

Quantum Algorithms

Lecture 4

The class NP: Reducibility and completeness

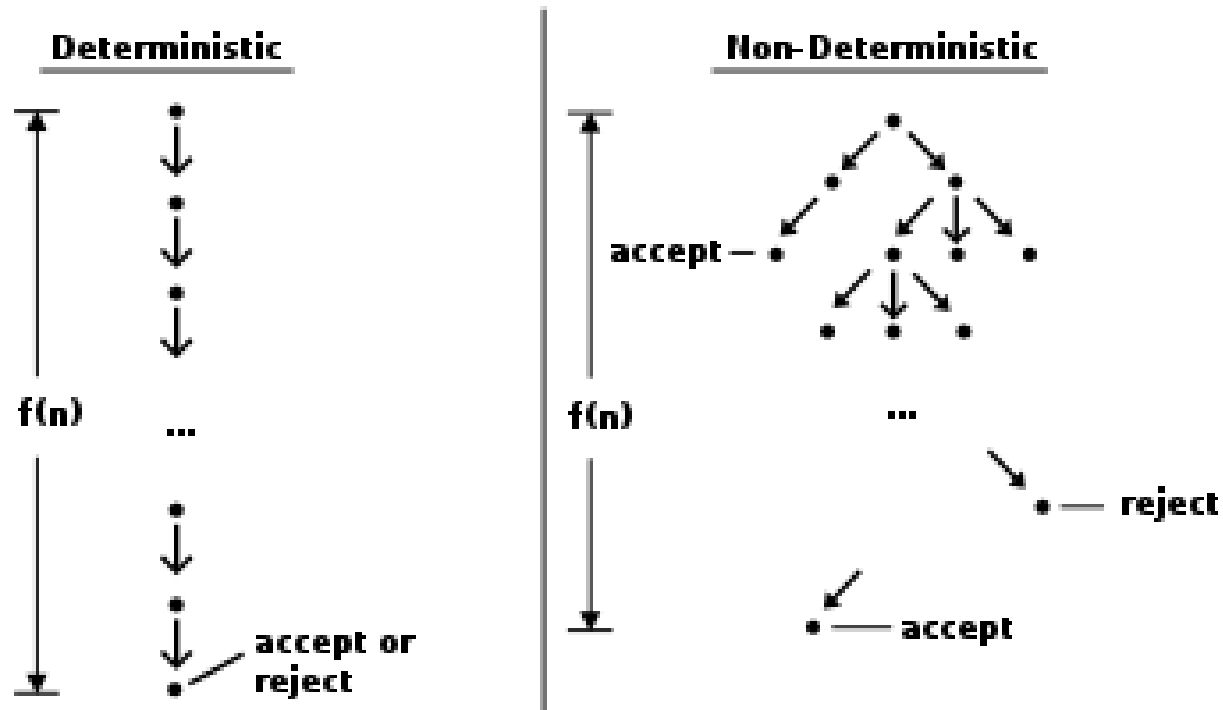
Zhejiang University

Nondeterministic Turing machines

Class NP – nondeterminism

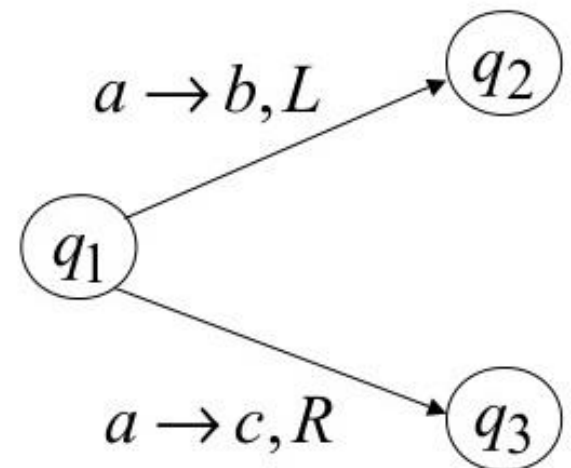
NP is the class of predicates recognizable in polynomial time by "nondeterministic Turing machines."

The class NP is defined only for predicates.



Nondeterministic Turing machine

A nondeterministic Turing machine (NTM) resembles an ordinary (deterministic) machine, but can nondeterministically choose one of several actions possible in a given configuration. More formally, a transition function of an NTM is multivalued: for each pair (state, symbol) there is a set of possible actions.



Nondeterministic Turing machine

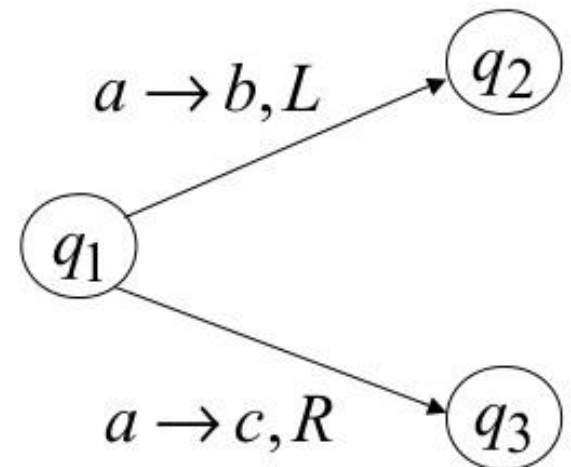
$$\delta(q, s_p) = (q', s', \Delta p)$$

For each pair (q, s_p) can exist more than one transition, as if machine would be able to do this transitions in parallel.

In this example:

$$\delta(q_1, a) = (q_2, b, L)$$

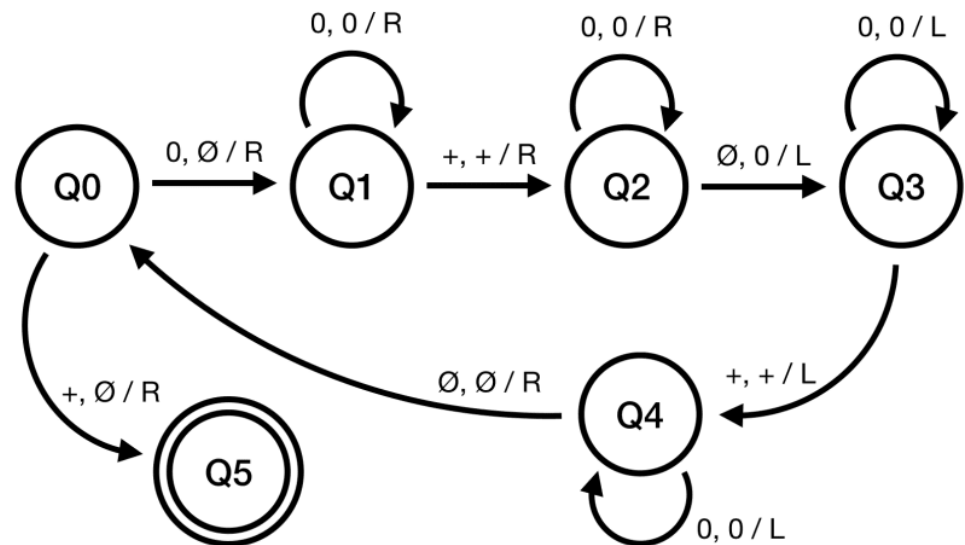
$$\delta(q_1, a) = (q_3, c, R)$$



Nondeterministic Turing machine

$$\delta(q, s_p) = (q', s', \Delta p)$$

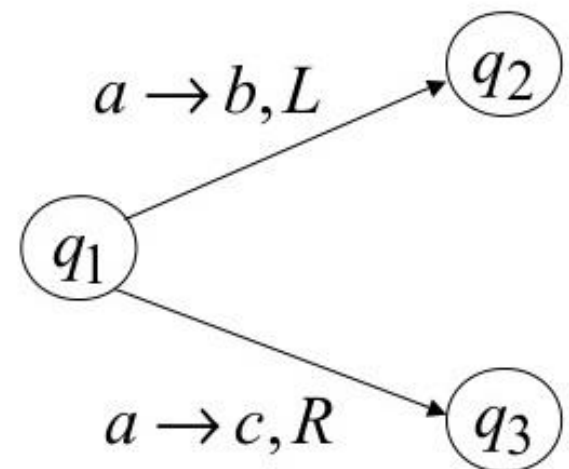
If the set of possible actions has cardinality at most 1 for each state-symbol combination, we get an ordinary Turing machine.



Nondeterministic Turing machine

A computational path of an NTM is determined by a choice of one of the possible transitions at each step; different paths are possible for the same input.

It is like having several parallel computational paths.



NP definition

L belongs to NP.

NTM M and a polynomial $p(n)$.

- $L(x) = 1 \Rightarrow$ there exists a computational path for NTM that gives answer "yes" in time not exceeding $p(|x|)$;
- $L(x) = 0 \Rightarrow$ there is no path with this property (also, no path of any length gives an answer "yes").

No-condition remark

To exclude "yes" answers for long computational paths, it suffices to simulate NTM while counting its steps, and to abort the computation after $p(|x|)$ steps.

P vs NP

By definition $P \subseteq NP$. Is this inclusion strict? Rather intense although unsuccessful attempts have been made over the past 50 years to prove the strictness.

In other words, question whether $P=NP$ is open for a long time and there are even rewards announced for those who will solve the problem. Many believe that $P \neq NP$, but this is also very hard to prove.

Polynomially decidable predicate

A predicate $R(x, y)$ (where x and y are strings) is polynomially decidable (decidable in polynomial time) if there is a (deterministic) TM that computes it in time $\text{poly}(|x|, |y|)$ (which means $\text{poly}(|x| + |y|)$ or $\text{poly}(\max\{|x|, |y|\})$ — these two expressions are equivalent).

NP - second definition

A predicate L belongs to the class NP if it can be represented as

$$L(x) = \exists y((|y| < q(|x|)) \wedge R(x, y)),$$

where q is a polynomial (with integer coefficients), and R is a predicate of two variables decidable in polynomial time.

NP – simplified definition

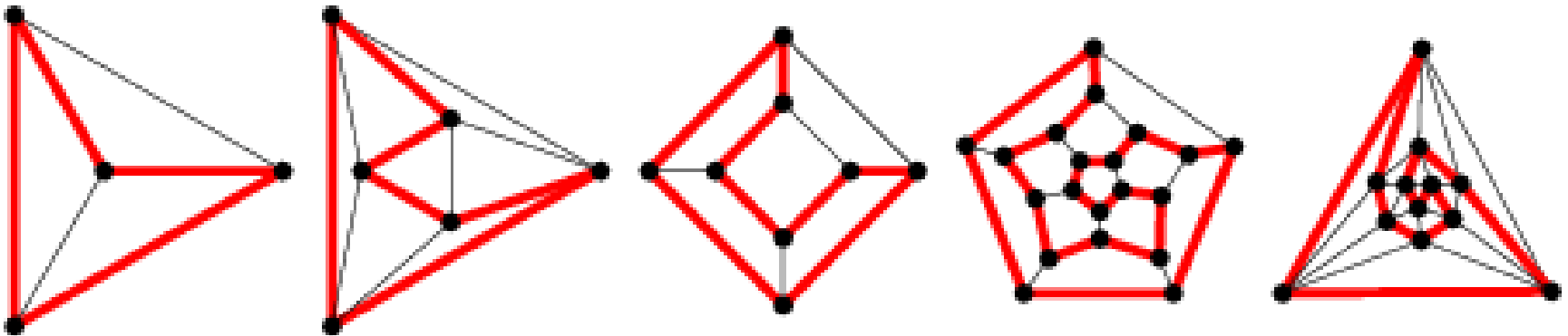
Problems, whose solutions can be checked efficiently, belong to class NP. Given possible solution can be verified in polynomial time.

$R(x, y)$ – here x is given input, and y is provided solution.

Hamiltonian cycle

Hamiltonian cycle in a graph – it is possible to go a cycle through all vertices, by visiting each vertex exactly once.

$L(x) = 1$, if graph x has a Hamiltonian cycle.
This problem belongs to NP.



Hamiltonian cycle in NP

Let $R(x, y) = "y \text{ is a Hamiltonian cycle in the graph } x"$. More precisely, we should say: " x is a binary encoding of some graph, and y is an encoding of a Hamiltonian cycle in that graph". Take $q(n) = n$. Then $L(x)$ means that graph x has a Hamiltonian cycle.

We can check the solution easily in polynomial time.

Equivalence of definitions

Let M be an NTM and let $p(n)$ be the polynomial of the first definition (time complexity). Consider the predicate $R(x, y) = "y \text{ is a description of a computational path that starts with input } x, \text{ ends with answer 'yes', and takes at most } p(|x|) \text{ steps}"$.

R belongs to P. We just must check that we are presented with a valid description of some computational path (this is a polynomial task), and that this path starts with x , takes at most $p(|x|)$ steps, and ends with "yes" (that is also easy).

Equivalence of definitions

Now we have $R(x, y)$ and construct NTM M . We describe two stages for work of M .

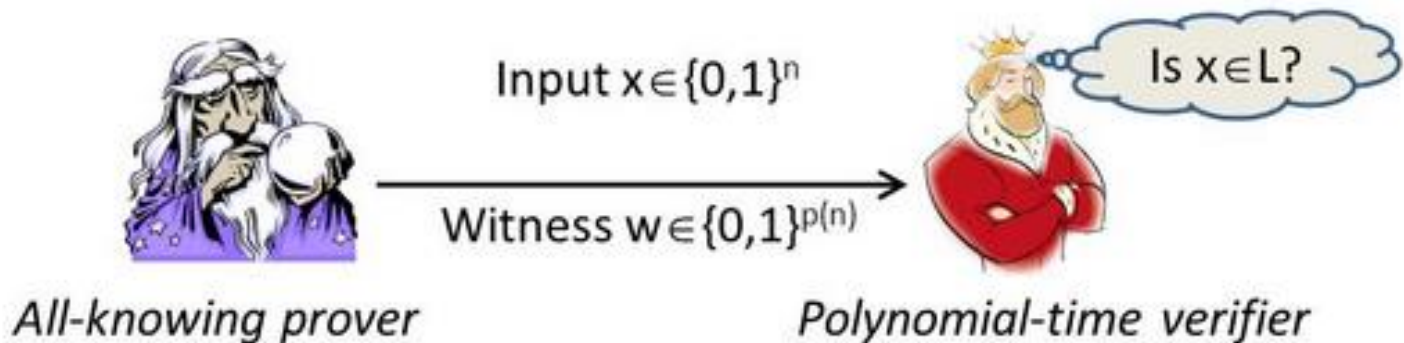
First, M nondeterministically guesses y . More precisely, this means that M goes to the end of the input string, writes $\#$, moves once more to the right, and then writes some string y . After that, the tape has the form $x\#y$ for some y , and M goes to the second stage.

Equivalence of definitions

At the second stage, M checks that $|y| < q(|x|)$ (note that M can write a very long y for any given x) and computes $R(x, y)$. If $x \in L$, then there is some y such that $|y| < q(|x|)$ and $R(x, y)$. Therefore M has, for x , a computational path of polynomial length ending with "yes". If $x \notin L$, no computational path ends with "yes".

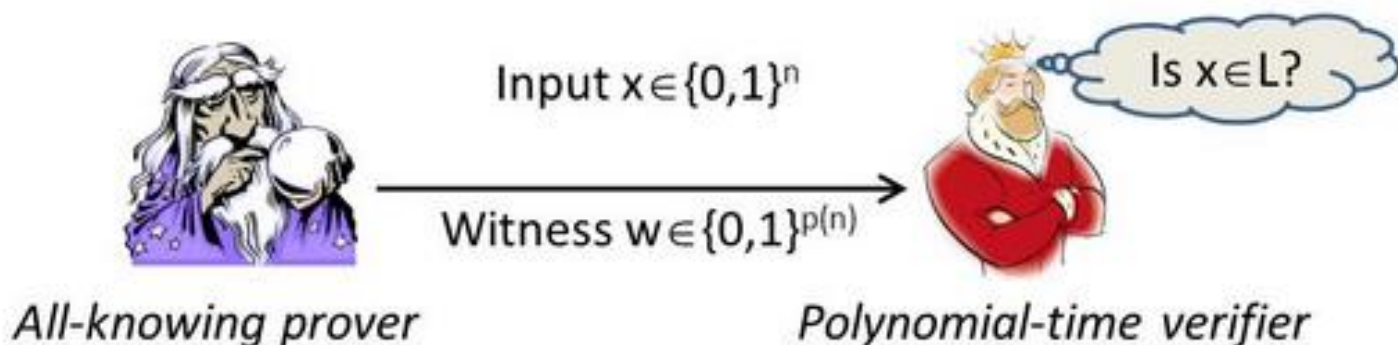
Another description of NP

Imagine two persons: King Arthur (A), whose mental abilities are polynomially bounded, and a wizard Merlin (M), who is intellectually omnipotent and knows everything. A is interested in some property $L(x)$. M wants to convince A that $L(x)$ is true. But A does not trust M and wants to make sure $L(x)$ is true, not just believe M.



Another description of NP

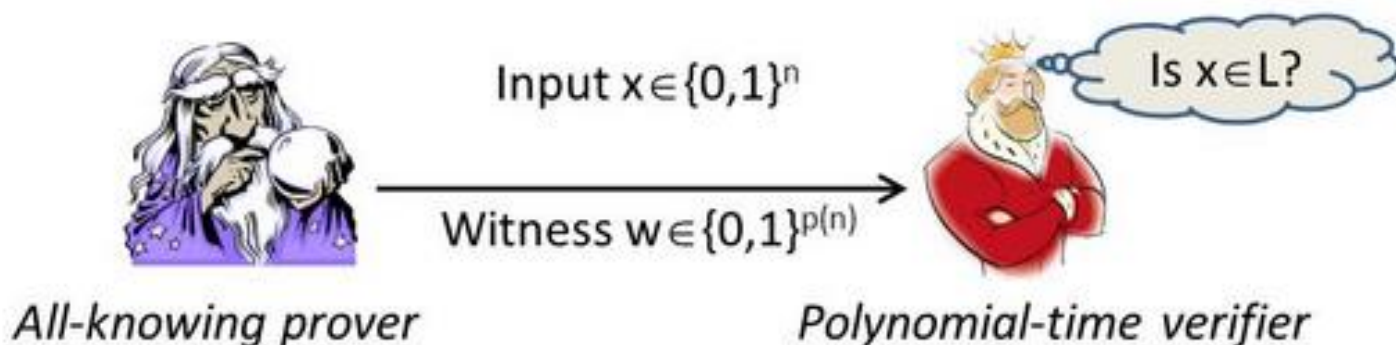
So Arthur arranges that, after both he and Merlin see input string x , M writes a note to A where he "proves" that $L(x)$ is true. Then A verifies this proof by some polynomial proof-checking procedure. The proof-checking procedure is a polynomial predicate $R(x, y) = "y \text{ is a proof of } L(x)"$.



Another description of NP

- $L(x) = 1 \Rightarrow$ M can convince A that $L(x)$ is true by presenting some proof y such that $R(x, y)$;
- $L(x) = 0 \Rightarrow$ whatever M says, A is not convinced: $R(x, y)$ is false for any y .

Moreover, the proof y should have polynomial (in $|x|$) length, otherwise A cannot check $R(x, y)$ in polynomial (in $|x|$) time.



Reducibility and NP-completeness

Karp reducibility

A predicate $L1$ is reducible to a predicate $L2$ if there exists a function $f \in P$ such that $L1(x) = L2(f(x))$ for any input string x .

We say that f reduces $L1$ to $L2$. Notation: $L1 \propto L2$.

Karp reducibility is also called "polynomial reducibility".

Karp reducibility

Let $L1 \propto L2$. Then

- (a) $L2 \in P \Rightarrow L1 \in P$;
- (b) $L1 \notin P \Rightarrow L2 \notin P$;
- (c) $L2 \in NP \Rightarrow L1 \in NP$.

To prove (a) let us note that $|f(x)| = \text{poly}(|x|)$ for any $f \in P$. Therefore, we can decide $L1(x)$ in polynomial time as follows: we compute $f(x)$ and then compute $L2(f(x))$.

Part (b) follows from (a).

Karp reducibility

$$L2 \in NP \Rightarrow L1 \in NP.$$

It can be explained in various ways. Using the Arthur–Merlin metaphor, we say that Merlin communicates to Arthur a proof that $L2(f(x))$ is true (if it is true). Then Arthur computes $f(x)$ by himself and checks whether $L2(f(x))$ is true, using Merlin's proof.

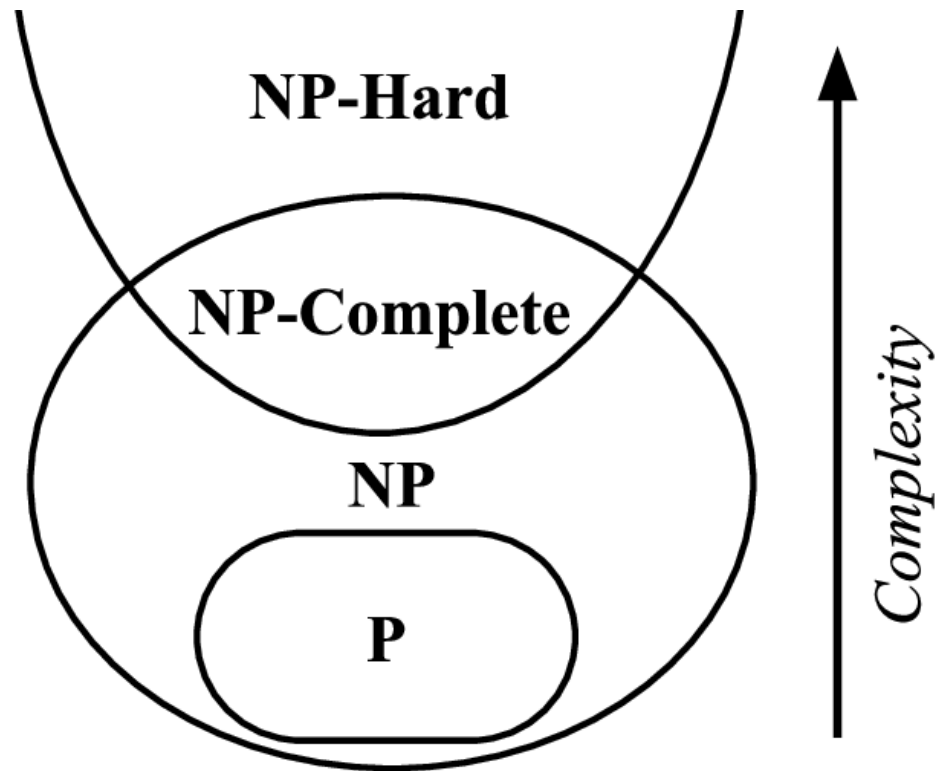
NP-completeness

A predicate $L \in NP$ is NP-complete if any predicate in NP is reducible to it.

If some NP-complete predicate can be computed in time $T(n)$, then any NP-predicate can be computed in time $poly(n) + T(poly(n))$. Therefore, if some NP-complete predicate belongs to P, then $P = NP$. Put it this way: if $P \neq NP$ (which is probably true), then no NP-complete predicate belongs to P.

NP-completeness

If we measure running time "up to a polynomial", then we can say that NP-complete predicates are the "most difficult" ones in NP.



NP-completeness

The key result in computational complexity says that NP-complete predicates do exist. Here is one of them, called satisfiability: $SAT(x)$ means that x is a propositional formula (containing Boolean variables and operations \neg , \wedge , and \vee) that is satisfiable, i.e., true for some values of the variables.

Theorem of Cook, Levin

(1) $\text{SAT} \in \text{NP}$;

(2) SAT is NP-complete.

Corollary. If $\text{SAT} \in \text{P}$, then $\text{P} = \text{NP}$

SAT \in NP - proof

To convince Arthur that a formula is satisfiable, Merlin needs only to show him the values of the variables that make it true. Then Arthur can compute the value of the whole formula by himself.

For example, if our formula is: $x_1 \wedge \neg x_2 \wedge x_3$

Then Merlin can show: $x_1 = 1, x_2 = 0, x_3 = 1$.

SAT is NP-complete

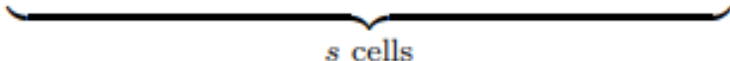
Let $L(x)$ be an NP-predicate and

$$L(x) = \exists y((|y| < q(|x|)) \wedge R(x, y))$$

for some polynomial q and some predicate R decidable in polynomial time.

We need to prove that L is reducible to SAT. Let M be a Turing machine that computes R in polynomial time. Consider the computation table for M working on some input $x\#y$.

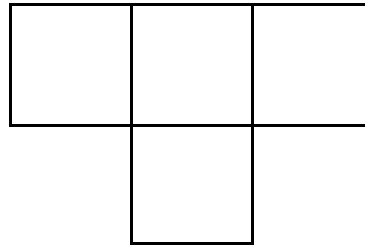
$t = 0$		$\Gamma_{0,1}$		
$t = 1$				
	...			
$t = j$		$\Gamma_{j,k-1}$	$\Gamma_{j,k}$	$\Gamma_{j,k+1}$
$t = j + 1$			$\Gamma_{j+1,k}$	
	...			
$t = T$...			



 s cells

SAT is NP-complete

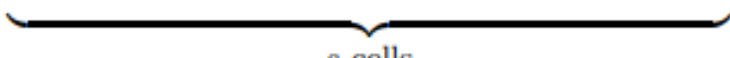
Now we write a formula that says that values of variables form an encoding of a successful computation (with answer "yes"), starting with the input $x\#y$. To form a valid computation table, values should obey some local rules for each four cells configured as follows



SAT is NP-complete

These local rules can be written as formulas in $4t$ variables (if t variables are needed to encode one cell). We write the conjunction of these formulas and add formulas saying that the first line contains the input string x followed by $\#$ and some binary string y , and that the last line contains the answer "yes".

$t = 0$		$\Gamma_{0,1}$		
$t = 1$				
	...			
$t = j$		$\Gamma_{j,k-1}$	$\Gamma_{j,k}$	$\Gamma_{j,k+1}$
$t = j + 1$			$\Gamma_{j+1,k}$	
	...			
$t = T$...			

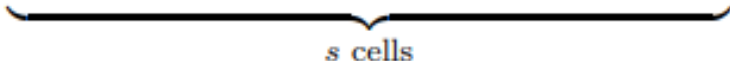


s cells

SAT is NP-complete

The satisfying assignment for our formula will be an encoding of a successful computation of M on input $x\#y$ (for some binary string y). On the other hand, any successful computation that uses at most S tape cells and requires at most T steps can be transformed into a satisfying assignment.

$t = 0$		$\Gamma_{0,1}$		
$t = 1$				
	...			
$t = j$		$\Gamma_{j,k-1}$	$\Gamma_{j,k}$	$\Gamma_{j,k+1}$
$t = j + 1$			$\Gamma_{j+1,k}$	
	...			
$t = T$...			

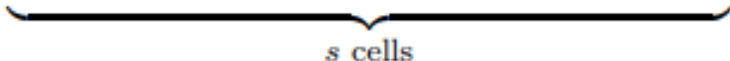


 s cells

SAT is NP-complete

If we consider a computational table that is large enough to contain the computation of $R(x, y)$ for any y such that $|y| < q(|x|)$, and write the corresponding formula as explained above, we get a polynomial-size formula that is satisfiable if and only if $L(x)$ is true. Therefore, L is reducible to SAT.

$t = 0$		$\Gamma_{0,1}$		
$t = 1$				
	...			
$t = j$		$\Gamma_{j,k-1}$	$\Gamma_{j,k}$	$\Gamma_{j,k+1}$
$t = j + 1$			$\Gamma_{j+1,k}$	
	...			
$t = T$...			



 s cells

NP-complete problems

If $SAT \propto L$ and $L \in NP$, then L is NP-complete.
More generally, if $L1$ is NP-complete, $L1 \propto L2$,
and $L2 \in NP$, then $L2$ is NP-complete.

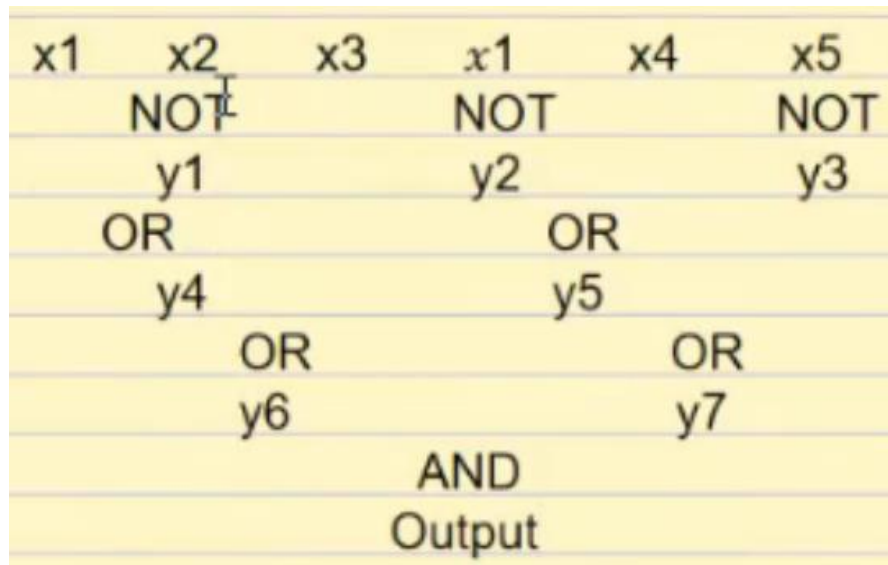
NP-complete problems

The reducibility relation is transitive: if $L1 \propto L2$ and $L2 \propto L3$, then $L1 \propto L3$. (Indeed, the composition of two functions from P belongs to P). According to the hypothesis, any NP-problem is reducible to $L1$, and $L1$ is reducible to $L2$. Therefore any NP-problem is reducible to $L2$.

3-CNF

Recall that a CNF (conjunctive normal form) is a conjunction of clauses; each clause is a disjunction of literals; each literal is either a variable or a negation of a variable. If each clause contains at most three literals, we get a 3-CNF.

$$(x1 \vee \neg x2 \vee x3) \wedge (\neg x1 \vee x4 \vee \neg x5)$$



3-SAT

$3\text{-SAT}(x) = \text{“}x \text{ is a satisfiable 3-CNF”}$.

3-SAT is reducible to SAT (because any 3-CNF is a formula). The next theorem shows that the reverse is also true: SAT is reducible to 3-SAT. Therefore 3-SAT is NP-complete.

SAT \propto 3-SAT

For any propositional formula (and even for any circuit over the standard basis {AND, OR, NOT}), we construct a 3-CNF that is satisfiable if and only if the given formula is satisfiable (the given circuit produces output 1 for some input).

SAT \propto 3-SAT

Let x_1, \dots, x_n be input variables of the circuit, and let y_1, \dots, y_s be auxiliary variables (see the definition of a circuit). Each assignment involves at most three variables (1 on the left-hand side, and 1 or 2 on the right-hand side).

For example, $y_1 = x_1 \wedge x_2$.

SAT \propto 3-SAT

Now we construct a 3-CNF that has variables $x_1, \dots, x_n, y_1, \dots, y_s$ and is true if and only if the values of all y_j are correctly computed (i.e., they coincide with the right-hand sides of the assignments) and the last variable is true. To this end, we replace each assignment by an equivalence (of Boolean expressions) and represent this equivalence as a 3-CNF.

SAT \propto 3-SAT

$$y = (x_1 \vee x_2)$$

$$(x_1 \vee x_2 \vee \neg y) \wedge (\neg x_1 \vee x_2 \vee y) \wedge (x_1 \vee \neg x_2 \vee y) \wedge (\neg x_1 \vee \neg x_2 \vee y),$$

$$y = (x_1 \wedge x_2)$$

$$(x_1 \vee x_2 \vee \neg y) \wedge (\neg x_1 \vee x_2 \vee \neg y) \wedge (x_1 \vee \neg x_2 \vee \neg y) \wedge (\neg x_1 \vee \neg x_2 \vee y),$$

$$y = \neg x$$

$$(x \vee y) \wedge (\neg x \vee \neg y).$$

Finally, we take the conjunction of all these 3-CNF's and the variable y_s (the latter represents the condition that the output of the circuit is 1).

SAT \propto 3-SAT

Let us assume that the resulting 3-CNF is satisfied by some $x_1, \dots, x_n, y_1, \dots, y_s$. If we plug the same values of x_1, \dots, x_n into the circuit, then the auxiliary variables will be equal to y_1, \dots, y_s , so the circuit output will be $y_s = 1$. Conversely, if the circuit produces 1 for some inputs, then the 3-CNF is satisfied by the same values of x_1, \dots, x_n and appropriate values of the auxiliary variables.

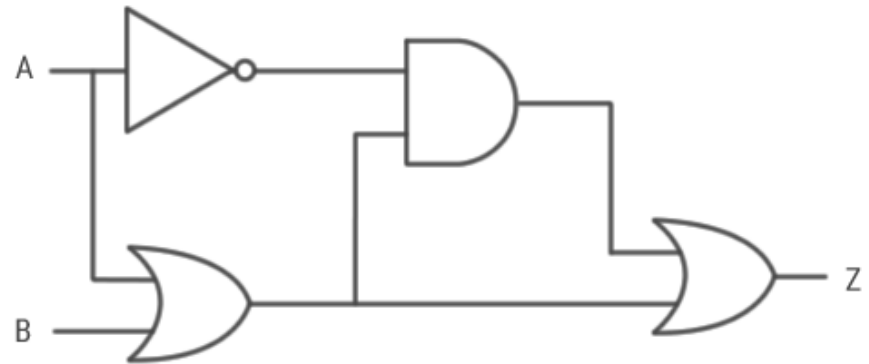
Reduction example

$$y_1 = \neg a \Rightarrow (a \vee y_1) \wedge (\neg a \vee \neg y_1)$$

$$y_2 = a \vee b \Rightarrow (a \vee b \vee \neg y_2) \wedge (\neg a \vee b \vee y_2) \wedge (a \vee \neg b \vee y_2) \wedge (\neg a \vee \neg b \vee y_2)$$

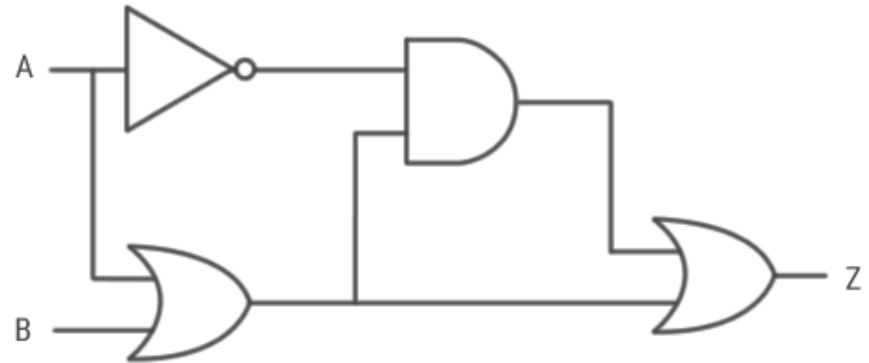
$$y_3 = y_1 \wedge y_2 \Rightarrow (y_1 \vee y_2 \vee \neg y_3) \wedge (\neg y_1 \vee y_2 \vee \neg y_3) \wedge (y_1 \vee \neg y_2 \vee \neg y_3) \wedge (\neg y_1 \vee \neg y_2 \vee y_3)$$

$$z = y_2 \vee y_3 \Rightarrow (y_2 \vee y_3 \vee \neg z) \wedge (\neg y_2 \vee y_3 \vee z) \wedge (y_2 \vee \neg y_3 \vee z) \wedge (\neg y_2 \vee \neg y_3 \vee z)$$



Reduction example

$$\begin{aligned} & (a \vee y_1) \wedge (\neg a \vee \neg y_1) \wedge \\ & (a \vee b \vee \neg y_2) \wedge (\neg a \vee b \vee y_2) \wedge (a \vee \neg b \vee \\ & y_2) \wedge (\neg a \vee \neg b \vee y_2) \wedge \\ & (y_1 \vee y_2 \vee \neg y_3) \wedge (\neg y_1 \vee y_2 \vee \neg y_3) \wedge (y_1 \vee \neg y_2 \\ & \vee \neg y_3) \wedge (\neg y_1 \vee \neg y_2 \vee y_3) \wedge \\ & (y_2 \vee y_3 \vee \neg z) \wedge (\neg y_2 \vee y_3 \vee z) \wedge (y_2 \vee \neg y_3 \vee z) \\ & \wedge (\neg y_2 \vee \neg y_3 \vee z) \wedge \\ & (z) \end{aligned}$$



ILP

Integer linear programming.

Given a system of linear inequalities with integer coefficients, is there an integer solution? (In other words, is the system consistent?)

ILP

In this problem the input is the coefficient matrix and the vector of the right-hand sides of the inequalities.

It is in fact true that, if a system of inequalities with integer coefficients has an integer solution, then it has an integer solution whose binary representation has size bounded by a polynomial in the bit size of the system. Therefore, ILP is in NP.

ILP

$$5x_1 + 7x_2 + 0x_3 + 3x_4 \leq 14$$

$$8x_1 + 0x_2 + 4x_3 + 4x_4 \leq 12$$

$$2x_1 + 10x_2 + 6x_3 + 4x_4 \leq 15$$

$$x_1, x_2, x_3, x_4 \in \{0,1\}$$

3-SAT \propto ILP

Assume that a 3-CNF is given. We construct a system of inequalities that has integer solutions if and only if the 3-CNF is satisfiable. For each Boolean variable x_i we consider an integer variable p_i .

3-SAT \propto ILP

The negation $\neg x_i$ corresponds to the expression $1 - p_i$. Each clause $Xj \vee Xk \vee Xm$ (where X^* are literals) corresponds to the inequality $Pj + Pk + Pm \geq 1$, where Pj, Pk, Pm are the expressions corresponding to Xj, Xk, Xm . It remains to add the inequalities $0 \leq p_i \leq 1$ for all i .

3-SAT \propto ILP

Here we have our 3-SAT formula being equal to F_1 AND F_2 AND F_3 AND F_4 .

$$F_1 = x_1 \vee x_2 \vee x_3 \Rightarrow x_1 + x_2 + x_3 \geq 1$$

$$F_2 = \neg x_1 \vee x_4 \vee \neg x_5 \Rightarrow (1 - x_1) + x_4 + (1 - x_5) \geq 1$$

$$F_3 = \neg x_2 \vee x_5 \vee \neg x_6 \Rightarrow (1 - x_2) + x_5 + (1 - x_6) \geq 1$$

$$F_4 = \neg x_3 \vee x_4 \vee \neg x_5 \Rightarrow (1 - x_3) + x_4 + (1 - x_5) \geq 1$$

It remains to add that $0 \leq x_1, x_2, x_3, x_4, x_5, x_6 \leq 1$

Linear programming

If we do not require the solution to be integer-valued, we get the standard linear programming problem. Polynomial algorithms for the solution of this problem (due to Khachiyan and Karmarkar) are described.

NP-complete problems

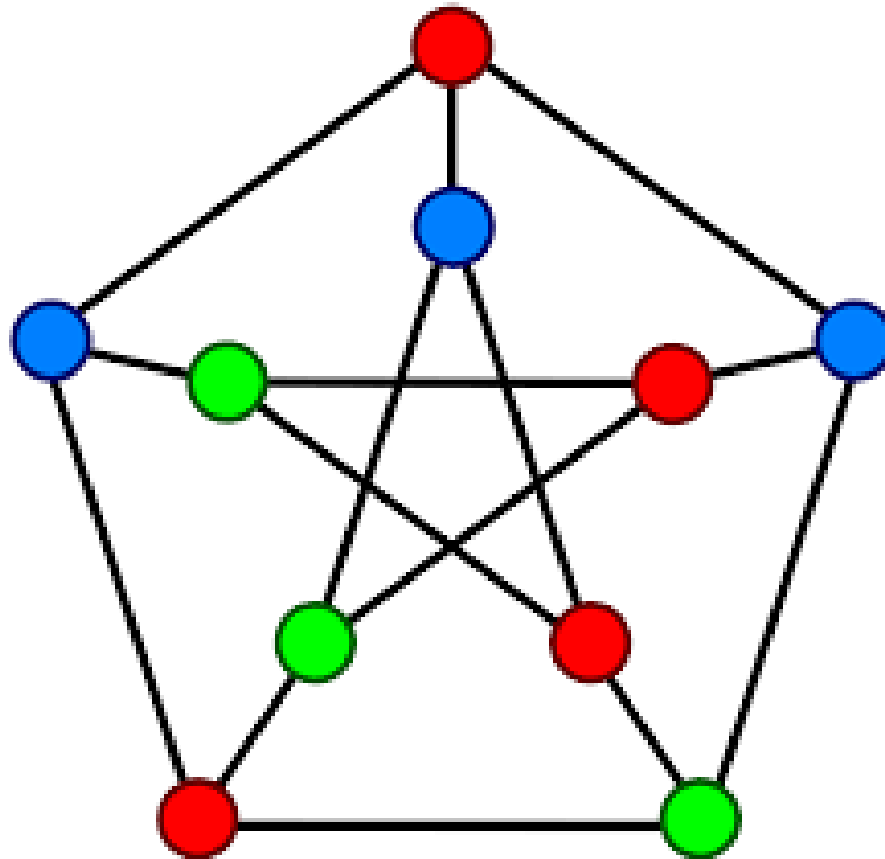
There are hundreds of NP-complete problems:

https://en.wikipedia.org/wiki/List_of_NP-complete_problems

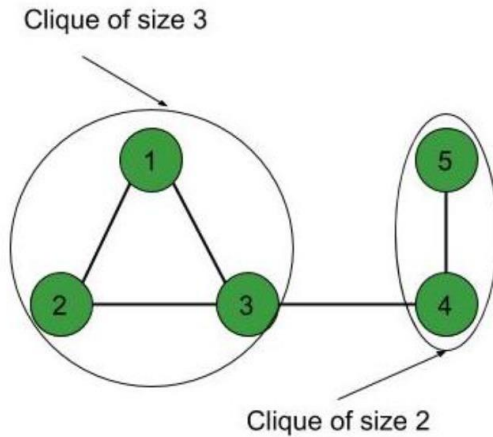
Including from the following subfields:

- Graphs and hypergraphs,
- Mathematical programming,
- Formal languages and string processing,
- Games and puzzles

3-coloring

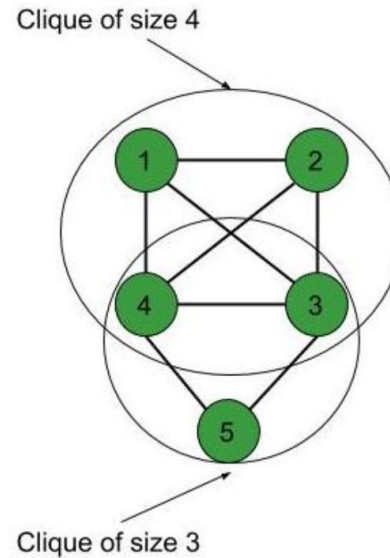


Clique



The above graph contains a maximum clique of size 3

Fig. (1)



The above graph contains a maximum clique of size 4

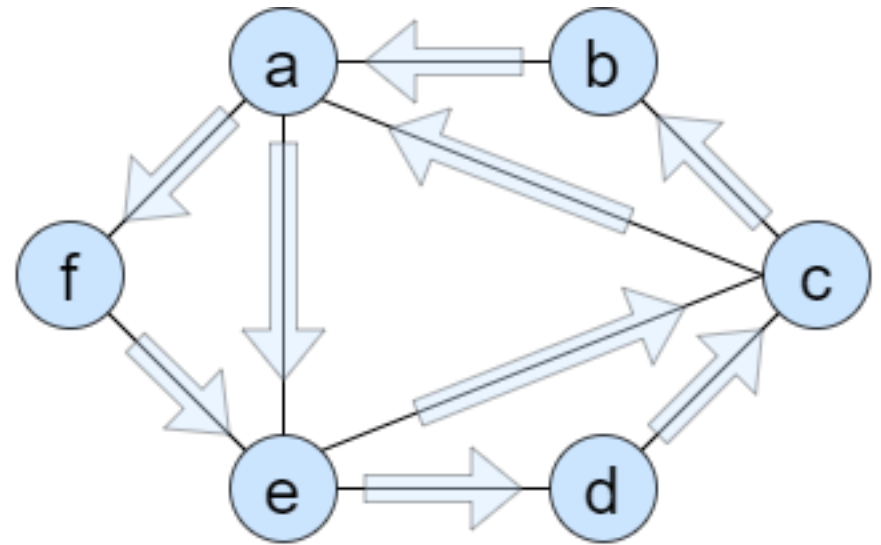
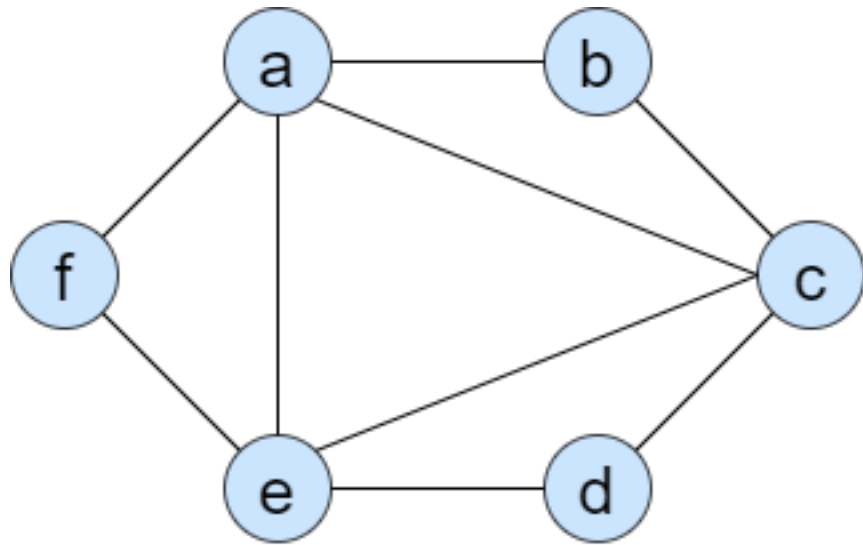
Fig. (2)

* A clique of size 2 is also present in Fig. (2)

P vs NP

P	NP
2-coloring	3-coloring
2-CNF	3-CNF
Euler cycle	Hamilton cycle
LP	ILP

Euler cycle



Euler Circuit: a-e-c-a-f-e-d-c-b-a

**Thank you for your
attention!**