Formal definition of P

A *formal language* L is a set of strings.

Example:

{"abc", "qwerty", "xyzzy"}
{binary strings of odd lenght}
{binary strings that represents prime numbers }
{syntactically correct C-programs}

A language can be describe in different ways:

- An enumeration of the strings in the language.
- A set of rules defining the language.
- An algorithm which recognize the strings in the language.

To every decision problem there is a corresponding language:
The language of all yes-instances.

We say that the algorithm A decides L if

$$A(x) = \text{Yes if } x \in L,$$
$$A(x) = \text{No if } x \notin L.$$

A runs in *polynomial time* if $A(x)$ runs in time $O(|x|^k)$ for all x and some integer k.

$P = \{L : \exists A \text{ that decides } L \text{ i polynomial time}\}$

A formal definition of NP

A *verifies* the instance x of the problem L if there is a certificate y such that $|y| \in O(|x|^s)$ and

$$A(x,y) = \text{Yes} \quad \Leftrightarrow \quad x \in L$$

This means that A decides the language
$$L = \{x \in \{0,1\}^* : \exists y \in \{0,1\}^* : A(x,y) = \text{Ja}\}$$

$NP = \{L : \exists A$ that verifies L in polynomial time$\}$

$P \subseteq$ since all problem that can be decided in polynomial time also can be verified in polynomial time.

A second definition of NP:

A non-deterministic algorithm is an algorithm that makes random choices. The output is stochastic. We say that A decides a language L if:

$$x \in L \Rightarrow A(x) = \text{Yes} \quad \text{with probabilty} > 0$$
$$x \notin L \Rightarrow A(x) = \text{No} \quad \text{with probability } 1$$

$NP = \{L$: $\exists$polynomial time non-deterministic algorithm that decides $L\}$

Proving NP-Completeness

In order to show that A is NP-Complete it is enough to show that $A \in NP$ and $SAT \leq_P A$. **Why:** If $X \in$ we know that $X \leq SAT$. If we also have $SAT \leq A$ we know that $X \leq A$! This shows that A is NP-Complete.

Another approach: We can form i directed graph such that $A \to B$ means $A \leq B$. $SAT \to A \to B \to C \to ...$ tells us that $A, B, C, ...$ are NP-Complete.

To show that A is NP-Complete we can try to find a known NP-Complete problem B such that $B \leq A$.

HAMILTONIAN CYCLE $\leq$ TSP

TSP

Input: A weighted complete graph G and a number K.
Goal: Is there a Hamiltonian cycle of length at most $\leq K$ in G?

HAMILTONIAN CYCLE

Input: A graph G.
Goal : Is there a Hamiltonian cycle in G?

Let $x = G$ be input to HC. We construct a complete graph G' with $w(e) = 0$ if $e \in G$ and $w(e) = 1$ if $e \notin G$. Then set $K = 0$. This will be the input to the TSP.

Other NP-Complete problems

Exact Cover
Given a set of subsets of a set M, is it possible to find a selection of the subsets such that each element in M is in exactly one of the subsets?

Subset Sum
Given a set P of positive integers and an integer K, is there a subset of the numbers in P with sum K?

Integer Programming
Given an $m \times n$-matrix A, an m-vektor b, an n-vektor c and a number K, is there an n-vektor x with integer coefficients such that $Ax \leq b$ and $c \cdot x \geq K$?

If we relax the condition that the coefficients x should be integers we get a special case of **Linear Programming**.

Subgraph isomorphism is NP-Complete

Given two graphs G_1 and G_2, Is G_1 a subgraph of G_2?

The problem obviously belongs to NP.

We reduce from Hamilton Cycle.

A graph $G = (V, E)$ contains a Hamiltonian cycle if and only if it contains a subgraph that is a cycle C with $|V|$ nodes. So we can set $G_1 = C$ and $G_2 = G$. som G.

Partition is NP-Complete

Given a set S of positive integers.
Can we split S into two disjoint parts S_1 and S_2 such that $\sum_{x \in S_1} x = \sum_{x \in S_2} x$?

The problem is obviously in NP.

We reduce from Subset Sum:
Given an instance $p_1, p_2, \ldots, p_n$ and K of Subset Sum we create the following instance of Partitioning: Set $P = \sum p_i$

$$p_1, p_2, \ldots p_n, P - 2K$$

if $K \leq P/2$ and

$$p_1, p_2, \ldots p_n, 2K - P$$

otherwise.

There is a partitioning precisely when there is a subset sum K.

0/1-programming is NP-Complete

Given an $m \times n$-matris A and an m-vektor b. **Is there an n-vektor x with coefficients $\in \{0, 1\}$ such that $Ax \leq b$?**

The problem is in NP since, given x, we can check in time $O(n^2)$ if $Ax \leq b$.

We reduce from 3-CNF-SAT:
Let Φ be an instance of 3-CNF-SAT With n variables. To each x_i in Φ we define a corresponding variable $y_i \in \{0, 1\}$ and let 1 Mean True and 0 mean False.

FOr each clause $c_j = l_1 \vee l_2 \vee l_3$ we define an inequality

$$T(l_1) + T(l_2) + T(l_3) \geq 1$$

where $T(x_i) = y_i$ and $T(\neg x_i) = (1 - y_i)$.

And that's it!