Deciding languages in NP

**Theorem.** If \( L \in \text{NP} \), then there exists a deterministic Turing machine \( M \) and a polynomial \( p(n) \) such that

- \( M \) decides \( L \) and
- \( T_M(n) \leq 2^{p(n)} \), for all \( n \in \mathbb{N} \).

**Proof:** Suppose \( L \) is accepted by a non-deterministic machine \( M_{nd} \) whose running time is bounded by the polynomial \( q(n) \).

To decide whether \( w \in L \), the machine \( M \) will

1. determine the length \( n \) of \( w \) and compute \( q(n) \).
2. simulate all executions of \( M_{nd} \) of length at most \( q(n) \). If the maximum number of choices of \( M_{nd} \) in one step is \( r \), there are at most \( r^{q(n)} \) such executions.
3. if one of the simulated executions accepts \( w \), then \( M \) accepts \( w \), otherwise \( M \) rejects \( w \).

The overall complexity is bounded by \( r^{q(n)} \cdot q'(n) = O(2^{p(n)}) \), for some polynomial \( p(n) \).

**An alternative characterization of NP**

- **Proposition.** \( L \in \text{NP} \) if and only if there exists \( L' \in \text{P} \) and a polynomial \( p(n) \) such that for all \( w \in \Sigma^* : \)

  \[
  w \in L \iff \exists v \in (\Sigma')^* : |v| \leq p(|w|) \text{ and } (w, v) \in L'
  \]

- Informally, a problem is in \( \text{NP} \) if it can be solved non-deterministically in the following way:
  1. guess a solution/certificate \( v \) of polynomial length,
  2. check in polynomial time whether \( v \) has the desired property.

**Propositional satisfiability**

- **Satisfiability problem SAT**

  **Instance:** A formula \( F \) in propositional logic with variables \( x_1, \ldots, x_n \).

  **Question:** Is \( F \) satisfiable, i.e., does there exist an assignment \( I : \{x_1, \ldots, x_n\} \rightarrow \{0, 1\} \) making the formula true?

  - Trying all possible assignments would require exponential time.
  - Guessing an assignment \( I \) and checking whether it satisfies \( F \) can be done in (non-deterministic) polynomial time. Thus:

    - **Proposition.** \( \text{SAT} \) is in \( \text{NP} \).
Polynomial reductions

- A polynomial reduction of \( L_1 \subseteq \Sigma^*_1 \) to \( L_2 \subseteq \Sigma^*_2 \) is a polynomially computable function \( f : \Sigma^*_1 \to \Sigma^*_2 \) with \( w \in L_1 \iff f(w) \in L_2 \).

- **Proposition.** If \( L_1 \) is polynomially reducible to \( L_2 \), then
  1. \( L_1 \in P \) if \( L_2 \in P \) and \( L_1 \in NP \) if \( L_2 \in NP \)
  2. \( L_2 \not\in P \) if \( L_1 \not\in P \) and \( L_2 \not\in NP \) if \( L_1 \not\in NP \).

- \( L_1 \) and \( L_2 \) are polynomially equivalent if they are polynomially reducible to each other.

NP-complete problems

- A language \( L \subseteq \Sigma^* \) is NP-complete if
  1. \( L \in NP \)
  2. Any \( L' \in NP \) is polynomially reducible to \( L \).

- **Proposition.** If \( L \) is NP-complete and \( L \in P \), then \( P = NP \).

- **Corollary.** If \( L \) is NP-complete and \( P \neq NP \), then there exists no polynomial algorithm for \( L \).

Structure of the class NP

![Structure of the class NP](image)

**Fundamental open problem:** \( P \neq NP \)?

Proving NP-completeness

- **Theorem** (Cook 1971). SAT is NP-complete.

- **Proposition.** \( L \) is NP-complete if
  1. \( L \in NP \)
  2. there exists an NP-complete problem \( L' \) that is polynomially reducible to \( L \).

- **Example:** INDEPENDENT SET

  Instance: Graph \( G = (V, E) \) and \( k \in \mathbb{N}, k \leq |V| \).
  Question: Is there a subset \( V' \subseteq V \) such that \( |V'| \geq k \) and no two vertices in \( V' \) are joined by an edge in \( E \)?