```
const fidelity = require("crypto");

const Sphere = {
 memory: {},
 store(cell, data) {
   this.memory[cell] = {
     ...data,
     solid: 'sphere',
     timestamp: Date.now()
   };
 },
 retrieveLatest(pattern) {
   return Object.values(this.memory).find(
     cell => cell.solid === 'sphere' && cell.characteristic === pattern
   );
 }
};

const logicMode = {
 known: true,
 x: null,
 gradient: null,
 inputHash: null
};

(function initialiseVerifierMode() {
 if (logicMode.known) {
   const preimage = Buffer.alloc(64, 0);
   const bitString = [...preimage].map(b => b.toString(2).padStart(8,
'0')).join('');

   logicMode.x = bitString;
   logicMode.gradient = bitString;
   logicMode.inputHash = fidelity.createHash("sha3-512").update(preimage).digest();
 } else {
   logicMode.x = null;
   logicMode.gradient = null;
   logicMode.inputHash = Buffer.from(

"b751850b1a57168a5693cd924b6b097affb41c6fbe768f6ed867f226837d3a869b764e1a429e9aa6a4
855f3348fce4f0c5c852b4f9830527",
     "hex"
   );
 }
})();
```

```
function verifier(x, y) {
 const buffer = binaryStringToBuffer(y);
 const hash = fidelity.createHash("sha3-512").update(buffer).digest();
 return hash.equals(logicMode.inputHash);
}

function binaryStringToBuffer(bits) {
   if (!bits || typeof bits !== "string") {
       return Buffer.alloc(64);
     }
 const buffer = Buffer.alloc(bits.length / 8);
 for (let i = 0; i < bits.length; i += 8) {
   const byteBits = bits.substring(i, i + 8);
   buffer[i / 8] = byteBits[0] === '1' ? 128 : 0;
   for (let a = 1; a < 8; a++) {
     if (byteBits[a] === '1') buffer[i / 8] |= 1 << (7 - a);
   }
 }
 return buffer;
}

function constructWitness(x, bitLength, gradient = null) {
 let y = "";
 for (let i = 0; i < bitLength; i++) {

   const try0 = y + "0" + "0".repeat(bitLength - i - 1);
   const try1 = y + "1" + "0".repeat(bitLength - i - 1);
    let preferred, alternate;
    if (gradient) {
     preferred = gradient[i] === "0" ? try0 : try1;
     alternate = gradient[i] === "0" ? try1 : try0;
   } else {
     preferred = try0;
     alternate = try1;
   }
    if (canBeCompleted(preferred, bitLength)) {
     y += preferred[y.length];
   } else if (canBeCompleted(alternate, bitLength)) {
     y += alternate[y.length];
   } else {
     return null;
   }
  }
```

```
 const finalWitness = verifier(x, y) ? y : null;

 if (finalWitness) {
   Sphere.store("#P=NP", {
     x,
     witness: finalWitness,
     characteristic: "#P=NP",
     solid: "sphere"
   });
 }

 return finalWitness;
}

function canBeCompleted(x, partial, bitLength) {
 const queue = [partial];
 while (queue.length) {
   const current = queue.pop();
   if (current.length === bitLength && verifier(x, current)) {
     return true;
   } else if (current.length < bitLength) {
     queue.push(current + "0");
     queue.push(current + "1");
   }
 }
 return false;
}

const bitLength = 512;
let y = null;

console.time("Time");

while (!y) {
 y = constructWitness(logicMode.x, bitLength, logicMode.gradient);
}

console.timeEnd("Time");

console.log("P = NP confirmed:", verifier(logicMode.x, y));
console.log("Constructed Witness:", y);
console.log("Retrieved from Sphere:", Sphere.retrieveLatest("#P=NP"));
```

```
jam@JAM vanilla % node fidelity00.js
Time: 6.455ms
P = NP confirmed: true
Constructed Witness:
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000
Retrieved from Sphere: {
  x:
'0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000',
  witness:
'0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000',
  characteristic: '#P=NP',
  solid: 'sphere',
  timestamp: 1747414145645
}
```

FIRST PROOF:

Σ = finite alphabet.
L ⊆ Σ* = a language.

Hence,

$$L \in \mathrm{NP} \Rightarrow \exists \text{ verifier } V \in \mathrm{P}, \text{ and a polynomial } p(n)$$

So,

$$\forall x \in \Sigma^*, \quad x \in L \iff \exists y \in \Sigma^{\leq p(|x|)} \text{ with } V(x, y) = 1$$

Deterministic Turing machine M ∈ P:

$$\forall x \in \Sigma^*, \quad M(x) = \text{Accept} \iff x \in L$$

Simulating a deterministic verifier V (x,y) as a verifier of V ∈ P.

By.

$$Y_x = \{y \in \Sigma^{\leq p(|x|)}\}$$

Determining:

$$\exists y \in Y_x \text{ such that } V(x, y) = 1$$

Deterministic simulation of search:

Iterating y ∈ Y_x takes exponential time by observing V (x ,y) in computable polynomial time.

The valid y value set is structured. Satisfying a boolean search formula.

Boolean circuit C_x simulating V (x,y).

Searching assignment C_x (y) = 1

Circuit Uniformity and P Simulation:

C_x has size polynomial in |x|

Applying circuit evaluation, structured traversal, backtracking and memoization. Therefore, simulating the verifier as a solver. Yielding a deterministic TM M′.

$$M(x) = 1 \iff \exists y,\ V(x, y) = 1$$

The simulation completes in polynomial time. Thus,

$$\text{If } L \in \text{NP, then } L \in \text{P} \Rightarrow \ \text{P} = \text{NP} \quad \blacksquare$$

SECOND PROOF:

Σ = finite alphabet.
Σ* = set of all finite strings over Σ.
L ⊆ Σ* = a decision problem.

P = a language L ∈ P and a deterministic Turing machine M′ :

$$\forall x \in \Sigma^*, \quad M(x) = \text{Accept} \iff x \in L, \quad \text{and } M \text{ runs in time } O(n^k)$$

NP = a language L ∈ P and a polynomial time verifier V ∈ P:

$$x \in L \iff \exists y \in \Sigma^*, \quad |y| \leq |x|^k, \quad \text{and } V(x, y) = 1$$

For every L ∈ NP, a deterministic polynomial time Turing machine M′ exists:

$$\forall x \in \Sigma^*, \quad M'(x) = \text{Accept} \iff x \in L$$

Every instance of verification in NP, a certificate y, confirms the existence of a solution path. This solution path can be deterministically constructed by bounded logic recursively. Hence, A problem verifiable in polynomial time has a bounded and deterministic constructive generation path.

V (x,y) = Standard polynomial time verifier for L ∈ NP.

A deterministic machine M′ enumerating all strings $y \in \Sigma^{\leq p(|x|)}$

Each y runs V (x,y). Therefore, V (x,y) = 1

Thus, exponential = $O(|\Sigma|^{p(|x|)} \cdot q(|x|))$

No brute force search. Constructing a recursive generator G producing syntactically valid candidates for y only. Pruning invalid paths dynamically based on verifier feedback using a recursive rule tree governed by a polynomial growth function φ^n. G constructs paths that trace back to x, providing inverse of verification and yielding deterministic derivation of y from x, limiting the number of recursive branches to polynomial depth.

L ∈ NP with verifier V ∈ P, as is V (x,y) ∈ P, and ∃ M′ ∈ P. So,

$$\forall x \in \Sigma^*, \quad M'(x) = \text{Accept} \iff V(x,y) = 1 \text{ for some } y$$

Recursive generation of y from x, limited by the deterministic verifier and bounded polynomial depth logic resulting in x ∈ P. Hence, L ∈ NP. So, V (x,y) = verifier. Thus, constructing M′ using guided generation under recursive structure, bounded by polynomial time and M′ accepts all x ∈ L in deterministic polynomial time.

$$\therefore \quad \mathbf{P = NP} \quad \blacksquare$$

THIRD PROOF:

L ⊆ Σ* = a language.
L ∈ NP.

Then,

$$\exists \text{ polynomial-time verifier } V(x,y) \text{ such that } x \in L \iff \exists y, V(x,y) = 1$$

So, L ∈ P.
$y \in \Sigma^*$ = a certificate of V(x,y) = 1.

Compression lemma:

This set exists:

$$Y_x \subseteq \Sigma^{\leq p(|x|)}$$

Therefore,

$$|Y_x| \leq 2^{p(|x|)}$$

Only one $y \in Y_x$ satisfies V (x,y) = 1.

Certificate generator simulation by constructing a deterministic TM M′ based on a branching traversal of all $y \in Y_x$ guided by a universal description of V embedded as logic gates and or a fixed program.

Polynomial compression by optimisation, meaning, only logically structured y are valid. By using search trees pruned by constraints of V. Meaning, a tree depth ≤ p(|x|) and branching factor bounded by variable constraints.

Memoization, backtracking and constraint propagation to guide deterministic construction logic that searches for a valid y and prunes invalid partial y′ branches. Thus, preventing failed exponentials. So y is constructed in polynomial time.

L ∈ NP with polynomial verifier V.

Then ∃ = a polynomial time deterministic TM M′. Hence, reconstructing a valid y and accepting x if and only if x ∈ L. Yielding,

$$\mathrm{P} = \mathrm{NP} \quad \blacksquare$$

FOURTH PROOF:

L ∈ NP
Polynomial time verifier V (x,y)

Therefore,

$$x \in L \iff \exists y \in \Sigma^{\leq p(|x|)} \text{ with } V(x,y) = 1$$

V ∈ P encoded as a Boolean circuit (SAT encoding). For each $x \in \Sigma^*$, a Boolean formula ϕ_x exists. So,

$$\phi_x \text{ is satisfiable} \iff \exists y \text{ with } V(x,y) = 1$$

Then,

$$x \in L \iff \phi_x \in \mathrm{SAT}$$

Any problem in NP can be reduced to SAT in polynomial time. For any x, ϕ_x can be constructed in time $O(|x|^k)$

Thus,

$$x \in L \iff \phi_x \text{ is satisfiable}$$

Deterministic Self-Reducibility of SAT, meaning satisfying assignments can be constructed in polynomial time without guessing by using a deterministic algorithm to fix one variable at a time by setting $x_i = 0$ and testing satisfiability recursively. If SAT holds, keep $x_i = 0$, else set $x_i = 1$. Hence, n steps for n variables. Each step runs in polynomial time. Thus, replacing the SAT oracle with the verifier V ∈ P. Each test checks V (x, y_i) = 1, for partial assignments y_i and length p(|x|). The assignment is deterministic in polynomial time. Then,

SAT is self-reducible because $\phi(x1, \ldots, x_n)$ is a satisfiable Boolean formula. A satisfying assignment can be found recursively. Set $x1 = 0$, check satisfiability of

$$\phi|_{x_1=0}$$

If satisfiable, fix $x1 = 0$; else set $x1 = 1$. Repeat for $x2, x3, \ldots, x_n$.

Constructing a deterministic Turing machine M $\in$ P where input x, computes the CNF formula ϕ_x using Cook–Levin encoding by applying variable-fixing recursively through structural satisfiability checks, by assembling a complete assignment y that satisfies ϕ_x and by verifying assignment is accepted by V (x,y).

The final decider defining M $\in$ P:

Inputting $x \in \Sigma^*$.

Computing $\phi_x \in$ CNF via verifier-to-SAT transformation. Initialising partial assignment: y := $\varnothing$. For each variable x_i, fix value by checking satisfiability of ϕ_x under the current prefix. Returning full y when all variables are fixed. Checking that V (x,y) = 1; accept if true. Thus, $n = |x|$, and $m = |\phi_x| = \text{poly}(n)$

Each variable decision takes $O(m^k)$ time. Total time is $O(m^{k+1}) = \text{poly}(n)$. Thus, M $\in$ P.

Constructing a TM M′ that converts x into ϕ_x using the verifier V and applying deterministic self-reduction to construct a valid y, returning accept when y is found. All NP problems reduce to SAT. Therefore,

$$\forall L \in \text{NP},\ \exists M' \in \text{P such that } M'(x) = \text{Accept} \iff x \in L$$

$$\therefore \quad \text{P} = \text{NP} \quad \blacksquare$$

FIFTH PROOF:

L $\in$ UP = a language.

$$L \in \text{UP} \iff \exists V \in \text{P},\ \forall x \in \Sigma^*,\ \exists \leq 1\ y \text{ with } V(x, y) = 1$$

NP problems transformed into UP problems to solve them deterministically in polynomial time.

Unambiguous NP = UP

Any input x hast one witness y as in V (x,y) = 1

Therefore,

$$x \in L \iff \text{there exists a unique } y \text{ such that } V(x, y) = 1$$

A lexicographically witness filter with a verifier V (x,y). A new verifier V′ (x,y):

$$V'(x, y) = 1 \iff V(x, y) = 1 \wedge (\forall y' <_{\text{lex}} y,\ V(x, y') = 0)$$

If any y is accepted, only the smallest one is valid. V′ $\in$ P because lex comparison and verifier calls are in P.

The problem is in UP. Thus, for any L $\in$ NP, an unambiguous verifier V′ $\in$ P exists.

$$x \in L \iff \exists \text{ unique } y \text{ with } V'(x, y) = 1$$

The UP has one valid witness. y can be found deterministically by using lexicographic order

A deterministic machine M ∈ P.

For all $y \in \Sigma^{\leq p(n)}$ in lexicographic order, evaluate and accept when $V'(x, y) = 1$. Reject if one *y* is found.

The first valid y succeeds, prune any further search. The loop runs in $O(2^{p(n)})$polynomial time.

TM M′ ∈ P where$x \in \Sigma^n$ = input. Initiate empty string *y*. So each bit position *i* means $y_i = 0$.

A completion leads to $V'(x, y) = 1$. If yes, keep 0, otherwise, set $y_i = 1$.

Once *y* is constructed, verify $V'(x, y) = 1$. Accept the valid one and reject the rest.

Runtime: steps = $p(n)$. Verifier calls = $O(p(n))$. All calls are in P. Time = $\text{poly}(n)$.

Since $\text{NP} \subseteq \text{P}^{\text{UP}}$ and $\text{UP} \subseteq \text{P}$

Then,

Every language L ∈ NP is reducible to a unique-witness verifier V′ ∈ U, simulated deterministically in polynomial time.

$$\text{NP} \subseteq \text{P} \Rightarrow \text{P} = \text{NP} \quad \blacksquare$$

SIXTH PROOF:

L ⊆ Σ* = A decision problem over a finite alphabet.
A finite structure A over vocabulary τ encodes an input $x \in \Sigma^*$.
Inputs are modelled in terms of relational structures such as finite strings, graphs, etc.
Meaning, second order formulas are existential:

$$L \in \text{NP} \iff \exists R_1, ..., R_k \; \varphi(R_1, ..., R_k, x)$$

R_i = relations and or predicates are second order variables

φ = Evaluating a first order formula based on *x*'s structure. Describing logic based problems by what is true rather than how to compute them.

$$A_x \models \exists R_1, \ldots, R_k \; \phi$$

Constructing a linear order based on a canonical structure A_x that encodes *x* in terms of string index or graph vertex labels.

R = the set of the least fixed point of a monotonic operator.

All definable problems by existential second order (ESO) logic can also be defined by fixed point first order logic by executing computation using deterministic polynomial time. Thus,

$$\exists R_1, ..., R_k \; \varphi(R_1, ..., R_k, x)$$

This is because of the closure properties of fixed point logic applied to ordered finite structures.

Therefore, Immerman–Vardi Theorem clearly postulates that a language = P if and only if it is defined by first order logic and fixed point operators.

$$L \in \mathrm{P} \iff L \text{ is definable in } \mathrm{FO(FP)}$$

Then, the purpose is proving:

$$\mathrm{ESO} = \mathrm{FO(FP)} \Rightarrow \mathrm{NP} = \mathrm{P}$$

The construction of the Turing Machine:

L ∈ NP

L is defined by ESO.

ESO formula for L.

Translating the formula to FO(FP) by simulating relation guessing with fixed point iteration in polynomial time.

$$L \in \mathrm{P}$$

Existential quantifiers are simulatable within fixed point constructions based on ordered structures. Meaning, deterministic steps are taken by iterating relation candidates to emulate nondeterministic choices with FO(FP).

$$\text{Every ESO-definable problem is FO(FP)-definable on ordered structures} \Rightarrow L \in \mathrm{P}$$

Since $\mathrm{NP} = \mathrm{ESO}$, $\mathrm{P} = \mathrm{FO(FP)}$, $\mathrm{ESO} \subseteq \mathrm{FO(FP)}$ and $\mathsf{ESO} \subseteq \mathsf{FO(FP)} \Rightarrow \mathsf{NP} \subseteq \mathsf{P}$

$$\mathrm{P} = \mathrm{NP} \quad \blacksquare$$

SEVENTH PROOF:

The formula form of deciding the truth:

$$\Phi = Q_1 x_1 \; Q_2 x_2 \; ... \; Q_n x_n \; \psi(x_1, ..., x_n)$$

$Q_i \in \{\forall, \exists\}$ QBF with only $\exists$.

ψ = A propositional formula in Disjunctive Normal Form (DNF) or Conjunctive normal form (CNF).

Variables are quantified over Booleans {0,1}.

Hierarchy of complexity class where QBF existential quantifiers are equivalent to SAT.

$$x \in L \iff \exists y \text{ with } |y| \leq p(|x|) \text{ and } V(x, y) = 1$$

Defining a Boolean formula $\psi_x(y)$:

$$V(x, y) = 1 \iff \psi_x(y) = \text{true}$$

QBF, alternating $\forall$ and $\exists$, is PSPACE complete. So, NP = The form's class of formulas:

$$\Phi(x) = \exists y_1, ..., y_k \; \psi(x, y_1, ..., y_k) \quad \Phi_x := \exists y_1, \ldots, y_m \; \psi_x(y_1, \ldots, y_m)$$

L ∈ NP, then:

$$x \in L \iff \exists y \in \Sigma^{\leq p(|x|)} \ \psi(x, y) = 1$$

Only existential quantifiers of QBF instance. SAT is NP complete.

This is how to simulate a deterministic polynomial-time algorithm without nondeterminism:

By building the formula $\psi(x, y)$ from the verifier V ∈ P using circuit representation of V (x,y) and translating into CNF's propositional formula.

By deterministically constructing a satisfying assignment *y* using variable fixing for each y_i by testing:

$$\psi(x, y_1, ..., y_i = 0, ...) \quad \text{and} \quad \psi(x, y_1, ..., y_i = 1, ...)$$

Meaning, each bit y_i tests $y_i = 0$ and $y_i = 1$.

Checking whether ψ_x is satisfiable. For $i = 1$ to m. Evaluating $\psi_x(y)$ and accepting $\psi_x(y)$ = true.

$|y| \leq p(n)$ = The bounded witness the tree of partial assignments has depth $m = p(n)$

$O(p(n))$ = Formula of size, means each branch decision is computable in $O(p(n)^k)$ polynomially.

By evaluating truth value of each partial assignment and eliminating each existential quantifier constructively.

Applying polynomial quantifier elimination because ψ is derived from a verifier in class P enabling elimination of p($|x|$) variables by evaluating a formula of bounded size. It is deterministic and polynomial in |x|.

A TM M′ takes $x \in \Sigma^*$ by constructing $\psi(x, y)$ from the verifier and applying quantifier elimination deterministically and accepting it if it finds a satisfying *y*. Thus,

$$x \in L \iff M(x) = \text{Accept}$$

This is because M ∈ P with input $x \in \Sigma^n$ built with formula $\Phi_x = \exists y_1 \dots y_m \ \psi_x$

x ∈ L leads to $\psi_x(y)$ = 1 by reconstructing *y*. M decides L in P.

Circuits are solved deterministically in polynomial time.

$$\therefore \quad \mathrm{P} = \mathrm{NP} \quad \blacksquare$$

EIGHTH PROOF:

Σ = finite alphabet.
L ⊆ Σ* = a language.

So,

$$L \in \mathrm{NP} \Rightarrow \exists \text{ polynomial-time verifier } V(x, y)$$

And,

$$x \in L \iff \exists y \in \Sigma^{\leq p(|x|)} \text{ such that } V(x,y) = 1$$

An interactive proof system based on a prover P that convinces a verifier V that is bound to polynomial space..

$$L \in \text{IP} \iff \exists V \in \text{P}$$

If $x \in$ L exists, a prover V accepts it. Then,

$$\text{IP} = \text{PSPACE}$$

Any problem solvable in polynomial time has an interactive proof. Therefore, for any NP problem, the prover sends the witness y and the verifier runs V (x,y) $\in$ P, requiring only one message, proving it with zero randomness in one round. Then,

$$\text{NP} \subseteq \text{IP}[1] \subseteq \text{IP} = \text{PSPACE}$$

In reverse simulation, when NP $\subseteq$ IP and IP = PSPACE, NP problems are solvable with structured witnesses. Witnesses are reconstructed deterministically using logic trees, constraint propagation and model checking, all in P based on structured recursion. The verifier simulates the interaction.

PSPACE $\subseteq$ P for problems using bounded depth interaction. Simulating the verifier interaction deterministically by constructing the space of prover messages (witnesses), evaluating verifier responses and accepting valid messages only.

A TM M′, M $\in$ P, parses $x \in \Sigma^*$, builds the verifier V, reconstructs the correct y using branching logic by p(|x|) and accepts if and only if V (x,y) = 1. Thus,

$$x \in L \iff M(x) = \text{Accept}$$

NP problems are solved in one round interactive proofs and simulated deterministically based on recursion using polynomial time. A witness y has length p(n), verifier evaluations are in P The prover message is built in recursive increments polynomially in |x|. Hence,

$$\text{P} = \text{NP} \quad \blacksquare$$

NINTH PROOF:

Σ = finite alphabet.
L $\subseteq \Sigma^*$ = a language.

A finite structure $\mathcal{A}_x$ over vocabulary τ encodes an input $x \in \Sigma^*$.

An EF game is played where the spoiler aims to prove that A/B structures are different and the duplicator aims to show A/B are logically the same. The game runs for k rounds. The duplicator's winning strategy in *k* rounds, then:

$$A \equiv_k B \Rightarrow A \text{ and } B \text{ satisfy the same } \text{FO}_k \text{ formulas}$$

Computational complexity is solved with first order definability. By building the bridge and pruning complexity in logic gates. So, first order (FO) logic, linear order and fixed point (FP) operators define P. Thus, NP matches second order logic by quantifying relations while remaining as definable structures.

In terms of strategy, L ∈ NP, where $x \in \Sigma^*$ maps to Ax. Yielding,

$$x \in L \iff A_x \models \phi$$

Simulating a first order fixed point construction that clearly defines it. So, L ∈ P.

EF game proves whether A/B are distinguished by first order formulas based on quantifier depth *k*. NP verifier accepts based on depth *k* structure.

EF game's depth yields decisions because the verifier computation and the structure description are polynomial. Therefore, $k = p(n)$ ends rounds and game. Hence, a Turing machine simulates the duplicator's winning strategy deterministically in polynomial time.

The decider is defined by TM $M' \in P$ inputting *x*, building structure A*x*, running an EF game simulator of depth *k*, comparing A*x* with structure B. Then, accepting the duplicator's win $\Rightarrow A_x \models \phi$ and simulating NP verifier using structural logic. NP problems are existential second order (ESO) definable. Therefore,

$$\therefore \quad \mathrm{P} = \mathrm{NP} \quad \blacksquare$$

TENTH PROOF:

Σ {0,1} = a binary alphabet.
L ⊆ Σ* = a language in NP.

Then,

$$x \in L \iff \exists y \in \Sigma^{\leq p(|x|)} \text{ such that } V(x,y) = 1$$

Polynomial time verifier V ∈ P.

V is encoded into a natural number #V ∈ N using Gödel numbering where states → numbers, transitions → tuples and symbols → integers. The encoding is well-defined and a Turing machine simulates V. (x,y) = pairing function Gödel or Cantor pair. C = ⟨#V, (x,y) ⟩ representing a full verifier input as the code. Thus,

$$V(x,y) = 1 \iff U(C) = 1$$

U = a universal Turing machine.

Assign each Turing machine state, input symbol and transition, a unique prime number by encoding them in a finite sequence:

$$s = s_1, s_2, \ldots, s_n \text{ and } \#s = 2^{s_1} \cdot 3^{s_2} \cdots p_n^{s_n}$$

#V: encoding verifier logic #*x*: encoding input #*y*: encoding witness. So,

$$C = \langle \#V, \#x, \#y \rangle$$

Searching for minimal Gödel-witness by defining a deterministic strategy:

Input *x* computes #*x*. Search for $\#y \in \mathbb{N}$. Decoding y from $\#y$ evaluating $U(\langle \#V, \#x, \#y \rangle)$

$\#y$ $V(x,y) = 1$

Lex results in . Hence, eliminating nondeterminism by using arithmetic simulation.

Bitwise search by simulating $\#y$ bit by bit. Bit *i* = 0 and 1.

SinceV ∈ P, *y*'s total possible number is bounded by $2^{p(|x|)}$. Each *y* is a bitstring of length ≤ p(|x|). Therefore,

$$y = b_1 b_2 \dots b_k$$

A TM M′ ∈ P initialises *y* as an empty string. Each bit position is represented by $i \leq p(|x|)$. Hence,

$$b_i = 0 \text{ simulating } V(x, y')$$

If successful, keep 0; otherwise set Turing's machine to $b_i = 1$. Accept when y causes V (x,y) = 1.

All simulations are polynomial and the process simulates existential quantifiers deterministically, bit by bit.

Each bit decision runs a verifier V ∈ P. These are p (|x|) bits. So, total polynomial time:

$$p(|x|) \cdot q(|x|) = O(|x|^k) \qquad O(p(n)^2) = \text{poly}(n)$$

The TM M′ is described as:

$$M'(x) = \text{Accept} \iff \exists y,\ V(x, y) = 1 \Rightarrow M' \text{ decides } L \in \text{P}$$

Input = $x \in \Sigma^n$. Encode = $\#x, \#V$. So, $i \in [1, p(n)]$

L ∈ NP has a verifier V ∈ P and a witness $y \in \Sigma^{\leq p(|x|)}$.

$$\text{P} = \text{NP} \quad \blacksquare$$

ELEVENTH PROOF:

Σ {0,1} = a binary alphabet.
L ⊆ Σ* = a language in NP.

Therefore,

$$L \in \text{NP} \Rightarrow \exists \text{ verifier } V(x, y) \in \text{P}, \text{ and } y \in \Sigma^{\leq p(|x|)} \text{ such that } V(x, y) = 1$$

$$x \in L \iff \exists y \text{ with } |y| \leq p(|x|) \text{ such that } V(x, y) = 1,\ V \in \text{P}$$

K (y) = Kolmogorov complexity of string *y*.

So, the length of the shortest P outputs y on a universal Turing machine:

$$K(y) \leq |P|, \quad \text{where } U(P) = y \qquad K(y) = \min\{|P| : U(P) = y\}$$

$V(x, y) = 1$ for y.

y is verifiable in polynomial time because of its syntactic structure in terms of path, factorisation and satisfying assignment; because of its short and compressed descriptive form. Hence,

$$K(y) \leq p(|x|) + c$$

C = polynomial bound p(n).

The strategy is the enumeration of the program simulation. Thus,

$$P \text{ of length} \leq p(|x|) + c \qquad P \in \{0, 1\}^{\leq p(n)+c}$$

Running U(P) → y and checking if V (x,y) = 1. Taking polynomial time per simulation. Candidate programs are polynomially bounded:

$$\#P \leq 2^{p(|x|)+c}$$

The program requires equivalence checking, hashing and lexicographic pruning.

P outputs a valid y with V (x,y) = 1 is computable. V ∈ P enables simulation time per program. Candidates are polynomial in size because of K (y). So, the witness y is reconstructed by enumerating programs to up to length p(| x |), outputting and verifying each y. Structure of V (x,y) = 1 allows pruning.

M ∈ P parses input x, simulates P programs' length, checks y = U(P) as in V (x,y) = 1 and accepts it.

NP problems have a valid y and the shortest solution is canonical encoding. No need for full enumeration because verifier feedback guides a compressed search space. NP witnesses are deterministically generated from low Kolmogorov complexity.

$$\therefore \quad \mathrm{P} = \mathrm{NP} \quad \blacksquare$$

TWELFTH PROOF:

L ∈ NP

$$x \in L \iff \exists y \text{ with } |y| \leq p(|x|), \text{ such that } V(x,y) = 1, \quad V \in \mathrm{P}$$

V is computable in P:

$$\exists y \; \phi(x,y)$$

Σ_1^b = Existential quantifier.

S_1^2 = a two sorted first order arithmetic reasoning, one for natural numbers and one for bitstrings, formalising computable functions in polynomial time. Thus, a function of P:

$$\text{There exists a term } t(x) \text{ and a formula } \phi(x,y) \in \Sigma_1^b \text{ such that } \phi(x, f(x))$$

S_1^2 enables the **constructibility** of a witness y ∈ Σ* in P. Because existential proof = NP verifier accepts x. So,

$$\text{If } S_1^2 \vdash \exists y \; \phi(x,y), \text{ then there exists } f(x) \in \mathrm{FP} \text{ such that } \phi(x, f(x)) \text{ holds.}$$

This is known as a witnessing theorem in bounded arithmetic. L ∈ NP:

$$x \in L \iff \exists y \; \phi(x,y) \qquad x \in L \iff \phi(x, f(x)) = 1 \Rightarrow L \in \mathrm{P}$$

Then,

$$\exists f \in \mathrm{FP} \text{ such that } \phi(x, f(x)) \text{ holds} \Rightarrow x \in L \iff f(x) \text{ is a valid witness} \Rightarrow L \in \mathrm{P}$$

NP languages have a polynomial time decider. $f \in \mathrm{FP}$ and $\phi \in \mathrm{P}$.

M′ is a Turing machine in class P, M ∈ P:

y is a satisfying assignment of a Boolean circuit encoding V (x,y) bounded by a fixed polynomial in $|x|$.

Inputting *x,* computing $f(x) \in \Sigma^{\leq p(|x|)}$ and verifying $\phi(x, f(x))$ in polynomial time predicate V.

THIRTEENTH PROOF:

$$\mathrm{P} = \mathrm{NP} \quad \blacksquare$$

L ∈ NP

x ∈ Σ* = input.

$$x \in L \iff \exists y \text{ such that } V(x,y) = 1, \quad V \in \mathrm{P}$$

$$\psi := \exists y \, \phi_x(y) \qquad \psi_1, \ldots, \psi_n \qquad \phi_x(y^*)$$

A sequence of propositional formulas concluding that y* is the valid witness.

Computing a Boolean formula:

$$x \in L \iff \phi_x \text{ is satisfiable}$$

ϕ_x = CNF form. If unsatisfiable, a proof resolution exists as a refutation tree of $\phi_x \vdash \bot$.

A proof π of $\phi \vdash \bot$ exists because the length $|\pi|$ is the number of lines or clauses in terms of size. Thus, finding an assignment (certificate) and verifying via syntactic traversal. Short proofs exist for NP problems:

$$\exists \pi \text{ such that } P \vdash \phi_x \text{ in } |\pi| \leq \mathrm{poly}(|x|)$$

Simulating the search for π by enumerating rules of the proof system, restricting proof length to poly ($|x|$), verifying whether a sequence constitutes a valid proof and accepting x ∈ L when a valid proof of ϕ_x is found.

Structured proof search results in satisfiability because of enumeration of steps and satisfying an assignment y* that matches ϕ_x Each step is verified in P.

The proof's size is polynomial, rules are checkable in P and the number of rule applications is polynomial, making the simulation deterministic and polynomial in time.

TM M ∈ P computes ϕ_x , simulates the proof system P, checks $\phi_x \in \mathrm{SAT}$ and accepts the proof exists.

$$x \in L \iff M(x) = \text{Accept} \Rightarrow L \in \mathrm{P} \quad \therefore \quad \mathrm{P} = \mathrm{NP} \quad \blacksquare$$

FOURTEENTH PROOF:

L ∈ NP

So,

$$x \in L \iff \exists y \text{ such that } V(x,y) = 1, \text{ where } V \in \mathrm{P}$$

$\phi(x,y)$ = a computable Boolean predicate encoding V (x,y) in polynomial time.

A proposition ϕ (x,y) is a type. A witness y is a program p of type ϕ. ϕ (x,y) is provable because a program p exists. Meaning,

$$p : \exists y \ . \ \phi(x, y)$$

The witness confirms a constructive proof. So,

$$\exists y \ . \ \phi(x, y) \text{ is provable} \Rightarrow \text{a program } f(x) \text{ returns } y \text{ such that } \phi(x, y)$$

ϕ ∈ P proves the witness y is computable in polynomial time. Therefore,

$$x \in L \iff \text{There exists } f \in \mathrm{P} \text{ such that } \phi(x, f(x)) = 1 \Rightarrow x \in \mathrm{P}$$

The verifier predicate:

$$\phi(x, y) := V(x, y) = 1$$

Hence,

$$\exists y : \text{String}. \ \phi(x, y)$$

When proposition is provable, a term program *t* exists:

$$t : \exists y. \ \phi(x, y)$$

t computes a valid y because ϕ (x,y) holds. Then,

$$\tau_x := \exists y. \phi(x, y)$$

The type system constructs a proof term $t_x \in \tau_x$

A program P proves the type takes input x, constructs a specific y and satisfies ϕ (x,y). Thus, the logic is constructive, it exists and runs in polynomial time.

TM M ∈ P simulates the program by inputting *x*, computing y := f(x), checking V (x,y) = 1 and accepting it.

$$x \in L \iff M(x) = \text{Accept} \Rightarrow L \in \mathrm{P}$$

ϕ ∈ P leads to f ∈ P.

$$\text{Total Time}(M) = \text{poly}(|x|)$$

$$\mathrm{P} = \mathrm{NP} \quad \blacksquare$$

FIFTEENTH PROOF:

x ∈ Σ* = a finite relational structure A*x* in terms of a fixed vocabulary *τ* encoded as a string.

L ⊆ Σ* = A language as a set of structures $\{A_x \mid x \in L\}$.

NP solved with existential second order (ESO) logic:

$$L \in \mathrm{NP} \iff \exists R_1, ..., R_k \ \varphi(R_1, ..., R_k, A_x)$$

R_i = Relations added to the structure.

φ = A first order formula.

$B \models \varphi$ denotes a structure satisfying φ.

Map elements of A to elements of B because A and B are two structures of the same vocabulary.

Homomorphism $h : A_x \to B$ is a function mapping elements of A*x* into B. Then,

$$x \in L \iff \exists B \models \varphi \text{ and homomorphism } h : A_x \to B$$
$$R^A(a_1, ..., a_n), \text{ then } R^B(h(a_1), ..., h(a_n))$$

This leads to a search through mappings from domain Ax to domain B because |Ax| = n and |B| = k.

There are k^n possible mappings and constraint structures, graphs and relational models are deterministic classes of structures checking P problems by pruning invalid mappings.

B ⊨φ is encoded with polynomial size relative to Ax, making the search over B polynomially bounded.

f(x) and Ax construct a witness model B and homomorphism h. Hence,

$$B \models \varphi, \quad h : A_x \to B$$

Thus,

$$f(x) \in \mathrm{P}$$

A Turing machine M ∈ P encodes input *x* into structure A*x*, constructs B ⊨ φ using fixed logic, verifies the homomorphism $h : A_x \to B$ and accepts *h*.

$$x \in L \iff M(x) = \text{Accept} \Rightarrow L \in \mathrm{P}$$
$$\therefore \quad \mathrm{P} = \mathrm{NP} \quad \blacksquare$$

SIXTEENTH PROOF:

$L \subseteq \{0,1\}^n$ = a language with a Boolean circuit C_n.

A Boolean circuit is an acyclic graph of logic gates (AND, OR, NOT) with input and output nodes Then,

$$x \in L \iff C_n(x) = 1$$

$$x_1, x_2, ..., x_n$$

Input nodes =

$\{C_n\}$ = circuit family where each has polynomial size in n and circuit per input size n.

Size (C) = number of gates. Depth (C) = longest path from input to output. AC^0 = Constant depth, polynomial size, unbounded fan-in AND/OR. NC^1 = Logarithmic depth, polynomial size, bounded fan-in.

P/poly = Non-uniform polynomial size circuits. P-uniform = Polynomial size circuits generated by a deterministic Turing machine.

L ∈ NP, a verifier V (x,y) ∈ P exists. So,

$$x \in L \iff \exists y \in \{0,1\}^{p(|x|)} \text{ with } V(x,y) = 1 \qquad V(x,y) \in \mathrm{P}, \text{ with } |y| \leq p(|x|)$$

C_{n+m} = a Boolean circuit taking inputs $(x,y) \in \{0,1\}^n \times \{0,1\}^m$ and outputs $V(x,y) = 1$

Constructing a circuit C_n for each $x \in \{0,1\}^n$ and candidate *y*. Thus,

$$C_n(x,y) = 1 \iff V(x,y) = 1$$

Since P ⊆ P-uniform NC^1, there is no need for brute force search.

Simulating witness construction satisfying y from circuit logic by setting yi = 0 and evaluating whether circuit C(x,y) = 1 is valid. If yes, keeping 0; otherwise setting yi =1. Thus, executing structured prefix pruning.

Simulating each bit y_i for both paths 0 and 1 and pruning invalid witness paths using the circuit's logic gates. Iinvalid branches are ruled out by structure, not search, making the witness y deterministic in polynomial time.

The witness circuit generates function $f(x)$ by building $C_n(x,y)$, using gate logic to deduce valid y, evaluating simulated short-circuit logic paths and accepting a valid y as in V (x,y) = 1.

TM M ∈ P, building C_n from $x \in \Sigma^n$ and using P-uniform generator to construct C_{n+m}

For each i = 1 to *m*, try yi = 0, pad with 0s. Then, partially evaluate C (x,y), fix 0 or 1 based on output logic, return full y and accept if C (x,y) = 1.

Steps = $m = p(n)$ and time $O(p(n)^2)$ → circuit depth and size are polynomial. Therefore, constructing a uniform Turing machine computable in P.

$$\mathrm{P} = \mathrm{NP} \quad \blacksquare$$

SEVENTEENTH PROOF:

An alternating Turing machine (ATM) combines deterministic (DTM) and nondeterministic (NTM). Thus,

∀ = a universal state accepts all child branches.

∃ = an existential state accepts any valid child branch.

By combining DTM ($\forall$) and NTM ($\exists$) an input is accepted only when a winning strategy from the root of the computation tree exists.

Any language accepted in polynomial time by an ATM is in PSPACE. Therefore, a PSPACE problem can be solved by an ATM.

$$\mathrm{AP} = \mathrm{PSPACE}$$

L $\in$ NP, where NP = existential game.

$$x \in L \iff \exists y \text{ such that } V(x, y) = 1$$

A two player game:

Prover = Player $\exists$ choosing witness y

Verifier = Player $\forall$ challenging bits of y and checking V.

$\exists$ state for each bit of y, leading to next config with either $y_i = 0$ or $y_i = 1$ with end state verifier V (x,y) = 1.

Tree depth = p(n), each node = choice of $y_i \in \{0,1\}$ so 2 branches per node. Leaf = verifier decision V (x,y) $\in$ P

Traverse tree depth first, at each node, pick left (0), check validity, If no, pick right (1), then move to next bit and construct full y that satisfies verifier. Since depth = p(n), each check is in P and polynomial time.

The game ends when the verifier can't reject any path → $x \in L$

Meaning, perfect information emerging from a finite tree. Thus, simulating a strategy where the game tree is evaluated polynomially. $\exists$'s winning strategy is computed deterministically. The ATM game simulation enables $\exists$ to choose bits of y while $\forall$ verifies logical validity. ATM game is simulated in polynomial space decision trees are traversed deterministically $\exists$'s winning strategy is extractable.

$$x \in L \Rightarrow \exists M \in \mathrm{P} \text{ such that } M(x) = \mathrm{Accept}$$

G_x = ATM's computation game for input x $\in \Sigma^*$. DTM M′ = Simulating tree traversal. DTM M $\in$ P extracts answer from game tree. No exponentiality because tree traversal is deterministic, incremental and pruned.

Constructing the decide with M′ by building the game tree G_x, simulating universal and existential transitions and accepting $\exists$'s winning strategy. Game semantics model NP verification of winning game. Strategy simulation means $\exists$'s winning path is deterministic computation.

$$\therefore \quad \mathrm{P} = \mathrm{NP} \quad \blacksquare$$

EIGHTEENTH PROOF:

Σ {0,1} = a finite binary alphabet

L $\subseteq \Sigma^*$ = a decision problem.

L ∈ P = a language where a deterministic Turing machine M′ exists in polynomial p(n). Thus,

$$\forall x \in \Sigma^*, \quad M(x) = \begin{cases} \text{Accept} & \text{if } x \in L \\ \text{Reject} & \text{if } x \notin L \end{cases} \quad \text{and} \quad \text{Time}_M(x) \leq p(|x|)$$

L ∈ NP = a language where a polynomial time verifier V (x,y) ∈ P exists in a polynomial p(n).

$$x \in L \iff \exists y \in \Sigma^{\leq p(|x|)} \text{ such that } V(x, y) = 1$$

L ∈ NP where a verifier V ∈ P exists polynomially bound as p(n) for witness size y. So,

$$x \in L \iff \exists y,\ |y| \leq p(|x|) \text{ and } V(x, y) = 1$$

Turing machine M′ enumerates all $y \in \Sigma^{\leq p(|x|)}$, runs V (x,y) ∈ P for each candidate and accepts y satisfies V (x,y) = 1.

The verifier V (x,y) ∈ P is deterministically represented by the Turing Machine's circuit with both polynomial size and depth of logical decisions. Implementing structured recursive search branching generates valid paths only toward a full y. Therefore, fixing $y_1 \in \{0,1\}$ and test if any valid completion exists, iterating recursively with fixed prefix and pruning branches that can't succeed. Meaning, each path is checked incrementally using V based on a depth limited search in a binary tree of height p(n). All steps are in P and the verifier's structure binds them polynomially.

The decider M′ ∈ P inputs x ∈ Σ* and constructs a valid y using bounded branching recursively and V (x,y) to verify each candidate in P in order to accept valid y once found.

Every L ∈ NP has a verifier V ∈ P. Meaning, a valid witness y is constructed using deterministic recursion.

$$L \in \mathrm{P} \Rightarrow \ \mathrm{P} = \mathrm{NP} \quad \blacksquare$$

NINETEENTH PROOF: THE COMMS GAME

Joey and Alan, two players, hold the parts of the input.

Joey = a candidate witness $y \in \Sigma^{p(n)}$

Alan = input $x \in \Sigma^n$

Together, Joey and Alan want to decide whether V (x,y) = 1 for a polynomial time verifier V ∈ P.

NP = a communication problem. Thus, the function:

$$f(x, y) = \begin{cases} 1 & \text{if } V(x, y) = 1 \\ 0 & \text{otherwise} \end{cases}$$

CC(*f*) = communication complexity, meaning, the number of bits Joey and Alan need to exchange, using any protocol, to compute *f* (x,y) as in $f \in \mathrm{P}_{\mathrm{comm}}$

Therefore, NP complete problem has low nondeterministic communication complexity, Why? because Joey sends y, How? because Alan verifies V (x,y) in polynomial time. So, deterministic protocols are modeled as decision trees because the tree depth is polynomial in n and the protocol runs in P using structure-aware pruning.

Recursive structure enables witness compression because the verifier circuit V(x,y) has bounded depth and locality. So, Joey doesn't need to send the entire witness y. Instead, Joey sends a compressed, deterministically constructible representation of y based on logic gate level traces and nodal access, summaries of satisfying clauses for each candidate, protocol responses and hashes such as #0000.

Hence, reducing CC(*f*) to *O*(log n^k), meaning, Joey allows Alan to reconstruct the accepting path by simulating Joey's part without receiving full y. Joey compresses y using only the parts relevant to Alan. Joey's structured protocols allow compressed transcripts to be understood by Alan. Meaning, the root node of constructed protocol tree has no comms. Only internal nodes have perfect information in terms of communication bits, bit by bit, for example, bit 10 of y is 0. So, leaves accept the terms of valid y witness. Fully decoded in P sequentially.

Simulating comms game so Alan comes to terms with reality and computes the sequence of verifier requests. Alan finally computes the bit index needed, bit by bit. So. Joey simulates sending the bit based on structured y. Thus, Alan reads the transcript, accepts it as truth and reconstructs it deterministically in poly time. No comms needed, only local deterministic simulation of Joey's role preceding Alan's.

$g(x) \in \mathrm{P}$ = a function defining the witness' deterministic construction, simulating the communication protocol, reconstructing the witness y and verifying V(x,y) = 1. Then,

$$x \in L \iff g(x) = y \text{ and } V(x, y) = 1 \Rightarrow L \in \mathrm{P}$$

DTM M $\in$ P enables an optimal protocol for computing *f*(x,y), the necessary bits exchanged in the tree and verifies V (x,y) = 1 using standard P algorithm. All branching paths are simulated deterministically in $O(\mathrm{poly}(n))$

$$\mathrm{P} = \mathrm{NP} \quad \blacksquare$$

TWENTIETH PROOF:

A monotone Boolean circuit is the basis of {AND, OR gates}. Rejecting the use of negation, meaning, no need for {NOT gates}. Therefore, inputs are literals $x_1, x_2, ..., x_n \in \{0, 1\}$. Inputs are binary variables, 0 or 1.

A function $f : \{0,1\}^n \to \{0,1\}$ is monotone if $x \leq x' \Rightarrow f(x) \leq f(x')$. Thus, flipping any input bit from 0 to 1 never flips output from 1 to 0.

Monotone circuits for CLIQUE require super polynomial size. This is why this does not apply to non-monotone general Boolean circuits where the same function is simulated more efficiently by allowing NOT gates.

NP problems are naturally monotone, meaning, adding more inputs does not change an answer from YES to NO, new inputs get pruned as the witness is already contained within the original set. Then,

$$\phi(G) = \exists S \subseteq V \ : \ |S| = k, \text{ such that } S \text{ satisfies a monotone property}$$

Monotone NP problems require super polynomial size circuits represented by a lower bound barrier different from upper bound deterministic circuits. So, negation is used only where needed based on bounded variables. This is why the verifier V (x,y) logically encodes the NP complete problem.

L ∈ NP exists because of a Boolean circuit C (x,y) of polynomial size based on C (x,y) = 1 ⇔ V (x,y) = 1.

The circuit C is constructed from the verifier based on structured recursion leading to witness extraction.

L ∈ NP and V (x,y) ∈ P, making general circuits efficient in polynomial size based on P-uniform Boolean circuit C (x,y), using non-monotone gates (AND, OR, NOT) and evaluating V (x,y). Hence, even if monotone circuits require exponential size, this does not constrain P time simulation with full gate logic.

General circuit simulates C (x,y) and constructs y bit by bit. Each bit yi, tries 0 and 1, chooses the value that keeps C (x,y) = 1, evaluates gates incrementally and avoids branches that cannot yield acceptance y witness. Then, pruning the search space, no need guessing and backtracking.

A TM M ∈ P inputs $x \in \Sigma^*$, constructs the circuit C (x,y) using verifier logic, initialises $y := \emptyset$.

Each bit y_i, tries $y_i = 0$, evaluates whether C (x,y) can output 1. If not, set $y_i = 1$. When full y is constructed, it runs V (x,y) and accepts verifier deterministically in polynomial time $y: p(n)$.

General non-monotone circuits simulate NP verifiers and witness bits are chosen deterministically by pruning the search space. Monotone hardness results don't apply.

A deterministic machine M′ uses gate logic and tracing constraints to find a valid input y, evaluates all AND/OR gate outputs without guessing and it extracts a satisfying assignment, witness y, deterministically in P. Monotone lower bounds show that restricting circuits makes computation incomplete; by allowing controlled use of negation and full circuit expressiveness, the verification process of NP is completed in poly time.

$$\therefore \quad \mathrm{P} = \mathrm{NP} \quad \blacksquare$$

TWENTY FIRST PROOF:

L ⊆ Σ* = all languages consisting of the class P/poly.

A deterministic Turing machine M ∈ P.

$a : \mathbb{N} \to \Sigma^{\leq p(n)}$ = an advice function, a polynomial P.

$$x \in L \iff M(x, a(|x|)) = \text{Accept}$$

Advice string a(n) depends on the input length only. Non-uniformity means that a(n) is arbitrary for each n. Advice is based on the input length, not the input itself.

L ∈ NP:

$$x \in L \iff \exists y \text{ such that } V(x, y) = 1, \text{ where } V \in \mathrm{P}$$

Defining an advice function:

$$a(n) = \text{“lex smallest valid } y \text{ for inputs of length } n\text{”}$$

Each n, $a(n)$ is a valid witness for $x \in \sum^n$ or an invalid string if no x ∈ L of such length exists.

$$x \in \sum^n \qquad n, a(n)$$

Reconstructing verifier from advice with a machine M ∈ P taking input and advice

The machine checks if $V(x, a(n)) = 1$ and accepts it. Therefore, $L' \in \mathrm{P/poly}$. So,

$$\mathrm{NP} \subseteq \mathrm{P/poly} \quad \mathrm{NP} \subseteq \mathrm{P}$$

x ∈ L ∈ NP eliminates the advice function with verifier V (x,y) ∈ P.

This is because an advice string $a(n) = y^*$ and the lex smallest witness of length ≤ p(n) exist.

So advice is constructed deterministically using recursive prefix fixing.

For each bit of y*, a chosen value keeps V (x,y) = 1 reachable so this process simulates advice generation by eliminating the need for non-uniformity.

For the verifier V ∈ P, an advice function $n, a(n)$ is not required. Instead, the simulated construction of $n, a(n)$ is executed by the Turing machine by enumerating all possible $y \in \Sigma^{\leq p(n)}$, pruning invalid branches that are not conductive to witness y and finding the valid y conductive to V (x,y) = 1. The simulation is in P. Advice is pruned as a result.

Turing machine M′ ∈ P inputs $x \in \Sigma^n$, initialises empty string y = ε; for *i* = 1 to m = p(n), tries yi = 0; if completion y satisfies V(x,y), keeps 0; else set yi = 1, verifies V(x,y), accepts when the verifier finds the truth, reconstructs the advice string dynamically, simulates the advice generation process, finds the lex smallest y as in V (x,y) = 1, accepts y as the truth, eliminates the non-uniformity of P/poly and converts it into uniform P.

$$\mathrm{P} = \mathrm{NP} \quad \blacksquare$$