The Complexity Class NP and NP-Completeness

valentinpi

4. Dezember 2020
(neueste Version)

Proseminar Theoretische Informatik WiSe 2020-21
Institut für Informatik
Freie Universität Berlin

Nondeterminism

Remark
The *nondeterministic* definition of the Turing machine mainly differs in the state transitioning function $\delta\colon Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{ L, N, R \})$.

Nondeterminism

Remark
The *nondeterministic* definition of the Turing machine mainly differs in the state transitioning function $\delta\colon Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{ L, N, R \})$.

Definition
In analogy to the class TIME, the class *NTIME* of $f : \mathbb{N} \to \mathbb{R}^+$ is defined as all languages computable by a nondeterministic Turing machine in $\mathcal{O}(f(n))$ time:

$$\begin{aligned}\mathrm{NTIME}(f(n)) := \{ L \mid & \text{There is a nondeterministic TM that decides } L \text{ in} \\ & \mathcal{O}(f(n)) \text{ time for an input of size } n \}\end{aligned}$$

Verifiers

Definition
An algorithm V is called a *verifier* for a language L, if:

$$L = \{ \, w \mid \exists \, c \in \Sigma^* \colon V \text{ accepts } \langle w, c \rangle \, \}$$

V is called a *polynomial verifier*, if it runs in polynomial time in terms of $|w|$. L is called *polynomially verifiable*, if such a polynomial verifier exists. c is called *certificate*.

Definition
An algorithm V is called a *verifier* for a language L, if:

$$L = \{ \ w \mid \exists \ c \in \Sigma^* \colon V \text{ accepts } \langle w, c \rangle \ \}$$

V is called a *polynomial verifier*, if it runs in polynomial time in terms of $|w|$. L is called *polynomially verifiable*, if such a polynomial verifier exists. c is called *certificate*.

Example: Path for the PATH problem

Machines and Verifiers

Machines

Verifiers

Machines and Verifiers

$$\begin{tikzcd}
\text{Machines} \arrow[r, bend left, "\text{Guess certificate}"] & \text{Verifiers}
\end{tikzcd}$$

Machines and Verifiers

$$
\begin{tikzcd}
\text{Machines} \arrow[rr, bend left, "\text{Guess certificate}"] & & \text{Verifiers} \arrow[ll, bend left, "\text{Use certificate as step description}"]
\end{tikzcd}
$$

Definition
The class NP is the class of languages computable by nondeterministic TMs in polynomial time.

$$\mathrm{NP} := \bigcup_{k \in \mathbb{N}} \mathrm{NTIME}((n^k))$$

NP and Equivalence

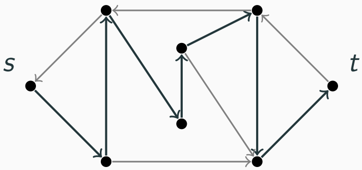
Definition
The class NP is the class of languages computable by nondeterministic TMs in polynomial time.

$$\mathrm{NP} := \bigcup_{k \in \mathbb{N}} \mathrm{NTIME}((n^k))$$

Fact
$\mathrm{NP} = \{ L \mid L \text{ has a polynomial time verifier} \}$

The HAMPATH Problem

$$\text{HAMPATH} := \{ \langle G, s, t \rangle \mid \text{There is a hamiltonian path between } s \text{ and } t \}$$



HAMPATH $\in$ NP

Theorem
HAMPATH $\in$ NP

HAMPATH $\in$ NP

Theorem
HAMPATH $\in$ NP

Beweis.

Input: $\langle G, s, t \rangle$, where $G = (V, E)$ is a directed graph with nodes s, t.

Function:

1: Index the nodes from 1 to $m := |V|$ (number of nodes). Nondeterministically write a list of m numbers $p_1, ..., p_m$, where $p_i \in \{1, ..., m\}$.
2: Check for repetitions. If any are found, reject.
3: Check if p_1, p_m correspond to the node indices of s, t. If not, reject.
4: **for** each $1 \le i \le m - 1$ **do**
5: Check if $(p_i, p_{i+1}) \in E$. If not, reject
6: Accept.

∎

The SUBSET-SUM Problem

$$\begin{aligned}\text{SUBSET-SUM} \coloneqq \{\ \langle S, t\rangle \mid S = \{x_1, ..., x_n\} \wedge \\ \exists\ \{y_1, ..., y_k\} \subseteq S\colon\ \sum y_i = t\ \}\end{aligned}$$

SUBSET-SUM $\in$ NP

Theorem
SUBSET-SUM $\in$ NP

SUBSET-SUM $\in$ NP

Theorem
SUBSET-SUM $\in$ NP

Beweis.
Construct a polynomial time verifier.

SUBSET-SUM $\in$ NP

Theorem
SUBSET-SUM $\in$ NP

Beweis.
Construct a polynomial time verifier.

Input: $\langle\langle S, t\rangle, C\rangle$ with S, C finite set of numbers and c number.

Function:

1: Test if $\sum c_i = t$ with $C = \{c_1, ..., c_k\}$.
2: Test if $C \subseteq S$.
3: Accept if both pass, otherwise reject at one stage.

■

Additional NP Problems

Additional NP Problems

$$\text{COMPOSITES} = \{ \langle n \rangle \mid n = pq \text{ with } p, q > 1 \text{ natural numbers} \}$$

Additional NP Problems

$$\text{COMPOSITES} = \{\ \langle n \rangle \mid n = pq \text{ with } p, q > 1 \text{ natural numbers } \}$$

$$\text{CLIQUE} = \{\ \langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique } \}$$

Definition
A language A is *polynomial time reducible* to a language B, written $A \leq_p B$, if there is a polynomial time computable function $f : \Sigma^* \to \Sigma^*$, where for every $w \in \Sigma^*$:

$$w \in A \leftrightarrow f(w) \in B$$

SAT, 3SAT

SAT, 3SAT

$$\text{SAT} = \{\ \langle\phi\rangle \mid \phi \text{ has a satisfying assignment }\}$$

SAT, 3SAT

$$\text{SAT} = \{\ \langle\phi\rangle \mid \phi \text{ has a satisfying assignment }\}$$

$$\text{3SAT} = \{\ \langle\phi\rangle \mid \phi \text{ is a satisfiable 3cnf-formula }\}$$

3SAT $\leq_p$ CLIQUE

Theorem
3SAT $\leq_p$ CLIQUE

Theorem
3SAT $\leq_p$ CLIQUE

Beweis.
Present reduction as function f: Generate an undirected graph G out from the input 3cnf-formula ϕ given. Let ϕ have k clausels.

- Organize clausels in triples $t_1, ..., t_k$
- Connect nodes if not from the same triple
- Or if not contradicting (e.g. x and $\neg x$ are not connected)

■

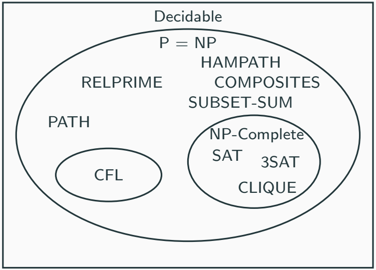
NP-Completeness

Definition
A language L is called *NP-complete*, if it satisfies two conditions:

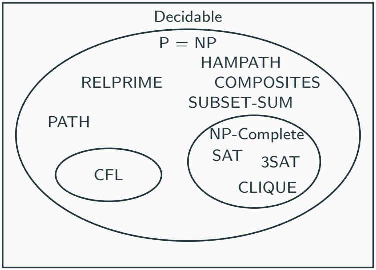
1. $L \in \mathrm{NP}$
2. $L \leq_p X$ for every language X, also called *NP-hard*

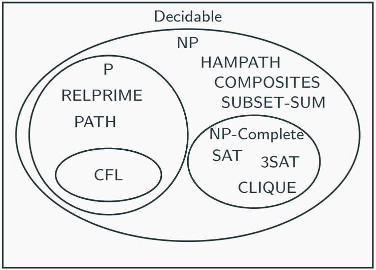
The New Landscape

The New Landscape



The New Landscape





Resources

Michael Sipser.
Introduction to the Theory of Computation, Third Edition.
Cengage Learning, 2012.