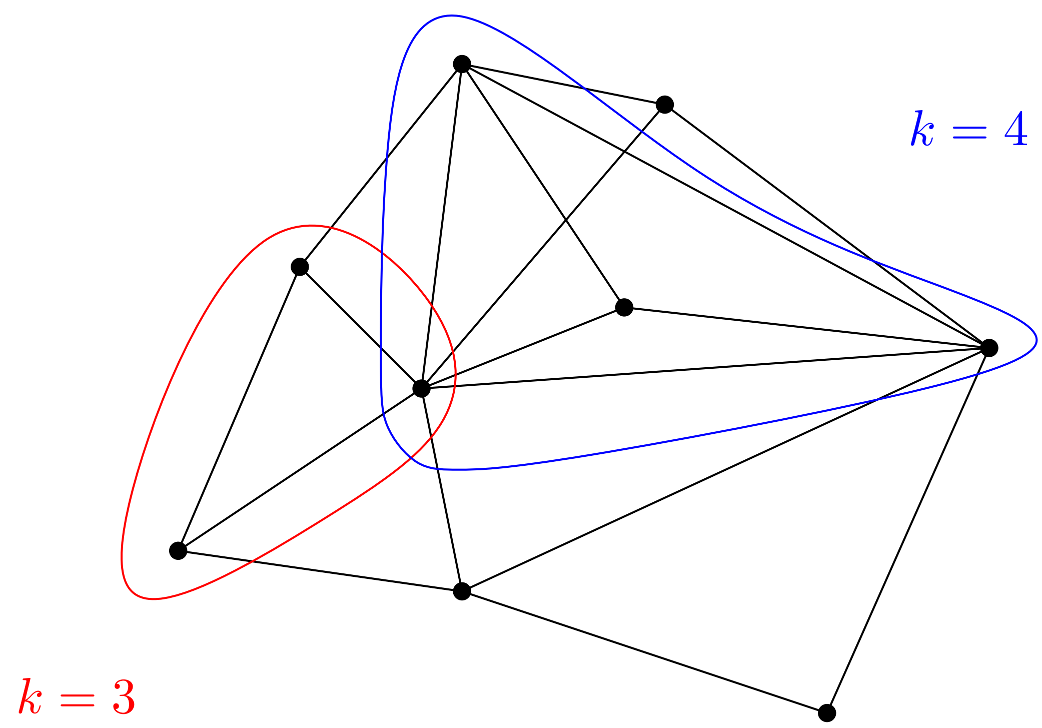
Example (CNF-SAT $\leq_p$ Clique)
Problem CNF-SAT:
Given Boolean expression φ in a conjunctive normal form (CNF). Does there exist a satisfiable assignment?

$$(x \vee y \vee z) \wedge (\neg x \vee z \vee w) \wedge (\neg x \vee \neg w) \wedge (\neg w \vee x)$$

assignment: x, y any value, $z = \mathrm{true}$, $w = \mathrm{false}$

Polynomial-time reduction

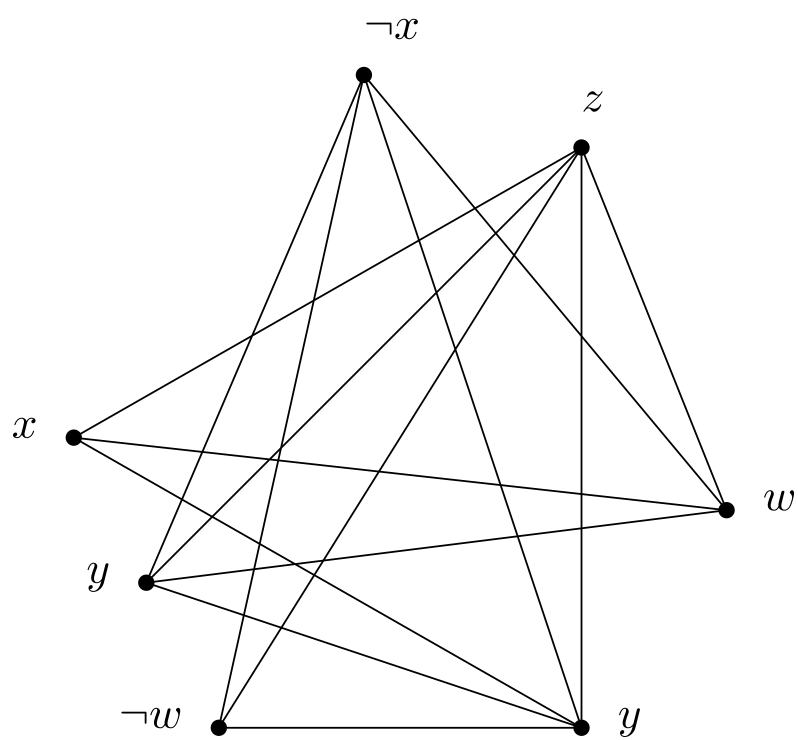
Example (CNF-SAT $\leq_p$ Clique (cont.))
Problem Clique:
Given is a graph $G = (V, E)$ and number k. Does there exist a clique of size k, i.e. a subset of vertices S of size k such that for every $u, v \in S$, $(u, v) \in E$?



Polynomial-time reduction

Example (CNF-SAT $\leq_p$ Clique (cont.))
Reduction:

$V =$ a set of all literals of expression φ. We connect by edges all literals of different clauses that are not negations each other.
$(x \vee y \vee \neg w) \wedge (\neg x \vee z) \wedge (y \vee w)$

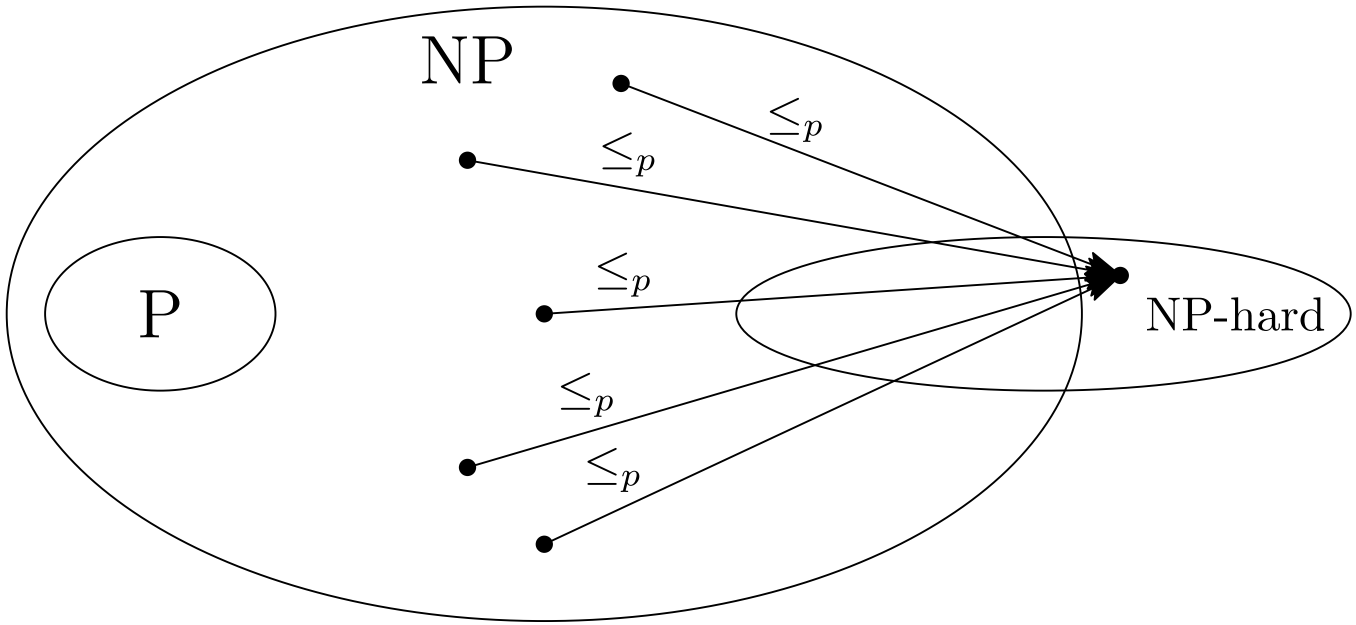


Expression φ is satisfiable. $\Leftrightarrow$ There exists a cligue of size $k =$ number of clauses.

NP-hard problem

Definition (NP-hard)
We say that B is NP-hard if $A \leq_p B$ for every $A \in \mathrm{NP}$.

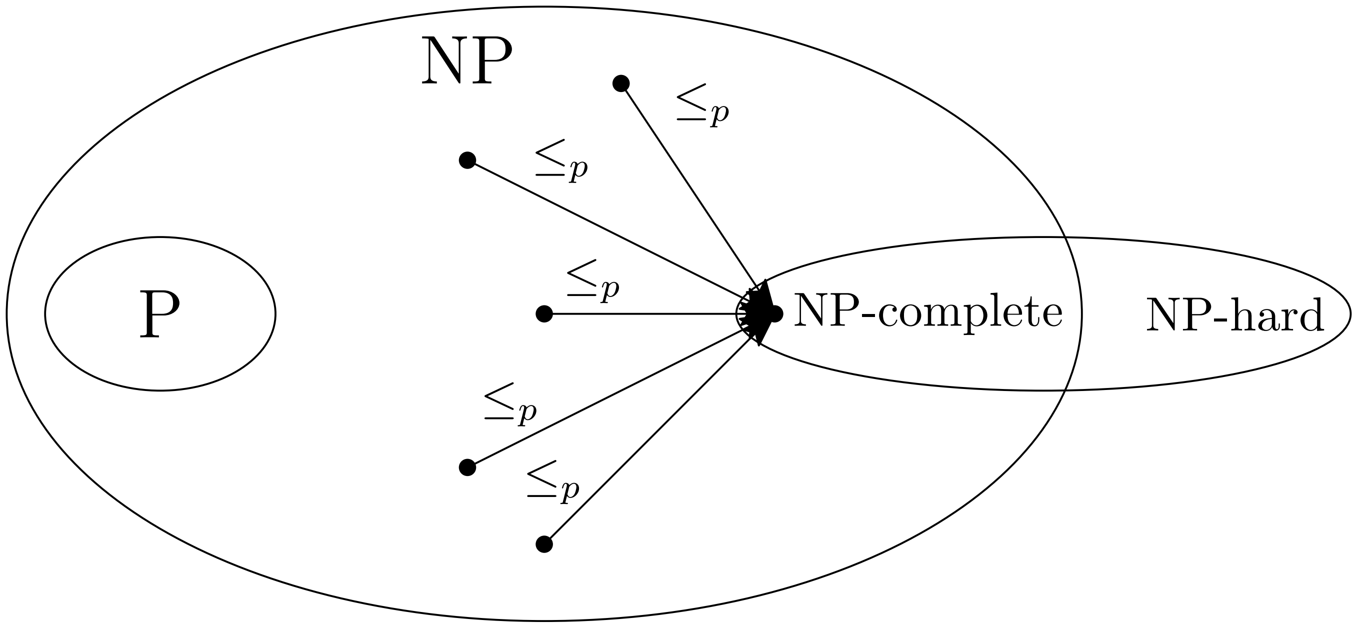
NP-hard problems are such problems that any problem in NP can be polynomial-time reduced to them. (They are at least as hard as the hardest problems in NP.)



NP-complete problem

Definition (NP-complete)
We say that B is NP-complete if B is NP-hard and $B \in \mathrm{NP}$.

NP-complete (NPC) problems are all non-deterministic polynomial-time problems such that every problem in NP can be polynomial-time reduced to them in polynomial time. (NP-complete = the "hardest" problems in NP)



NP-complete problems

- **Cook' Theorem**
  The CNF-SAT language is NP-complete.

- Graph coloring, Clique, Tiling, Subset sum, Knapsack problem

- . . .