Reductions and $\mathcal{NP}$–Completeness

`Wed Sep 24 02:54:44 PM PDT 2025`

Reductions

If L_1, L_2 are languages over the alphabets Σ_1 and Σ_2, respectively, a *reduction* from L_1 to L_2 is function $R : \Sigma_1^* \to \Sigma_2^*$ such that $R(w) \in L_2$ if and only if $w \in L_1$. We write $L_1 \leq_R L_2$. We say "L_1 reduces to L_2."[1] If R is $\mathcal{P}$–TIME, we write $L_1 \leq_{\mathcal{P}} L_2$. Reductions are used often in practice to shortcut calculations. A problem that can be easily reduced to an easy problem is easy.

Remark 1 *If $L_1 \leq_{\mathcal{P}} L_2$ and L_2 is $\mathcal{P}$, then L_1 is $\mathcal{P}$.*

Instances. A reduction from a problem to another need only be defined on instances of of the first problem. since we can let $R(w) = \lambda$ if w is not an instance. Reductions are typically defined only on instances.

A language L is in the class $\mathcal{NP}$–TIME (or simply $\mathcal{NP}$) if there is a non-deterministic machine M which which accepts L is time which is polynomial in the number of input bits of the given instance of L.[2] The computation tree of M given an input w is a binary tree whose height is a polynomial function of $|w|$, the number bits required to write w. The input is *accepted* by if M is in an accepting state at at least one leaf of the computation. Thus, L can be decided by a deterministic machine in exponential time, by simply exploring the entire computation tree. But could we determine the answer in polynomial time? That is an unsolved problem, the famous "$\mathcal{P} = \mathcal{NP}$" problem.

[1] We usually mean that R is recusive, *i.e.* computable.

[2] We can assume that at any step, M has at most two legal choices.

Verification Definition of $\mathcal{NP}$

A language L is $\mathcal{NP}$ if and only if there is some machine V and some integer k such that:

1. For every $w \in L$ there exists a string c, called a *certificate* for w, such that V accepts the string (w, c) in $O(n^k)$ time, where $n = |w|$.
2. If $w \notin L$ and c is any string, V does not accept the string (w, c).

$\mathcal{NP}$–Completeness

We define a language L to be $\mathcal{NP}$*–complete* if

1. $L \in \mathcal{NP}$, and
2. Every $\mathcal{NP}$ language reduces to L in polynomial time.

Theorem 1 *If there is any language which is both $\mathcal{P}$–TIME and $\mathcal{NP}$–complete, then $\mathcal{P} = \mathcal{NP}$.*

Proof: Suppose that there is a language L_1 which is both $\mathcal{P}$–TIME and $\mathcal{NP}$–complete. Let Let L_2 be any $\mathcal{NP}$ language Then $L_2 \leq_{\mathcal{P}} L_1$ by the definition of $\mathcal{NP}$-completeness. Since L_1 is $\mathcal{P}$, L_2 is $\mathcal{P}$ by Remark rem: P implies P. □

Boolean Satisfiability

Many $\mathcal{NP}$–COMPLETE problems (languages) have been identified, and the number grows constantly. The first such problem identified is SAT, Boolean satisfiability, proved $\mathcal{NP}$–complete by Theorem 2, the Cook Levin theorem. Using that theorem and Theorem 3, thousands (or more) additional $\mathcal{NP}$–complete problems have been found.

Let Bool be the languages of all *Boolean expressions*, defined to be expressions consisting of variables and operators, where all variables have Boolean type and all operators are Boolean. To shorten our notation, we use "+" for *or*, "·" for *and* and "!" for *not*. An *assignment* of a Boolean expression E is an assignment of truth values (there are only two truth values, *true* $= 1$ and *false* $= 0$) to each variable that appears in E. An assignment is *satisfying* if given those values, E is *true*. E is *satisfiable* if it has a satisfying assignment, otherwise E is a *contradiction*. For example, $x{\cdot}!x$ is a contradiction, since its value is false regardless of the assigned value of x, while $x{\cdot}!y$ is satisfiable, because the assignment $x = 1$, $y = 0$ is satisfying. Let SAT $\subseteq$ BOOL be the satisfiable expressions. We also write SAT to be the problem of determining whether $E \in$ SAT. Any satisfying assignment of a E is a certificate which verifies that $E \in$ SAT.

Theorem 2 (Cook-Levin) *SAT is $\mathcal{NP}$–complete.*

The proof of Theorem 2 is long, but straightforward. You can find it in books or on the internet.

Theorem 3 *If L_1 is $\mathcal{NP}$-complete and L_2 is $\mathcal{NP}$, and there is a polynomial reduction R_1 of L_1 to L_2, hence L_2 is $\mathcal{NP}$–complete.*

Proof: We need only prove that every $\mathcal{NP}$ language reduces to L_2 in polynomial time. Let $L_3 \in \mathcal{NP}$. Since L_1 is $\mathcal{NP}$-complete, there is a polynomial time reduction R_2 of L_3 to L_1. The composition $R_2 \circ R_1$ is a polynomial time reduction of L_3 to L_2. $\square$

Here is a reduction chain of $\mathcal{NP}$–complete problems.

$$SAT \leq_{\mathcal{P}} 3 - SAT \leq_{\mathcal{P}} IND \leq_{\mathcal{P}} SubsetSum \leq_{\mathcal{P}} Partition$$

These problems and reductions are described below.

k-SAT

A Boolean expression is in CNF, *conjuctive normal form* if it is the conjunction (and) of *clauses*, each of which is the disjunction (or) of *terms*, each of which is either a variable or the negation (not) of a variable. CNF $\subseteq$ BOOL is the set of all Boolean expressions written in conjunctive normal form, while k-CNF $\subseteq$ CNF is the subset where each clause has at most k terms.

Note that k-CNF $\subseteq$ CNF $\subseteq$ BOOL.

We define k-SAT $=$ k-CNF $\cap$ SAT.

Theorem 4 *For any $k \geq 3$, k-SAT is $\mathcal{NP}$-complete.*

Theorem 5 *2-SAT is $\mathcal{P}$–sc time.*

We postpone the proofs of Theorems 4 and 5.

Independent Set

An instance of the independent set problem, abbreviated IND, is an ordered pair (G, K) where G is a graph and K is a positive integer. We say a set of vertices of G is *independent* if no two are connected by an edge. A solution (certificate) of (G, K) is an independent set of K vertices of G, thus IND is $\mathcal{NP}$ by the verification definition of $\mathcal{P}$. We give a polynomial time reduction R of 3-SAT to IND. We define R only on 3-CNF, the language of instances of 3-SAT. By Theorem 3, IND is $\mathcal{NP}$-complete.

Subset Sum

An instance of the subset sum problem consists of a sequence of numbers $\sigma = x_1, \ldots x_k$, together with a number K. That instance has a solution if there is some subsequence of σ whose sum is K. Without loss of generality, we assume that the x_k are positive. A subset of sum K is an easily verified certificate, hence subset sum is $\mathcal{NP}$. We give a polynomial time reduction of IND to subset sum, and thus subset sum is $\mathcal{NP}$-complete.

Partition

An instance of the partition problem is a sequence $\tau = y_1, \ldots y_k$ of positive numbers. A solution to τ is a subsequence of τ whose sum is half the sum of the terms of τ, which is an easily verified certificate. We give a polynomial time reduction of subset sum to partition, and thus partition is $\mathcal{NP}$-complete.