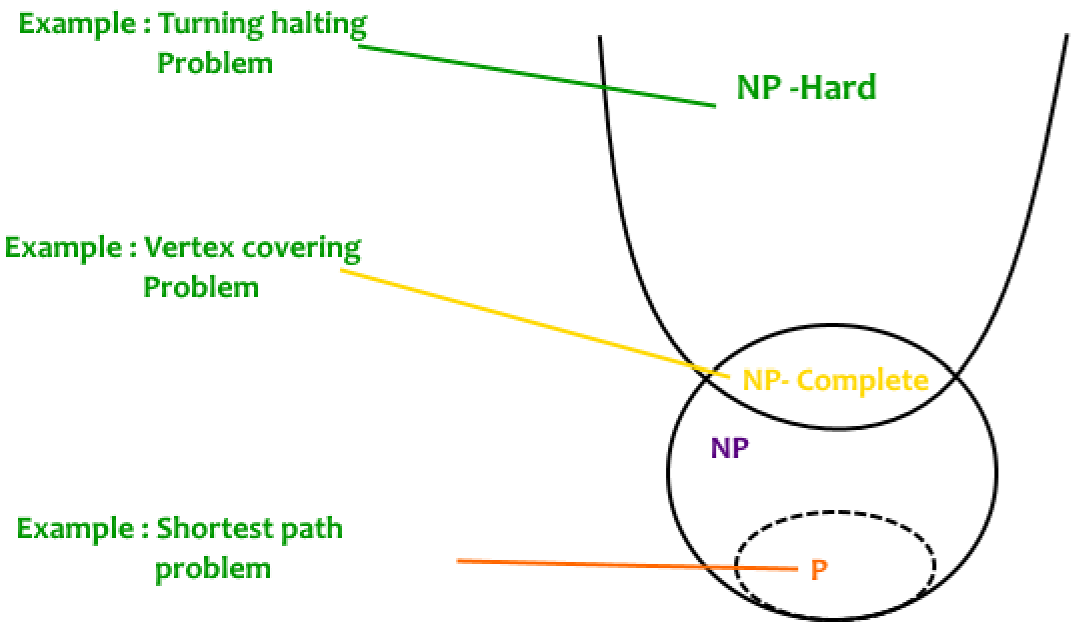
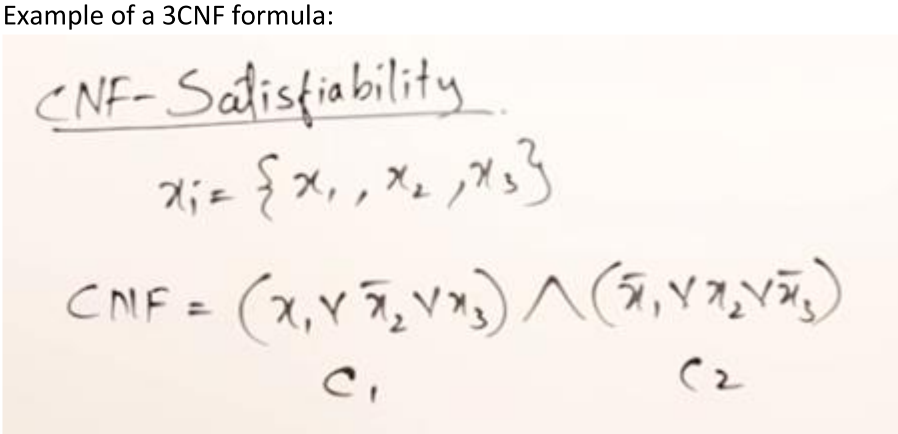
- NP is the set of decision problems for which the program instances, where the answer is “yes”, have proofs verifiable in polynomial time.
- Decision problems are assigned complexity classes (such as NP) based on the fastest known algorithms. Therefore, decision problems may change classes if faster algorithms are discovered.
- It is easy to see that the complexity class P (all problems solvable, deterministically, in polynomial-time) is contained in NP, because if a problem is solvable in polynomial time then a solution is also verifiable in polynomial time by simply solving the problem.
- An algorithm solving a NP-complete problem in polynomial time is also able to solve any other NP problem in polynomial time.



Here P != NP

-
- Example of a 3CNF formula:



Video: https://www.youtube.com/watch?v=e2cF8a5aAhE

- NP is a complexity class that represents the set of all decision problems for which the instances where the answer is “yes” have proofs that can be verified in polynomial time.
- NP-complete is a complexity class which represents the set of all problems X in NP for which it is possible to reduce any other NP problem Y to X in polynomial time. Intuitively, this means that we can solve Y quickly if we know how to solve X quickly. E.g. it can be shown that every NP problem can be reduced to 3-SAT.
- What makes NP-complete problems important is that if a deterministic polynomial time algorithm can be found to solve one of them, then every NP problem is solvable in polynomial time (one problem to rule them all).
- NP-hard: intuitively, these are the problems that are at least as hard as the NP-complete problems/ Note that NP-hard problems do not have to be in NP, and they do not have to be decision problems. NP-hard definition: a problem X is NP-hard, if there is an NP-complete problem Y, such that Y is reducible to X in polynomial time.
- The 0/1 knapsack problem is NP-complete.
- P1054 we define an **abstract problem** Q to be a binary relation on a set I of problem instances and a set S of problem solutions.
- The theory of NP-completeness restricts attention to decision problems only.
- P1055 A computer algorithm that “solves” some abstract decision problem actually takes an encoding of a problem instance as input. We call a problem whose instance set is the set of binary strings a **concrete problem**.
- P1056 We say that a function f: {0,1}* -> {0,1}* is **polynomial-time computable** if there exists a polynomial-time algorithm A that, given any input x ∈ {0,1}*, produces as output f(x). For some set I of problem instances, we say that two encoding e1 and e2 are **polynomially related** if there exist two polynomial-time computable functions f12 and f21 such that for any i ∈ I, we have f12(e1(i)) = e2(i) and f21(e2(i)) = e1(i).
- If two encodings e1 and e2 of an abstract problem are polynomially related, whether the problem is polynomial-time solvable or not is independent of which encoding we use.
- P1056 Lemma 34.1
  Let Q be an abstract decision problem on an instance set I, and let e1 and e2 be polynomially related encodings on I. Then, e1(Q) ∈ P if and only if e2(Q) ∈ P.

- P1057 formal language framework
- P1058 concatenation and closure of L
- P1058 From the point of view of language theory, the set of instances for any decision problem Q is simply the set $\sum$ * where $\sum\ = \{0,1\}$.
- Note: a language is a set.
- P1058 The language **accepted** by an algorithm A is the set of strings $L = \{x \in \{0,1\} *: A(x) = 1\}$. An algorithm A rejects a string x if A(x) = 0 .
- A language L is **decided** by an algorithm A if every binary string in L is accepted by A and every binary string not in L is rejected by A.
- To accept a language, an algorithm need only produce an answer when provided a string in L, but to decide a language, it must correctly accept or reject every string in {0,1}*.
- P1059 P = $\{L \in \{0,1\} *: there\ exists\ an\ algorithm\ A\ that\ decides\ L\ in\ polynomial\ time\}$
- Theorem 34.2

P = {L: L is accepted by a polynomial-time algorithm}
i.e. P is the class of languages that can be accepted in polynomial time

- P1061 Formally, a **Hamiltonian cycle** of an undirected graph G = (V,E) is a simple cycle that contains each vertex in V. A graph that contains a Hamiltonian cycle is said to be Hamiltonian.
- HAM-CYCLE = {<G>: G is a Hamiltonian graph}
- P1063 We define a verification algorithm as being a two-argument algorithm A, where one argument is an ordinary input string x and the other is a binary string y called a certificate.
- P1064 A language belongs to NP if and only if there exist a two-input polynomial-time algorithm A and a constant c such that

  L = {x ∈ {0,1}*: there exists a certificate y with |y|=O(|x|^c) such that A(x,y) = 1}

  We say that algorithm A verifies language L in polynomial time.

- If L ∈ P, then L ∈ NP.
- We define the complexity class **co-NP** as the set of languages L such that the complement of L ∈ NP.
- Fig 34.3 (d)
- P1067 If any NP-complete problem can be solved in polynomial time, then every problem in NP has a polynomial-time solution, that is, P = NP.
- The NP-complete languages are the "hardest" languages in NP.
- P1067 definition of polynomial-time reducible
- P1068 If L1 $\leq_P$ L2, then L1 is not more than a polynomial factor harder than L2, which is why the "less than or equal to" notation for reduction.
- P1069 A language L ⊆ $\{0,1\}$* is **NP-complete** if
  1. L ∈ NP, and
  2. L' $\leq_P$ L for every L' ∈ NP.

  If a language L satisfies property 2, then L is **NP-hard**.
  (For NP-completeness proof, we usually use Lemma 34.8 in 34.4)

- NP-completeness is critical in deciding whether P = NP.
- Theorem 34.4
  If any NP-complete problem is polynomial-time solvable, then P = NP. Equivalently, if any problem in NP is not polynomial-time solvable, then no NP-complete problem is polynomial-time solvable.

- Since no polynomial-time algorithm for any NP-complete problem has yet been discovered, a proof that a problem is NP-complete provides excellent evidence that it is intractable.
- P1071 A **Boolean combinational circuit** consists of one or more Boolean combinational elements interconnected by wires.
- P1071 The number of element inputs fed by a wire is called the **fan-out** of the wire.

- Boolean combinational circuits cannot contain cycles.
- P1072 a one-output Boolean combinational circuit is **satisfiable** if it has a **satisfying assignment**: a truth assignment that causes the output of the circuit to be 1.
- CIRCUIT-SAT= {<C>: C is a satisfiable Boolean combinational circuit}
- P1078 Lemma 34.8
  If L is a language such that L' $\leq_P$ L for some L' ∈ NPC, then L is NP-hard. If, in addition, L ∈ NP, then L ∈ NPC.

- SAT = {<φ>: φ is a satisfiable Boolean formula}
- P1080 Theorem 34.9
  Satisfiability of Boolean formulas is NP-complete.

3-CNF satisfiability

- P1082 A **literal** in a Boolean formula is an occurrence of a variable or its negation.
- A Boolean formula is in **conjunctive normal form**, or **CNF**, if it is expressed as an AND of clauses, each of which is the OR of one or more literals.
- See P1082 for an example of 3-CNF formula
- P1082 Theorem 34.10
  Satisfiability of Boolean formulas in 3-CNF is NP-complete.

- P1084
  NOT(a AND b) = NOT a OR NOT b
  NOT(a OR b) = NOT a AND NOT b

- In contrast to those more general problems which are NP-complete, 2-satisfiability can be solved in polynomial time.

The clique problem

- P1086 A **clique** in an <u>undirected graph</u> G = (V,E) is a subset V' ⊆ V of vertices, each pair of which is connected by an edge in E. A clique is a complete subgraph of G.
- The size of a clique is the number of vertices it contains.
- Optimization version of a clique problem: find a clique of maximum size
- Decision version of a clique problem:
  CLIQUE = {<G,k>: G is a graph containing a clique of size k}

- Theorem 34.11
  The clique problem is NP-complete

The vertex-cover problem

- P1089 A **vertex cover** of an undirected graph G = (V,E) is a subset V’ ⊆ V such that if (u,v) ∈ E, then u ∈ V’ or v∈ V’ or both.
- A vertex cover for G is a set of vertices that covers all the edges in E.
- **The size of a vertex cover is the number of vertices in it.**
- The optimization version of vertex-cover problem is to find a vertex cover of minimum size in a given graph.
- The decision problem of vertex-cover problem is to determine whether a graph has a vertex cover of a given size k.
  VERTEX-COVER = {<G,k>: graph G has a vertex cover of size k}

- Theorem 34.12
  The vertex-cover problem is NP-complete.

- Reduction of vertex-cover problem: the graph G has a clique of size k if and only if the complement of G G’ has a vertex cover of size |v| - k.
- The size of a vertex cover produced by the approximation algorithm is at most twice the minimum size of a vertex cover.