A New Proof of Inequality of P-NP

Angelo Raffaele Meo*

Department of Control and Computer Science, Polytechnic University of Turin, Academy of Sciences of Turi, Turin, Italy

Theory

Received: 02-Apr-2026, Manuscript No. GRCS-26-187502; **Editor assigned:** 03-Apr-2026, PreQC No. GRCS-26-187502 (PQ); **Reviewed:** 17-Apr-2026, QC No. GRCS-26-187502; **Revised:** 24-Apr-2026, Manuscript No. GRCS-26-187502 (R); **Published:** 30-Apr-2026, DOI: 10.54615/2231-7805.17.1.003

***For Correspondence:**
Angelo Raffaele Meo, Department of Control and Computer Science, Polytechnic University of Turin; Academy of Sciences of Turi, Turin, Italy

E-mail: angelo.meo@polito.it

Citation: Meo AF. A New Proof of Inequality of P-NP. J Glob Res Comput Sci. 2026;17:003.

Copyright: © 2026 Meo AF.
This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution and reproduction in any medium, provided the original author and source are credited.

INTRODUCTION

The author of this paper presented a first proof of the inequality of the classes of decision problems P and NP in three papers presented to the Academy of Sciences of Turin in 2016 and to the Journal of Computer Science in 2020 and 2022. According to the Journal of Computer Science more than 10000 scientists have read papers and more than 3000 readers have downloaded those papers [1-3].

Two years ago he wrote a new paper. presenting a new version of the first proof, the answers to the questions asked by some readers and the proofs of some theorems which had been omitted for the sake of brevity in the preceding papers. That paper was published by the Journal of Global Research on Computer Science [4]. That paper was copied and became Chapter 2 of the book "Mathematics and Computer Science Contemporary Developments" published by BP International [5].

This paper presents a third proof of the inequality of P-NP, which is absolutely new. It is based on the theory of np-completeness which makes it possible to reduce the length of the proof by many pages.

The starting point. NP-Completeness

P denotes the class of all the decision problems (that is, the problems which have a binary solution TRUE or FALSE) which can be solved in polynomial time.

NP denotes the class of all the decision problems satisfying the property that the function check(f) analyzing a witness of the decision problem is polynomial time decidable.

"P=NP?", or, in other terms, "Is P a proper subset of NP?", is one of the most important open questions in computational complexity theory. It is the question discussed in this paper.

According to the theory of np-completeness (which can be analyzed in detail on “NP – Completeness Wikipedia”), a decision problem C in NP is NP-complete if it is in NP and if every other problem L in NP is reducible to it, in the sense that there is a polynomial time algorithm which transforms all and only the instances of L into instances of C producing the same output values [6-16].

The importance of NP-completeness derives from the fact that, if we find a polynomial time algorithm for just one NP-complete problem, then we can construct polynomial time algorithms for all the problems in NP and, conversely, if any single NP-complete problem does not have a polynomial time algorithm, then no NP-complete problem has a polynomial time solution.

The analysis discussed in this paper will be based on the following well-known NP- complete problem which is called “satisfiability problem” or “SAT”.

Given a Boolean expression containing the names of variables (some of which may be complemented), the operators AND, OR and NOT, and parentheses, is there an assignment of TRUE or FALSE values to the variables which makes the entire expression TRUE?

It is well known that the problem remains NP-complete also when all the expressions are written in “conjunctive normal form” with 3 variables per clause (problem 3SAT). In this case, the analyzed expressions will be of the type:

$$
\begin{aligned}
&3SAT(t) = \\
&(x_{11}\ OR\ x_{12}\ OR\ x_{13})\ AND \\
&(x_{21}\ OR\ x_{22}\ OR\ x_{23})\ AND \\
&\ldots\ldots\ldots\ldots \\
&(x_{t1}\ OR\ x_{t2}\ OR\ x_{t3})
\end{aligned}
\qquad \text{Eq. (1)}
$$

where:

t is the number of clauses or triplets;

every literal x_{ij} is a variable in complemented or uncomplemented form; every variable may appear multiple times in that expression.

The synthesis of the state of art of question PvsNP can be found [6,7].

The computation of satisfiability problem described by Eq. (1) can be decomposed into two processing layers called “compatibility layer” and “core layer”.

Compatibility layer

A variable j of triplet i will be defined as “compatible” with variable k of triplet h when, and only when, either

- the sign s_{ij} of the former variable is equal to the sign s_{hk} of the latter variable,

or

- the binary code $<n_{ij1}\ n_{ij2}\ \ldots\ n_{ijm}>$ of the name of the former variable is different from the binary code $<n_{hk1}\ n_{hk2}\ \ldots n_{hkm}>$ of the name of the latter variable.

From that definition it follows that two “not compatible” variables have different signs and the same name; therefore, there AND is identically FALSE.

The compatibility layer is composed of $3 \cdot t \cdot (3 \cdot t - 3)/2 = 9 \cdot t \cdot (t-1)/2$ identical operations, one for each pair of variables belonging to different triplets.

The input variables of one of these operations will be the sign s_{ij} and the binary code $<n_{ij1}\ n_{ij2}\ ...n_{ijm}>$ of the name of variable j of triplet i, and the sign s_{hk} and the binary code $<n_{hk1}\ n_{hk2}\ ...n_{hkm}>$ of the name of variable k of triplet h. The output variable of this operation c(i,j;h,k) will be TRUE when, and only when, the two input variables are mutually compatible.

Therefore, the function implemented by one of the operations of compatibility layer may be written as follows (by using the symbols $*$, +, and! for representing AND, OR and NOT operators, respectively):

$$\begin{aligned} c(i,j;h,k) &= s_{ij}*s_{hk}+!s_{ij}*!s_{hk}+ \\ &+ n_{ij1}*!n_{hk1}+!n_{ij1}*n_{hk1}+ \\ &+ n_{ij2}*!n_{hk2}+!n_{ij2}*n_{hk2}+ \\ &\quad \ldots\ldots\ldots\ldots\ldots\ldots \\ &+ n_{ijm}*!n_{hkm}+!n_{ijm}*n_{hkm} \end{aligned} \qquad \text{Eq. (2)}$$

The value c(i,j;h,k) described by Eq. (2) will be called a “compatibility variable” or simply a “compatibility”.

Core layer

The core layer processes only the $9\cdot t\cdot(t-1)/2$ compatibility variables c(i,j;h,k) and produces the global result of computation. The Boolean function implemented by the core layer will be called the “Core Function” of order t, where t is the number of triplets. It will be denoted with the symbol CF(t) (or CF(n)). (Indeed, the number t of triplets appearing in Eq. (1) plays the role of symbol n used in the standard complexity theory. In the following analysis, we shall use the symbol t when it is necessary to remember the number of triplets and n in the other cases). The Core Function can be computed by proceeding as follows.

Consider one selection of variables xij appearing in Eq. (1), one and only one for each triplet, for all the triplets. Let

$$<1\ i_1>, <2\ i_2>, ...,<t\ i_t> \qquad \text{Eq. (3)}$$

with $i_1, i_2, ..., i_t \in \{1, 2, 3\}$

be the indexes <number of triplet, number of the variable in the triplet> of a selection.

As an example, consider the following satisfiability equation

3SAT(3) =

(x_{11} OR x_{12} OR x_{13}) AND

(x_{21} OR x_{22} OR x_{23}) AND

(x_{31} OR x_{32} OR x_3)

The considered selection may be one of the following 27 selections.

Sel1 x_{11} x_{21} x_{31}

Sel2 x_{11} x_{21} x_{32}

Sel3 x_{11} x_{21} x_{33}

Sel4 x_{11} x_{22} x_{31}

Sel5 x_{11} x_{22} x_{32}

……

Sel27 x_{13} x_{23} x_{33}

Consider the product of all the compatibility variables relative to the k-th selection:

π_k=c(1 i1;2,i2)*c(1 i1;3,i3) * ... Eq. (4)

For example, in the case of the above defined 3SAT(27) and Sel 5

π_5=c(1,1;2,2)*c(1,1;3,2)*c(2,2;3,2)

The Core Function can be defined as the sum

$\Sigma_k \pi_k$ Eq. (5)

of all the products as the one described by Eq. (4) relative to all the selections specified by Eq. (3).

For example, in the case of CF(3) and CF(4), the Core Function can be defined as follows:

CF(3)= Eq. (6)

c(1,1;2,1)∗c(1,1;3,1)∗c(2,1;3,1)+
c(1,1;2,1)∗c(1,1;3,2)∗c(2,1;3,2)+
c(1,1;2,1)∗c(1,1;3,3)∗c(2,1;3,3)+
c(1,1;2,2)∗c(1,1;3,1)∗c(2,2;3,1)+
... (other 22 products) ... +
c(1,3;2,3)∗c(1,3;3,3)∗c(2,3;3,3)

CF(4)= Eq. (7)

c(1,1;2,1)*c(1,1;3,1)*c(1,1;4,1)*c(2,1;3,1)*c(2,1;4,1)*c(3,1;4,1)+
c(1,1;2,1)*c(1,1;3,1)*c(1,1;4,2)*c(2,1;3,1)*c(2,1;4,2)*c(3,1;4,2)+
c(1,1;2,1)*c(1,1;3,1)*c(1,1;4,3)*c(2,1;3,1)*c(2,1;4,3)*c(3,1;4,3)+
c(1,1;2,1)*c(1,1;3,2)*c(1,1;4,1)*c(2,1;3,2)*c(2,1;4,1)*c(3,2;4,1)+
... (other 76 products) ... +
c(1,3;2,3)*c(1,3;3,3)*c(1,3;4,3)*c(2,3;3,3)*c(2,3;4,3)*c(3,3;4,3)

It is easy to prove that there is an assignment of values TRUE or FALSE to variables appearing in Eq. (1) which makes the value of that equation equal to TRUE when, and only when, the Core Function takes the value TRUE.

Indeed, for example, if 3SAT(3) is TRUE by virtue of the following equations :

x11=NOT(v1)
x22= 2
x32=NOT(v3)

and

v1=FALSE
v2=TRUE
v3=FALSE

(where v1, v2, v3 are names of Boolean variables), then

c(1,1;2,2)*c(1,1;3,2)*c(2,2;3,2)=TRUE

and, therefore,

CF(3)=TRUE

The reverse is nearly obvious. Assume that CF(3)=TRUE because c(1,1;2,2)*c(1,1;3,2)*c(2,2;3,2)=TRUE. It

follows that variables x11, x22, x32 are compatible with each other and, therefore, it is sufficient to assign the right values TRUE or FALSE to the Boolean variables appearing in their definition to obtain that all the variables x11, x22, x32 are equal to TRUE and also satisfiability equation is equal to TRUE.

Notice that the processing work of the elementary operations of compatibility layer (Eq. (2)) increases as a logarithmic function P(t) of the number of the variables since the increment of the length of the code of a name is logarithmic. Therefore, the total processing work of the compatibility layer increases polynomially as 9·t·(t−1)·P(t)/2 where 9·t·(t−1)/2 is the total number of compatibility operations.

Besides, the problem solved by the core layer is clearly in NP, because it is easy to verify a witness solution. It follows that, since the compatibility layer polynomially reduces an NP- complete problem (3SAT) to the problem solved by the core layer, the Core Function describes a new NP-complete problem.

Some properties of Core Function have been discussed [9].

Notice that every product of compatibilities of CF(3) or CF(4) appearing in Eq. (6) or Eq. (7) is a prime implicant of the above described Boolean functions. A prime implicant of a Boolean function is a product of variables characterized by the following properties: a) it implies the considered Boolean function; b) the product of any subset of its variables does not imply the Boolean function. Prime implicants play a basic role in the design of Boolean functions as is shown, for example, by the well-known [13].

Completeley specified core function

The Boolean function implemented by the core layer, that is, the Boolean function that has been called Core Function CF(n) (or CF(t)), can be viewed as an incompletely specified function.

Indeed, assume that

c(i,j;l,m)=0

and

c(i,j;p,q)=0

This implies that variable <i,j> and variable <l,m> have the same name and a different sign; similarly, <i,j> and <p,q> have the same name and a different sign. It follows that <l,m> and <p,q> have the same name and the same sign. Therefore, c(l,m;p,q) cannot be equal to 0.

Therefore, all the minterms implying

!c(i,j;l,m) * !c(i,j;p,q) * !c(l,m;p,q)

are incomplete specifications of the Boolean function implemented by the core layer. (An incompletey specified function is characterized by a third type of minterms in addition to the two types of minterms defined as TRUE or FALSE. The minterms of this third type are defined as FREE. In the synthesis process a minterm of the type FREE can be freely used as TRUE or FALSE depending on the needs of the design).

It is easy to transform the incompletely specified Core Function CF(n) into a corresponding completely specified Core Function CSCF(n). This result can be obtained by assuming CSCF(n)=0 for all the minterms of CF(n) which are incompletely specified [1].

The result of Meo AR, et al., is the sum of many products of “positive” (or not complemented) compatibilities (PoPC). In this sum a PoPC A may imply a PoPC B; if this occurs, A can be deleted. All the remaining PoPC’s are prime

implicants of CSCF(3).

This result can be generalised. All the prime implicants of CF(n) and of CSCF(n) are PoPC's. Besides, all these prime implicants are "essential", that is, none of them is implied by the sum of other prime implicants.

Both the incompletely specified function CF(n) and the completely specified function CSCF(n) are NP-complete problems.

In order to prove that the class P is not equal to class NP, it is not necessary to show that both CF(n) and CSCF(n) describe NP-complete problems. Therefore, for the sake of brevity, attention will be restricted in this paper to the properties of CF(n).

Marks and remainders

Definition of mark

Consider a Product of Compatibilities (PoC) satisfying the property that all the indexes of triplet {1,2,...,t} appear at least once in some uncomplememted variable. The product of the variables of such a subset will defined as a "mark" of the prime implicant of which it contains a subset of compatibilities.

For example, in the case of CF(4), the PoC

$$M=c(1,a;2,b)*c(1,a;3,c)*c(1,a;4,d) \qquad \text{Eq. (8)}$$

(where the indexes of triplet are elements of the set {1,2,3,4} and a, b, c, d are elements of {1,2,3}) is a mark of the prime implicant.

$$P=c(1,a;2,b)*c(1,a;3,c)*c(1,a;4,d)*c(2,b;3,c)*c(2,b;4,d)*c(3,c;4,d)$$

since all the indexes of triplet appear at least once in Eq. (8).

Definition of remainder

A PoC which is not a mark will be called a "remainder". A remainder may be implied by more than one prime implicant. For example,

$$R=c(2,1;3,1) \text{ of CF(3)}$$

is implied by the following prime implicants

$$P1=c(1,1;2,1)*c(1,1;3,1)*c(2,1;3,1)$$
$$P2=c(1,2;2,1)*c(1,2;3,1)*c(2,1;3,1)$$
$$P3=c(1,3;2,1)*c(1,3;3,1)*c(2,1;3,1) \qquad \text{Eq. (9)}$$

A mark M can be defined as a "strong mark" if there is a PoC R such that M*R is a prime implicant of Core Function. For example, the strong mark c(1,1;2,1)*c(1,1;3,1)*c(1,1;4,1), multiplied by the remainder c(2,1:3,1)*c(2,1;4,1)*c(3,1;4,1), becomes a prime implicant of CF(4).

If a mark cannot be defined as a strong matk, it will be called "a weak mark". For example, the weak mark c(1,1;2,1)*c(1,1;3,1)*c(2,1;4,1), must be multiplied by the weak mark c(1,1:4,1)*c(2,1;3,1)*c(3,1;4,1) in order to produce a prime implicant of CF(4).

On the definitions of mark and remainder the following properties are based.

PROPERTY 4.1

Let P_1 and P_2 be two PoC's such that P_1*P_2 is equal to a prime implicant P of Core Function. It is easy to prove that either P_1 or P_2 is a mark of P (or both of them are marks).

PROPERTY 4.2

Theoretically, the product of a mark M_1 of the prime implicant I_1 by a mark M_2 of the prime implicant P_2 may be equal to I_1*I_2. For example, if

M_1=c(1,1;2,1)*c(1,1;3,1)*c(1,1;4,1)*c(2,1;3,1)

and

M_2=c(1,2;2,1)*c(1,2;3,1)*c(1,2;4,1)*c(2,1;4,1)*c(3,1;4,1)

then

$M_1*M_2= I_1*I_2$

However, it is easy to verify that M_1*M_2 is implied by (1/8) of the minterms of I_1 and by (1/8) of the minterms of I_2.

In general terms, it is easy to prove for CF(n) that, If M1 is a mark of prime implicant I_1 and M_2 is a mark of prime implicant I_2, M_1*M_2 can be implied by (1/2n-1) of the minterms of I_1 and by (1/2n-1) of the minterms of I_2.

The mark M_1 of prime implicant I_1, multiplied by a mark M_2 of prime implicant I_2, can produce a single prime implicant I_1. This prime implicant I_1 is spurious for at least one compatibility.

For example, the mark of CF(4) M_1=c(1,1;2,1)*c(1,1;3,1)*c(1,1;4,1) of prime implicant $I_1=M_1$*c(2,1;3,1)*c(2,1;4,1)*c(3,1;4,1), multiplied by mark M_2=c(1,2;2,1)*c(2,1;,3,1)*c(2,1;4,1)*c(3,1;4,1), produces prime implicant I=c(1,1;2,1)*c(1,1;3,1)*c(1,1;4,1)*c(1,2;2,1)*c(2,1;3,1)*c(2,1;4,1)*c(3,1;4,1), which is spurious for compatibility c(1,2;2,1).

Obviously, as already stated, the product of a weak mark M_1 of prime implicant I_1 by a weak mark M_2 of I_1 can be equal to I_1.

PROPERTY 4.2 makes reference to borderline examples. It is extended by the following analysis.

PROPERTY 4.3

The product of two marks M_1 and M_2 which are implied by two different prime implicants I_1 and I_2 of Core Function does not imply Core Function and it is not implied by Core Function. The proof of this PROPERTY is presented in Appendix 3.

PROPERTY 4.4

The best way to integrate operations OR and AND is suggested by the ability of a remainder R to produce 3m prime implicants (where m is the number of triplet indexes missing in R), through the multiplication by a suitable OoM ("Or of Marks"). For example, the remainder R=c(2,1;3,1) of CF(4), multiplied by nine different strong marks, can be used to obtain the following prime implicants of CF(4):

R*OoM = Eq. (10)

c(1,1;2,1)*c(1,1;3,1)*c(1,1;4,1)*c(2,1;3,1)*c(2,1;4,1)*c(3,1;4,1) +

c(1,2;2,1)*c(1,2;3,1)*c(1,2;4,1)*c(2,1;3,1)*c(2,1;4,1)*c(3,1;4,1) +
c(1,3;2,1)*c(1,3;3,1)*c(1,3;4,1)*c(2,1;3,1)*c(2,1;4,1)*c(3,1;4,1) +
c(1,1;2,1)*c(1,1;3,1)*c(1,1;4,2)*c(2,1;3,1)*c(2,1;4,2)*c(3,1;4,2) +
c(1,2;2,1)*c(1,2;3,1)*c(1,2;4,2)*c(2,1;3,1)*c(2,1;4,2)*c(3,1;4,2) +
c(1,3;2,1)*c(1,3;3,1)*c(1,3;4,2)*c(2,1;3,1)*c(2,1;4,2)*c(3,1;4,2) +
c(1,1;2,1)*c(1,1;3,1)*c(1,1;4,3)*c(2,1;3,1)*c(2,1;4,3)*c(3,1;4,3) +
c(1,2;2,1)*c(1,2;3,1)*c(1,2;4,3)*c(2,1;3,1)*c(2,1;4,3)*c(3,1;4,3) +
c(1,3;2,1)*c(1,3;3,1)*c(1,3;4,3)*c(2,1;3,1)*c(2,1;4,3)*c(3,1;4,3)

The last gate of CF(n)

For the sake of simplicity, we merely consider the case of CF(n), but the same conclusions also apply to CSCF(n). Assume that the last gate of the circuit implementing Core Function is an AND gate.

Some inputs of this gate may be the outputs of other AND gates and also the inputs of these gates may be the outputs of other AND gates. The subnetwork of all these AND gates is equivalent to a single AND gate whose inputs are the outputs of OR gates.

Since the output of this AND gate is CF and its i-th input must be equal to (CF + a_i), the output of the gate will be

$$((CF+a_1)*(CF+a_2)* \ldots*(CF+ai)*\ldots)=(CF+a_1*a_2* \ldots *a_i \ldots)$$

where ($a_1*a_2*\ldots a_i\ldots$) is equal to 0

The cost - that is, the number of AND and OR gates - of the subnetwork producing (CF + a_i) may be less than the cost of the subnetwork producing CF because of the reduction of the number of prime implicants, but the cost of another (CF + a_j) may be equal to or larger than the cost of CF because some complemented variables appearing in a_j increment the number of prime implicants of CF. Therefore, it is not useful to use an AND gate as the last gate of the implementation of Core Function, as shown in Appendix 1.

Therefore, the last gate of the network implementing Core Function must be an OR gate. Some inputs of this gate may be the outputs of other OR gates and also the inputs of these gates may be the outputs of other OR gates. The subnetwork of all the OR gates is equivalent to a single OR gate whose inputs are the outputs of AND gates.

An input Ij of the last OR gate (coinciding with the output of an AND gate) may be: A prime implicant of Core Function (Hp 1);

a PoC implying a prime implicant of Core Function (Hp 2);
a “Remainder And (Or of Marks)”, or “RAOM” (Eq. (10) (Hp 3).
For example, in the case of CF(4), the input of the last AND gate may be:
the prime implicant c(1,1;2,1)*c(1,1;3,1)*c(1,1;4,1)*c(2,1;3,1)*c(2,1;4,1)*c(3,1;4,1); or
the PoC c(1,1;2,1)*c(1,1;3,1)*c(1,1;4,2)*c(2,1;3,1)*c(2,1;4,2)*c(3,1;4,2)*c(1,1;4,3)*c(2,1;4,3); or
the output of the RAOM of Eq. (10).
No other solution is possible as shown in Appendix 2.

The AND gate whose output is a prime implicant of Core Function (Hp 1) or a PoC implying a prime implicant of Core Function (Hp 2) can be defined as a “basic AND gate”. Besides, we can define as a basic AND gate also the AND

gate whose output is an input of the OR gate inside the RAOM. Every basic AND gate is useful to produce one and only one prime implicant of Core Function. Therefore, the total number of basic AND gates appearing in the best implementation of Core Function is always equal to, or larger than, the total number of prime implicants of Core Function.

CONCLUSION

Since the number of AND and OR operations in the best implementation of CF(n) is larger than the number of prime implicants of CF(n) and the number of prime implicants of CF(n) is equal to 3^n, it follows that the number of elementary operations necessary to implement CF(n) grows exponentially. Since Core Function is a new NP-complete problem it is possible to state that the classes of decision problems P and NP do not coincide. In brief, it is easy to deduce the names and the signs of variables appearing in the satisfiability equation from the values of compatibilities. If satisfiability equation grew polynomially, also satisfiability equation would grow polynomially. Since it has been proved in this paper that Core Function grows exponentially, also satisfiability equation grows exponentially.

REFERENCES

1. Meo AR. On the P VS NP question: A New proof of inequality. J Comput Sci. 2021;17:511-524.
2. Meo AR. On a proof of inequality of PvsNP. J Comput Sci. 2023;19:87-98.
3. Meo AR. On the P vs NP question: A proof of inequality. Arxiv preprint.
4. Meo AR. On a proof of inequality of the classes of decision problems P and NP. Glob J Res Comput Sci. 2024;15:003.
5. BP International Editor. Mathematics and Computer Science Contemporary Development. 2024;8.
6. Fortnow L. The status of the P versus NP problem. Commun ACM. 2009;52:78-86.
7. Carlson J, et al. The millennium prize problems. Amer Math Soc. 2023.
8. Razborov. Lower bounds on the monotone complexity of some Boolean functions. InSoviet Math Dokl. 1985;31:485-493.
9. Meo AR. Some theorems concerning the core function. Springer. 2008.
10. Cook SA. The complexity of theorem proving procedures. ACM Press. 2023;143-152.
11. Mulmuley KD, et al. Geometric complexity theory I: An approach to the P vs. NP and related problems. SIAM J. Comput. 2001;31:496-526.
12. Wikipedia: NP-Completeness.
13. Hartmanis J. Computers and intractability: A guide to the theory of np-completeness. Siam Rev. 1982;24:90.
14. Karlsson R. Lecture 8: NP-complete problems. Comput Sci. 2009.
15. Sun HM. The theory of NP-completeness. Wikipedia.
16. Quine WV. The problem of simplifying truth functions. Am Math Mon. 1952;59:521-531.