A PROOF OF $P = NP$ USING INCOMPLETENESS OF ZFC

VICTOR PORTON

Abstract. My proof that $P = NP$. The text also contains useful “inverter” machinery. The proof does not present an efficient NP-complete algorithm, because it is generated as twice running “inverter” algorithm which is known to be hard.

The proof uses logic (incompleteness of ZFC), algorithms taking algorithms as input, inversion of bijections, reduction of SAT to another NP problem.

1. Introduction

In this article I use the word “proof” exclusively either to denote proofs in our formal systems or to denote the proof presented in this article. I do not use it as a synonym of “certificate”. (However, certificates used are proofs.)

2. How NP is defined

$NP$ problem and $NP$-complete problem are defined in the standard way.

Definition 1. NP algorithm is an atmost exponential-time algorithm that solves an NP problem.

Definition 2. NP-complete algorithm is an atmost exponential-time algorithm that solves an NP-complete problem.

3. Time complexity of modus ponens

Our machine will be with infinite random access memory and infinite word size.

First note that memory allocation is an $O(1)$ operation because we may never deallocate and just increase a pointer on each allocation.

Therefore adding an element (assuming we use dynamic types) at the beginning of a linked list is an $O(1)$ operation.

By predicates I will mean predicates in first-order predicate calculus. (Note that predicates may have free variables, I will denote them by the caligraphic font like $P$.)

Definition 3. Modus ponens resolution is an subroutine that takes a linked list (“proved predicates”) and two predicates $x$ and $y$ and returns true if some element of the arrays is $X \implies y$ where $X$ is a pattern matching $x$.

---

2Atmost exponential-time means EXPTIME or below complexity.
Definition 4. Modus ponens takes the same arguments (but now the linked list is modifiable) first invoking modus ponens resolution and then if yes, inserting $y$ at the end of the list.

Remark: The above operation is a part of the standard procedure of proof-checking in first-order predicate logic.

We can assume that running an algorithm for a time $t$ on an input of the size $s$ cannot produce a linked list longer than $s + t$.

Let’s calculate the time of pattern matching of predicate $x$ by a pattern $X$:

Here is the algorithm:

Let pattern $X$ is a tree of maximum $s$ nodes (free variables can be only terminal nodes).

1. Create a map $m$ (from free variables to bound variables) – $O(1)$.
2. Create a set $b$ (of bound variables in $X$) – $O(1)$. [TODO: Optimize by using distinct symbols for free and bound variables. (maybe wrong: a bound variable can become free or vice versa, if we add/remove $\forall$.)]
3. Traverse the pattern tree – $O(s)$,
4. during each step also add the node into $F$, if it’s a variable directly after a quantifier sign – each step is $O(s_F)$, where $s_F$ is the number of elements in $F$, therefore it’s $O(s^2_m)$.
5. during each step also “traverse in the same way” $x$ – each step is $O(1)$, therefore it’s still $O(n)$.
6. during each step also check (max $O(s_m)$ where $s_m$ is the $m$ number of elements) the map $m$ to know if the in $b$ – each step is $O(s_b)$, where $s_b$ is the number of elements in $b$, therefore it’s $O(s^2_m)$. If it’s in $m$ and isn’t equal (constant time) to the current node of $x$, return (constant time) false. If it isn’t add (max $O(s_m)$ where $s_m$ is the map size) it to the map.

So, pattern matching takes $O(s^2_m)$.

Let’s calculate the time of modus ponens for a list of the length $l$:

Obviously, it’s $l$ multiplied by the time of each pattern matching + constant. So, it’s $O(ls^2_m)$.

If the total current data size of our algorithm (calculated as the “number of elements”) is $u$, then $l + s_m \leq u$ and therefore $ls^2_m \leq u^2$. So running time of modus ponens is $O(u^2)$.

We will proclaim that running time of modus ponens is below $p_2u^2 + p_1u + p_0$.

4. Misc

Definition 5. $A|B$ will mean run algorithms $A$ and $B$ in parallel and return the result of the first one halted.
5. Inverting algorithms

Saying “algorithm”, I will mean an algorithm together with its “domain” (a set).

Definition 6. Let $\text{INV}(Y)$ for any algorithm $Y$ be the statement
\[(1) \quad \exists \text{algorithm } X' \forall Y \forall Z : (X(Y) = Z \Rightarrow X'(Z) = Y).\]

TODO: Is an algorithm invertible if it is not halting on its domain?

Definition 7. $X$ is decipherable means that
\[(2) \quad \exists \text{polynomial-time algorithm } X' : \forall Y \forall Z (X(Y) = Z \Rightarrow X'(Z) = Y).\]

Obvious 1. If $X$ is decipherable, then $\text{INV}(X)$.

Definition 8. Deciphering means $\text{INV}$ applied to a decipherable algorithm.

Definition 9. Decipherer is an algorithm for deciphering.

Definition 10. I will call NP-complete decipherer a pair of a set $G$ of bijective algorithms and a decipherer to which any deciphering is polynomially reducible.

6. Formalistics

Definition 11. Let $L_F$ for a formal system $F$ be a polynomial or exponential time algorithm mapping $(H, s)$ for an arbitrary provable statement $H$ and a number $s$ into the minimal proof (in $L$) of $H$ in some proofs order (such that shorter algorithms are before longer ones in this order), such that its length is not above $s$. If there is no such proof, then $L_F$ doesn’t halt.

Obvious 2. $L_F$ is a bijective algorithm.

Obvious 3. If $L_F(H, s)$ halts, then $L_F(H, t)$ halts and $L_F(H, s) = L_F(H, t)$ for every natural $t \geq s$.

By ZFC we will mean modified “standard” ZFC adding before each axiom $(\exists C : A \in C) \Rightarrow$. (So NBG (in Elliott Mendelson’s formulation) is an extension of our variant of ZFC.)

Definition 12. Let MY be ZFC with added:
- continuum hypothesis;
- “proper class” NGB axioms (in Elliott Mendelson’s formulation).

Definition 13.
\[(1) \quad R(X, s) \equiv (X, s, L_{\text{ZFC}}(\text{INV}(X), s)).\]
\[(2) \quad Q(X, s) \equiv (X, s, L_{\text{MY}}(\text{INV}(X) \land \neg \text{INV}(X), s) \text{ halts}, s) \text{ halts}).\]

Obvious 4. Both $R$ and $Q$ are decipherable bijections (on the input on which they halt).

Lemma 1. If we have already proved (in MY) that $\text{INV}(X)$ is not provable in ZFC then the run time of proving that $\neg L_{\text{ZFC}}(\text{INV}(X), s)$ halts is no more than $p_2u^2 + p_1u + p'_0$ for constant $p_0, p_1, p_2$. 
Proof. Run time proving that \( \text{INV}(X) \) not provable in ZFC + run time of proving \( \forall P : (P \text{ not provable in ZFC} \Rightarrow \neg L_{ZFC}(P, s) \text{ halts}) \) (constant) + run time of proving the corresponding modus ponens \( (\leq p_2u^2 + p_1u + p_0) \).

So run time increased not more than by \( p_2u^2 + p_1u + p_0' \).

\[\Box\]

Obvious 5. \( \forall s : R(X, s) \text{ halts} \iff \text{INV}(X) \text{ provable in ZFC} \).

Lemma 2.

\( \forall s : Q(X, s) \text{ halts} \iff (\text{INV}(X) \text{ provable in MY}) \land (\text{INV}(X) \text{ not provable in ZFC} \text{ provable in MY}) \).

Lemma 3. \( Q(X, s + p(u)) \) (here \( u = s + s_X \), \( s_X \) is the size of \( X \)) halts is implied by

\[L_{MY}(\text{INV}(X) \land \neg L_{ZFC}(\text{INV}(X), q(u)) \text{ halts}) \text{ for some quadratic polynomial } q \text{ less than } p.\]

Proof:

\( L_{MY}(\text{INV}(X) \land \neg L_{ZFC}(\text{INV}(X), q(u)) \text{ halts}) \text{ for some quadratic polynomial } q \text{ less than } p.\)
depends on \( w_0 \), thus it took at most \( p_2(u) \) steps for some quadratic polynomial \( p_2 \).

The sum of the number of additional steps is a quadratic polynomial. \( \square \)

**Definition 14.**

\[
R'(X, y) = (y, R((X, y + p(y)))); \quad Q'(X, y) = (y, Q(X, y + p(y))).
\]

**Obvious 6.** \( R' \) and \( Q' \) are bijection algorithms.

**Obvious 7.** \( R \) is a polynomial “oracle” for \( R' \) and \( Q \) is a polynomial “oracle” for \( Q' \).

**Obvious 8.** \( R'(X, s) \Rightarrow R(X, s), Q'(X, s) \Rightarrow Q(X, s) \).

**Lemma 4.** The bijections \( Q \) and \( Q' \), \( R \) and \( R' \) are reducible to each other in polynomial time.

**Proof.** Obvious. \( \square \)

**7. On NP**

Having an NP problem \( f \), it’s easy to conceive the polynomial or exponential provable bijection \( T(f)X = (f(X), X, v_f(X)) \) with the domain \( \{X | f(X) \text{ halts}\} \) (where \( v_f(X) \) is a certificate for \( f(X) \)) with obviously polynomial-time inverse.

**Proposition 1.** An NP-complete decipher allows to calculate an NP problem polynomially.

**Proof.** Having an NP-complete decipher, we can calculate \( T(f)X \) and thus \( (f(X), X, v_f(X)) \) polynomially. So, we can calculate \( f(X) \) polynomially. \( \square \)

**Proposition 2.** If \( f, g \) are NP-complete problems, then

\[
T(f, g)X = (T(f)X, T(g)X)
\]

is an NP-complete decipherer on \( \text{dom } f \cup \text{dom } g \).

**Proof.** \( T(f, g) \) is obviously a bijective algorithm.

It remains to prove that \( T(f, g) \) has a polynomial inverse. Really, let

\[
Z = T(f, g)X = (T(f)X, T(g)X) = ((f(X), X, v_f(X)), (g(X), X, v_g(X))).
\]

So, knowing \( Z \) we can easily reconstruct \( f(X) \) and \( v_f(X) \) and therefore obtain \( X \) by a polynomial-time algorithm if \( X \in \text{dom } f \). Similarly, we can obtain \( X \) by a polynomial-time algorithm if \( X \in \text{dom } g \). So, \( X \) can be reconstructed in polynomial time by running these two algorithms in parallel. \( \square \)

**Proposition 3.** There is a polynomial-time decipherer for every polynomial-time bijection.
Proof. Let $c$ be a polynomial-time bijection. Let’s find it’s polynomial inverse.

Knowing the deciphering $d(y)$ result $x$, we can verify it by checking $c(x)_i = y_i$ that is by that is verifying the halting polynomial-time (and therefore in NP) algorithms $(x, y_i) \mapsto c(x)_i = y_i$ (because the size of $c(x)$ is polynomial or exponential) with the certificate $(x, y_i)$. So, it (calculating each $x_i$) can be done by an two NP-complete algorithms $f$ and $g$ (for false and true) running in parallel, therefore $c$ has a polynomial-time inverse. 

8. Main Lemmas

**Lemma 5.** “$Q'(X, s)$ halts for arbitrary number $s$ and $INV(X)$ not provable in ZFC provable in MY, but provable in MY” is an NP-complete problem.

Proof. Consider two infinite sets of statements not provable in ZFC provable in MY and statements the union of these sets independent of each other, first one provable and the second one disprovable in MY extending ZFC. Non-provability of these statements becomes provable in MY (use Easton’s theorem, using proper classes from NBG). For a SAT problem $S$ (that is not always false) of $k$ variables and arbitrary $k$ boolean values, we can therefore produce a theorem $\exists \ldots U(\ldots)$ (with free variables) whose meaning is “$S$ may be true” such that is provable in MY in $N$ steps. So, proving such formulas in MY is an NP-complete problem as it allows to reduce SAT to proving it: As the answer of SAT we use is true if we can prove $U$ in MY, it can be true only if the formula is satisfied, because otherwise we would have no way to prove it in ZFC. (To prove NP-completeness, find (polynomial) complexity $P(N(k))$ suitable (it exists because SAT can be verified by a polynomial verifier) for this proof.)

**Lemma 6.** $R'(X, s)$ halts is an NP-complete problem for all halting algorithms $X$ and all numbers $s$ such that either $R'(X, s)$ halts $\lor Q'(X, s)$ halts.

Proof. $R(X, s)$ is NP-complete for halting algorithms $X$ because $X'$ is already found above because we assume $R(X, s)$ halts and $Q'$ (and therefore $Q$) halts on all $(X, s)$ for which $INV(Y)$ is not provable in ZFC provable in MY. So $INV(Y)$ is false for such $Y$ (for a proof enumerate all algorithms $X'$ and all values $Y, Z$).

Therefore $R'(X, s)$ is NP-complete for halting algorithms $X$.

**Lemma 7.** $Q'(X, s)$ halts is an NP-complete problem for all halting decision algorithms $X$ and numbers $s$ such that either $R'(X, s)$ halts $\lor Q'(X, s)$ halts.

Proof. $\{Q'(X, s) \mid X, s\}$ is a superset of the set of statements for which $INV(X)$ not provable in ZFC provable in MY, but provable in MY.

We will prove that lemma 5 NP-complete problem is reducible to our problem $Q'(X, s)$.

If $R'(X, s)$ halts, then lemma 5 NP-complete problem is false.

Therefore, we need to reduce only in the case if $Q'(X, s)$ halts.
Let $Q'(X, s)$ halts. Then lemma 5 NP-complete problem is reduced to $Q'(X, s)$ halts.
So, lemma 5 NP-complete problem is reduced to $Q'(X, s)$. □

**Lemma 8.** $R'|Q'$ is an NP-complete decipherer on its domain.

*Proof.* $R'|Q'$ is at most exponential-time because the sizes of outputs of both $R'(X, s)$ and $Q'(X, s)$ “potentials” variants of output sizes are bounded by $2^s$.
Both $R'$ halts or $Q'$ halts on such inputs and they are NP-complete algorithms on their domains. So, by 2 $R'|Q'$ is an NP decipherer for such inputs. □

9. **Main results**

**Theorem 1.** If $D$ is an atmost exponential-time NP decipherer then $D$ can be equivalently replaced by a polynomial-time algorithm.

*Proof.* By definition of an NP decipherer, there is a polynomial inverse $I$ of $D$. $I$ is decipherable because $D$ is a bijection.
So, $I$ is an NP-bijection. Therefore exists it polynomial inverse $I'$.
But $I'$ is a twice inverse of $D$, so produces the same values. □

**Theorem 2.** There is a polynomial-time NP-complete decipherer.

*Proof.* Let $I''$ be a polynomial-time algorithm for the NP-complete deciphering $R'|Q'$ from the lemma.
$R'|Q'$ is an atmost exponential-time NP-complete decipherer for all (an exponential algorithm for $R|Q$ is trivially created by enumerating all proofs, but $R'|Q'$ is polynomially reducible to it) bijective algorithms $X$ and numbers $s$ such that either $R'$ halts or $Q'$ halts. □

**Corollary 1.** There is a polynomial-time NP-complete algorithm.

*Proof.* The above considered polynomial-time algorithm $I''$ on $\{X \mid R'|Q' \text{ halts} \}$ is an NP-complete algorithm. □

**Corollary 2.** $P = NP$.

**Conjecture 1.** It is still open, but probably easy after publication of this article, question to prove (or disprove) that no one-way function exists.

**References**

Forum r/algorithms

[2] Porton, Victor. “If $P=NP$, then I have an NP-complete verifier (second proof attempt).” Reddit. April 6, 2021. Accessed April 07, 2021. [https://www.reddit.com/r/algorithms/comments/ml85dv/it_seems_i_proved_pnp_i_have_an_npcomplete/](https://www.reddit.com/r/algorithms/comments/ml85dv/it_seems_i_proved_pnp_i_have_an_npcomplete/).
Forum r/algorithms


[5] TODO

Email address: porton@narod.ru