*Journal of Computer Science*

Original Research Paper

# On the P VS NP Question: A New Proof of Inequality

**Angelo Raffaele Meo**

*Accademia delle Scienze di Torino, Politecnico di Torino, Italy*

*Article history*
Received: 23-09-2020
Revised: 29-04-2021
Accepted: 03-05-2021

Email: raffaele.meo@polito.it

**Abstract:** The analysis discussed in this study is based on a well-known NP-complete problem which is called "satisfiability problem or SAT". From SAT a new NP-complete problem derives, which is described by a Boolean function of the number n of the clauses of SAT called "core function". In this study a new proof is presented according which the number of gates of the minimal implementation of core function increases with n exponentially. Since the synthesis of core function is an NP-complete problem, this result can be considered as the proof of the theorem according which P and NP do not coincide.

**Keywords:** P-NP Question, Complexity, Boolean Functions, Satisfiability, Polynomial or Exponential Increase

## Introduction

In 2009 the author of this paper presented a first proof of inequality concerning the known P-NP question to the Academy of Sciences of Turin. Few years later the same paper was published in the known repository ArXive (Meo, 2018). After a very long debate in May of 2019 the Academy of Sciences of Turin has made the decision to publish that paper (Meo, 2016-2020).

This paper presents a simpler version of that proof based on some new theorems.

### *Definitions*

A brief description of the definitions and properties well known among the scientists of modern computational complexity theory which will be made reference to, is presented in this section:

- *P* denotes the class of all the decision problems which can be solved in polynomial time
- *NP* denotes the class of all the decision problems *f* 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 modern computational complexity theory

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 instances of *L* into instances of *C* producing the same output values.

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, than no *NP*-complete problem has a polynomial time solution.

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

Given a Boolean expression containing only the names of a set 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:

$$\begin{aligned} F = &\left( x_{11}\, OR\, x_{12}\, OR\, x_{13} \right) AND \\ &\left( x_{21}\, OR\, x_{22}\, OR\, x_{23} \right)\, AND \\ &\qquad AND \\ &\left( x_{t1}\, OR\, x_{t2}\, OR\, x_{t3} \right) \end{aligned} \tag{1}$$

where, $t$ is the number of clauses or triplets; each $x_{ij}$ is a variable in complemented or uncomplemented form; each variable may appear multiple times in that expression.

Usually, the deterministic Turin machine is assumed as the computational model. In this study analysis will be developed with reference to a family $\{C_n\}$ of Boolean circuits, where $C_n$ has $n$ binary inputs and it produces the same binary output as the corresponding Turing machine.

The equivalence between a deterministic Turing machine $M$ processing some input $x$ belonging to $\{0,1\}^n$ and an $n$-input Boolean circuit $C_n$ is well known. It is also known that the number of gates, or AND, OR, NOT operators, appearing in circuit $C_n$, is polynomial in the running time of the corresponding Turing machine.

The synthesis of the state of art of question PvsNP can be found in (Fortnow, 2009; Cook, 2006).

## The Core Function

In the case of the satisfiability problem with 3 variables for clause, Boolean circuit $C_n$ has $n$ (= $t$) sets of inputs which the binary data are applied to. The output of $C_n$ (with $n = t$) will take the value TRUE when and only when, there is an assignment of values TRUE and FALSE to variables making expression (1) TRUE.

In order to simplify analysis, circuit $C_n$ will be decomposed into two processing layers as shown in Fig. 1, where, as usual, the number $t$ of triplets plays the role of symbol $n$ in the standard analysis of 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.

### *Compatibility of Two Variables*

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

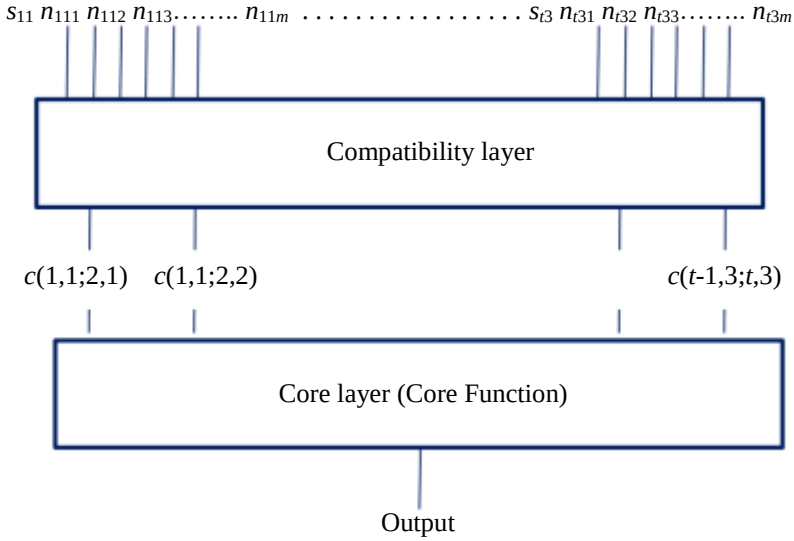
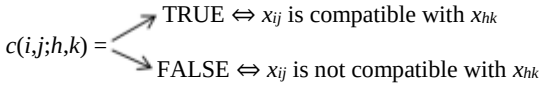



**Fig. 1:** Decomposition of Boolean circuit $C_n$ into a compatibility layer and a core layer



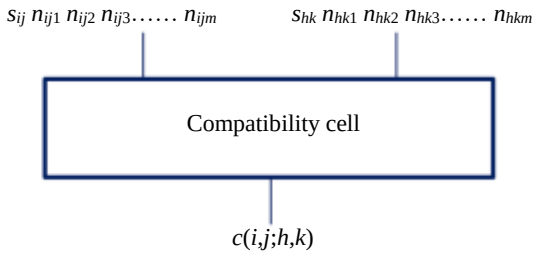




**Fig. 2:** Compatibility cell

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

The compatibility layer is composed of $3 \cdot t \cdot (3 \cdot t\text{-}3)/2$ identical cells, one for each pair of variables belonging to different triplets.

As shown in Fig. 2, the inputs of a cell will be the sign $s_{ij}$ and the binary code $<n_{ij1}\ n_{ij2}\ \ldots n_{ijm}>$ of variable $j$ of triplet $i$ and the sign $s_{hk}$ and the binary code $<n_{hk1}\ n_{hk2}\ \ldots n_{hkm}>$ of variable $k$ of triplet $h$. The output of the same cell $c(i,j;h,k)$ will be TRUE when and only when, the two variables are compatible between themselves.

Variable $c(i,j;h,k)$ will be called a compatibility variable or simply a compatibility.

### *Core Layer*

The core layer processes only the $9 \cdot t \cdot (t\text{-}1)/2$ compatibility variables $c(i,j;h,k)$ and produces the global result of computation.

As the circuit $C_n$, also the global Boolean function implemented by $C_n$ may be decomposed into two layers of functions. At the compatibility layer, the function implemented by a cell may be written as follows (by using the symbols $*$, + and ! for representing AND, OR and NOT operators, respectively):

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

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 $CF(n)$). The core function can be determined by proceeding as follows.

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

$$<1i_1>, <2i_2>, \ldots, <ti_t> \tag{3}$$

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

Be the indexes <number of triplet, number of variable in the triplet> of the selected variables. They will be called "characteristic indexes". Let $\Pi^k$ be the product of all the compatibility variables relative to the $k$-th of selections (3):

$$\begin{aligned} & \Pi^k = c(1,i_1;\ 2,i_2) * c(1,i_1;\ 3,i_3) * \ldots \\ & \ldots * c(t-1,i_{t-1};\ t,\ i_t) \end{aligned} \tag{4}$$

The core function can be defined as the sum:

$$\Sigma_k \Pi^k \tag{5}$$

of the products (4) relative to all the selections (3).

For example, in the case of $CF(3)$, the core function can be defined as follows:

$$\begin{aligned} & CF(3) = c(1,1;2,1) * c(1,1;3,1) * c(2,1;3,1) + \\ & c(1,1;2,1) * c(1,1;3,2) * c(2,1;3,2) + \\ & c(1,1;2,1) * c(1,1;3,3) * c(2,1;3,3) + \\ & c(1,1;2,2) * c(1,1;3,1) * c(2,2;3,1) + \\ & \ldots(other\ 22\ products)\ldots + \\ & c(1,3;2,3) * c(1,3;3,3) * c(2,3;3,3) \end{aligned} \tag{6}$$

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

Notice that the processing work of the cell of Fig. 2 increases as a polynomial function $P(t)$ of the number of the variables since the increment of the length of the code of the name is logarithmic. Therefore, the total processing work of the compatibility layer increases as:

$$9 \cdot t \cdot (t\ -\ 1) \cdot P(t)$$

where, $9 \cdot t \cdot (t\text{-}1)/2$ is the total number of the compatibility cells.

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 in Meo (2008).

Notice that, in order that the circuit represented in Fig. 1 exactly performs the processing work done by 3SAT, the Boolean function implemented by the core layer may be 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:

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

(where, !$c$ denotes the complement of $c$) are incomplete specifications of the Boolean function implemented by the Core Layer of Fig. 1.

However, it is easy to verify that the many incomplete specifications of the type of the preceding ones are not useful to simplify and to reduce the costs of implementation of Boolean function defined by Eq. (4) and Eq. (5). Therefore, in the following analysis the chances represented by the incomplete specifications will be ignored in the analysis of $CF(t)$.

## A Theorem of Boolean Monotonic Functions

Let $f(x_1, x_2, ..., x_t)$ be an isotonic Boolean function, that is a Boolean function which can be implemented with only AND and OR gates, applied to uncomplemented literals $x_1, x_2, \ldots, x_t$. It was believed that the minimum cost implementation of $f(x_1, x_2, \ldots, x_t)$ always contains only OR and AND gates, but Razborov (1985) proved that there are isotonic functions whose minimum cost implementation contains also NOT gates.

However, there is on upper bound on the comparison of the costs of the minimum cost implementations with and without NOT gates. It is specified by the following theorem.

*Theorem 3.1*

Let $I_{min}$ be one of the minimum cost implementations of the isotonic Boolean function $f(x_1, x_2, ..., x_h)$, the cost being defined as the total number of AND, OR or NOT gates. Let $C_{min}$ be the cost of $I_{min}$.

There exists always an implementation $J$ of $f$ containing only AND and OR gates (in addition, if necessary, to the NOT operators producing input variables $!x_1, !x_2, ..., !x_h$) such that:

$$cost(J) <= 2 \cdot C_{min} + h$$

where, $h$ is the number of variables.

In order to prove this theorem, let us divide the gates of implementation $I_{min}$ of f into different levels and let us modify $I_{min}$ as follows.

At level 1 we place the gates all inputs of which coincide with the complemented or uncomplemented input variables $x_i$ or $!x_i$ (where $!x_i$ denotes the complement of variable $x_i$).

Level 2 contains the gates whose inputs coincide with input variables or outputs of level 1 gates.

In general terms, level $q$ contains the gates whose inputs coincide with input variables or outputs of levels less than $q$.

We can transform $I_{min}$ into $J$ by deleting NOT gates and adding new AND or OR gates as follows.

We start from level 1.

For any level 1 AND gate we add an OR gate whose inputs are the complements of the inputs of the considered AND gate (Fig. 3). Similarly, for any level 1 OR gate we add an AND gate whose inputs are the complements of the corresponding OR gate.

By virtue of such operations, for any output $u$ of the level 1 gates a new node will be available in the new circuit we are generating whose value will be !$u$.

As a second step of processing, for any level 2 AND gate of implementation $I_{min}$ we shall add an OR gate whose inputs are the complements of the inputs of the corresponding AND gate, in both the cases in which these inputs coincide with input variables of f or with output of level 1 gates (Fig. 4).

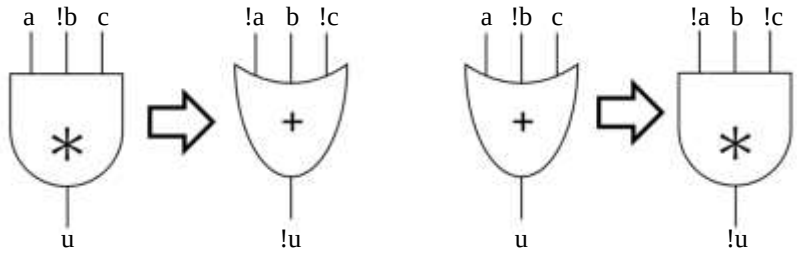



**Fig. 3:** The new gates of level 1

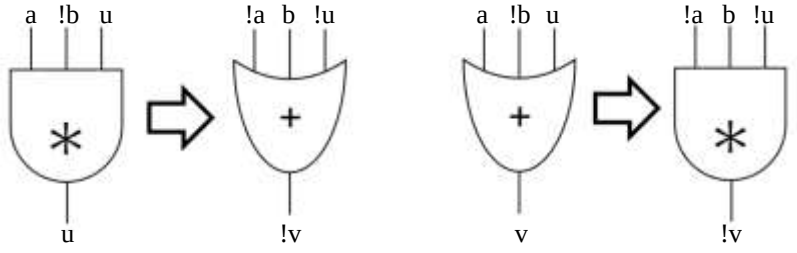



**Fig. 4:** The new gates of level 2

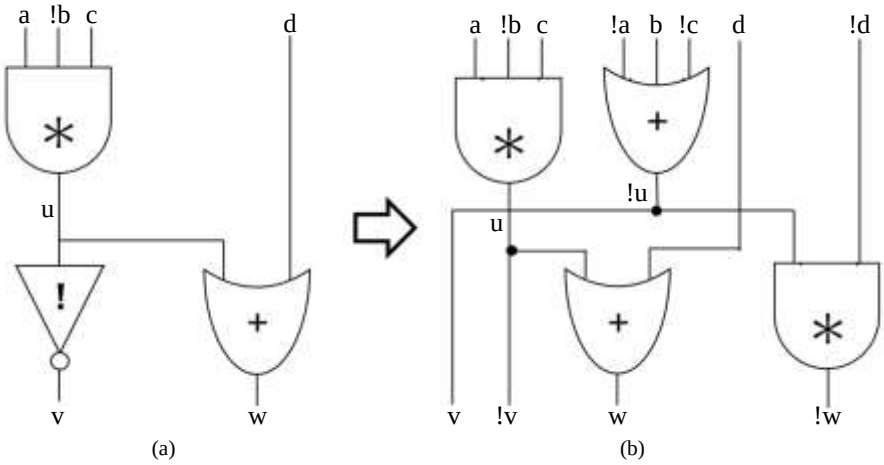



**Fig. 5:** (a) A two level subnetwork; (b) The transformation of the subnetwork of (a)

A similar transformation will be applied to all level 2 OR gates.

As an example, the two level subnetwork of Fig. 5a will be transformed into the subnetwork of Fig. 5b. Notice that at the outputs of J not only the outputs v and w of $I_{min}$ will be available, but also their complements! *v* and! *w*.

The preceding operations will be applied to all the levels of implementation of $I_{min}$, in the order of increasing levels. It is apparent that, if for any input variable $x_i$ also $!x_i$ is available, the number of gates of *J* is less than, or equal to, twice the number of gates of $I_{min}$.

At level 0, before the gates of Fig. 5b, h NOT gates might be necessary to generate the complemented input variables $!x_i$. Therefore, h has been added in the statement of the theorem.

This theorem will be important in order to simplify the analysis of core function circuits.

## Properties of Core Function

It is easy to prove the following properties of core function.

*Property 1*

Function defined by Eq. (5) is totally isotone.

*Property 2*

Any product (4) is a prime implicant of core function (that is, a Product of Compatibilities ("PoC") which implies core function and no other term of it).

*Property 3*

Since the different selections of each of variables (3) are 3, the number of prime implicants of core function is equal to $3^t$. Each of these prime implicants is essential (that is, it does not imply a sum of other prime implicants) and it is the product of $t\cdot(t\text{-}1)/2$ compatibilities.

## Products of Compatibilities

In the next sections, reference will be made to the following definitions.

*Definition of Spurious Compatibilities Pair*

A pair of compatibility variables $\{c(h,k;l,m), c(p,q;r,s)\}$ is defined as a spurious pair if:

($h = p$ and $k \neq q$)
or ($h = r$ and $k \neq s$)
or ($l = p$ and $m \neq q$)
or ($l = r$ and $m \neq s$)

For example, the pair $\{c(1,1;2,1), c(1,2;3,1)\}$ is a spurious pair since the triplet 1 is associated to two different indexes of variables (1 and 2).

*Definition of Spurious Products of Compatibilities*

A spurious Product of Compatibilities (spurious PoC) is a product of compatibility variables containing the elements of one or more than one spurious pair.

For example, the PoC:

$$c(1,1;2,1)*c(1,2;3,1)*c(2,1;3,1)$$

is a spurious PoC since it contains the elements of the spurious pair:

$$\{c(1,1;2,1), c(1,2;3,1)\}$$

*Definition of Impure Products of Compatibilities*

A PoC containing one or more complemented variables will be defined as an impure PoC. In particular a term *T* of CF (that is, a PoC implying CF) which contains one or more complemented variables, will be defined as an impure term.

### *Definition OF Core of a POC*

The product of all the uncomplemented variables of $T$ will be defined as the core of $T$.

### *Definition of Mark*

Consider a not spurious subset of compatibilities satisfying the property that all the indexes of triplet $\{1,2,\ldots,t\}$ appears at least once in some 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:

$$M = (1,a;2,b) * c(1,a;3,c) * c(1,a;4,d) \tag{7}$$

(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:

$$\begin{aligned} P = {} & c(1,a;2,b) * c(1,a;3,c) * c(1,a;4,d) \\ & * c(2,b;3,c) * c(2,b;4,d) * c(3,c;4,d) \end{aligned} \tag{8}$$

since all the indexes of triplet appear at least once in Eq. (7).

### *Definition of Spurious Mark*

A spurious PoC in which all the indexes of triplet appear at least once will be called a "spurious mark". Notice that a spurious mark may be the mark of more than one prime implicant. For example, in the case of CF(3):

$$c(1,1;2,1) * c(1,1;3,1) * c(1,1;2,2)$$

is a spurious mark of both the prime implicants:

$$c(1,1;2,1) * c(1,1;3,1) * c(2,1;3,1)$$

and:

$$c(1,1;2,2) * c(1,1;3,1) * c(2,2;3,1)$$

An impure PoC whose core is a (possibly spurious) mark will be a defined as a (possibly spurious) impure mark.

### *Definition of Extended Prime Implicants*

A term T of core function, that is, an implicant of core function (a product of literals implying core function), contains all the uncomplemented literals of a prime implicant. Therefore, it may be defined as an "extended prime implicant" (only) to remember that it contains all the compatibilities of a prime implicant.

It may be a spurious extended prime implicant or an impure extended prime implicant or both a spurious and impure extended prime implicant.

Notice that an extended prime implicant can be viewed as a (possibly spurious or impure) mark.

### *Definition of Remainder*

A PoC which is neither a (possibly spurious or impure) mark nor an (extended) prime implicant will be called a "remainder".

A remainder R may be associated to more than one prime implicant. For example, in the case of CF(3), $R = c(2,1;3,1)$ is a remainder of the prime implicants:

$$\begin{aligned} P1 &= c(1,1;2,1) * c(1,1;3,1) * c(2,1;3,1) \\ P2 &= c(1,2;2,1) * c(1,2;3,1) * c(2,1;3,1) \\ P3 &= c(1,3;2,1) * c(1,3;3,1) * c(2,1;3,1) \end{aligned} \tag{9}$$

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

### *Property 4*

A not spurious mark $M$ specifies a corresponding prime implicant $P$ uniquely. Indeed, if all the indexes of triplet appear in $M$, the product (4) is completely defined.

We shall write:

$$P = I(M)$$

to state that $P$ is the prime implicant specified by $M$.

As already mentioned, a remainder $R$ does not specify a corresponding prime implicant uniquely. In the example relative to $CF(3)$ above described, three prime implicants correspond to $R = c(2,1;3,1)$, as shown by Eq. (9), since a single index of triplet is missing in that remainder. In general, if $z$ triplets are not involved in $R$, there are $3^z$ different ways of involving the missing triplets.

Hence the following property follows.

### *Property 5*

A not spurious remainder $R$ in which the indexes of $z$ triplets are missing corresponds to $3^z$ different prime implicants.

Finally, the following property can be easily proved.

### *Property 6*

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. Either $P_1$ or $P_2$ is a mark of $P$.

## The External Core Function

Let $I_j$ be a prime implicant of CF($n$). The external core function relative to $I_j$, ECF($n$, $I_j$), is defined as the sum of all the minterms of CF($n$) which imply $I_j$ and no other prime implicant $I_k$ of CF($n$) with $k \neq j$. (Remember

that a minterm of a Boolean function *F* is a product of all the variables of *F*, some complemented and some uncomplemented, implying *F*).

Of course:

$$ECF\left(n,I_j\right)=I_j*\Pi_{k\neq j}\left(!I_k\right) \tag{10}$$

where all the prime implicants $I_j$ of core function are involved and $!I_k$ denotes the complement of $I_k$ (i.e., NOT $I_k$).

The global external core function of order *n*, or ECF(*n*), will be defined as the sum of ECF(*n*, $I_j$)'s relative to all the prime implicants $I_j$ of CF(*n*):

$$ECF\left(n\right)=\sum_j ECF\left(n,I_j\right) \tag{11}$$

The importance of external core function derives from the following theorems.

Their proofs can be found in Meo (2016-2020; 2018; 2008).

*Theorem 6.1*

Let *T* be a term (or extended prime implicant) of CF(*n*). It may be the product of all the compatibilities of a prime implicant $I_j$ of CF(*n*) and other compatibilities, that is:

$$T=I_j*X$$

where, *X* is a possibly empty PoC. *T* can also be written as $T = T(I_j)$.

All the minterms of $T(I_j)$ contained in ECF(*n*) are minterms of ECF(*n*, $I_j$).

*Theorem 6.2*

Let *T* be a term of CF(*n*) implying two or more than two prime implicants of CF(*n*):

$$T=T\left(I_j,I_k\right)$$

The number of minterms of $T(I_j, I_k)$ belonging to ECF(n) is equal to 0.

*Theorem 6.3*

Let $T = T(I_j) = I_j*X$ be a term of CF(*n*) which is spurious for a single compatibility *X*.

If NMT(F) denotes the number of minterms of Boolean function F, the number of minterms of $I_j*X$ contained in ECF($n$,$I_j$) is:

$$NMT\left(I_j*X*ECF\left(n,I_j\right)\right)<=\left(1/2\right)\cdot NMT\left(ECF\left(n,I_j\right)\right) \tag{12}$$

By proceeding in the same way it is possible to generalize the preceding Theorem 6.3 as follows.

*Theorem 6.4*

Let:

$$I_j*X_1*X_2*\ldots X_m$$

be a spurious term characterized by m spurious compatibilities.

The number of its minterms contained in ECF(*n*, $I_j$) is:

$$\begin{aligned}&NMT\left(I_j*X_1*X_2*...*X_m*ECF\left(n,I_j\right)\right)\\&<=\left(1/\left(2^m\right)\right)\cdot NMT\left(ECF\left(n,I_j\right)\right)\end{aligned} \tag{13}$$

*Theorem 6.5*

Let $T = T(I_j)$ be an impure term of CF(*n*) characterized by a single impure variable (!*X*):

$$T=I_j*\left(!X\right).$$

For large values of *n*, the number of minterms of ECF(*n*, $I_j$) contained in *T* is:

$$NMT\left(I_j*\left(!X\right)*ECF\left(n,I_j\right)\right)\approx\left(1/2\right)\cdot NMT\left(ECF\left(n,I_j\right)\right) \tag{14}$$

*Theorem 6.6*

Let $T = T(I_j)$ be an impure term of CF(*n*) characterized by *m* impure variables:

$$T=I_j*\left(!X_1\right)*\left(!X_2\right)*\ldots\left(!X_m\right)$$

For large values of *n*, the number of minterms of ECF(*n*, $I_j$) contained in *T* is:

$$NMT\left(T*ECF\left(n,I_j\right)\right)\approx\left(\left(1/2\right)^m\right)\cdot NMT\left(ECF\left(n,I_j\right)\right) \tag{15}$$

Notice that NMT(ECF(*n*, $I_j$)) = NMT(ECF(*n*, $I_k$)) for any *j* and *k*. It will be called NMT1(*n*).

## The Reference Architecture

Figure 6 shows the network which will implement core function. By virtue of Theorem 3.1, it does not contain NOT gates.

Notice in Fig. 6 that the output of an AND gate becomes always the input of an OR gate and, conversely, the output of an OR gate becomes always the input of an AND gate Indeed, if, for example, the output of an AND gate A becomes the input of another AND gate B, the two gates A and B can be merged into a single AND gate collecting all the inputs of A and B. By virtue of this operation the total number of gates remains constant or it is reduced by one unit. However, this hypothesis has not been applied in the following analysis.

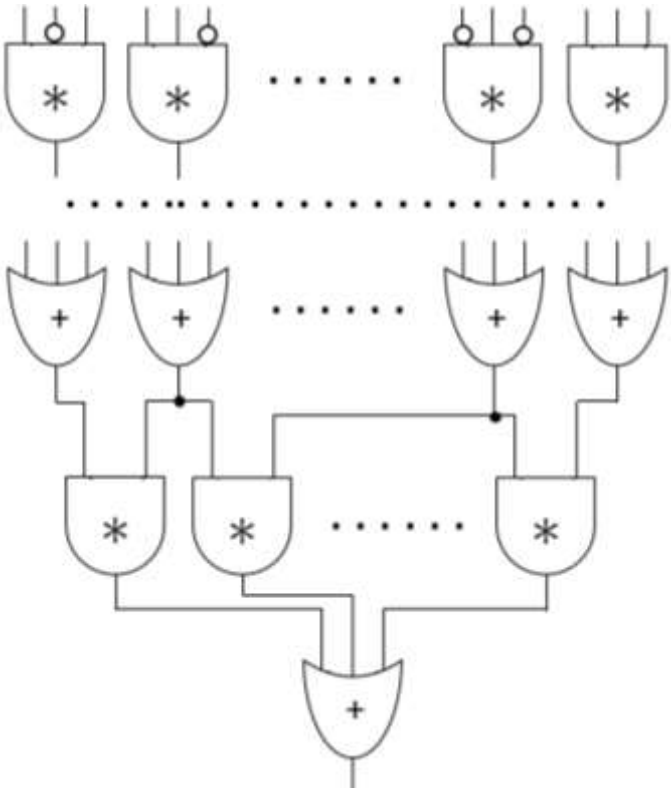


**Fig. 6:** The reference architecture

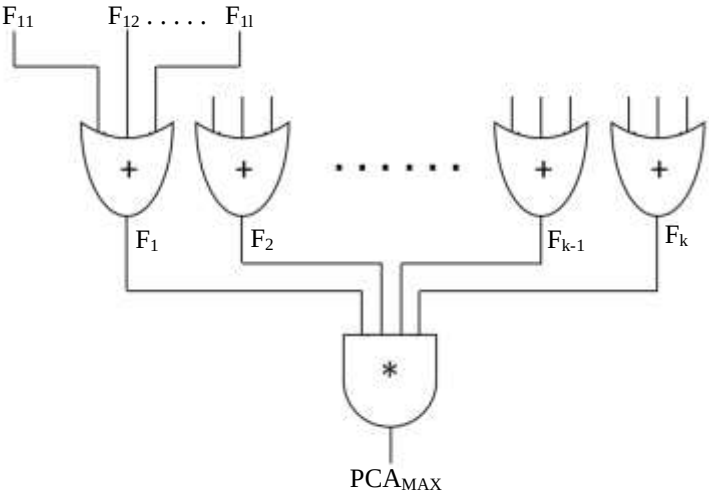



**Fig. 7:** The primary composite addendum of maximum value

Any input of the OR gate producing the final value of CF($n$) will be called a "Primary Composite Addendum (PCA)". Every $F_i$ will be called a "Primary Composite Addendum Factor" (PCAF).

If the number of PCA's of the minimum cost implementation of CF($n$) increased with n according to an exponential law, also the cost of this implementation would increase according to an exponential law, the cost being represented by the number of AND gates at the bottom of Fig. 6.

Therefore, the following analysis refers to the case in which the number of PCA's of the minimum cost implementation of CF($n$) increases with n according to a polynomial law. The PCA characterized by the maximum value among all the values of PCA's will be called $PCA_{MAX}$ (Fig. 7). If the best implementation of CF($n$) will

contain a single PCA, this will have the role of $PCA_{MAX}$. Notice in Fig. 7 that k-1 products ($F_1$*$F_2$*…*$F_k$) produce $PCA_{MAX}$, where $k$ increases according a polynomial law but the number of prime implicants of CF($n$) contained in $PCA_{MAX}$ increases exponentially.

## The Value of an "AND" Operation

Consider an elementary two inputs AND operation $U$ applied to Boolean Functions A and B: $U$ = A*B. Let A and B be specified as sums of their prime implicants. Some of these implicants are marks or prime implicants of core function ($m_1$, $m_2$, $m_3$,…); other implicants of A or B are remainders of core function ($r_1$, $r_2$, $r_3$,…).

The purpose of the considered Boolean product (for example, one of the $k$-1 products of Fig. 7) is to produce new marks and new prime implicants of core function through elementary products of remainders and other marks contained in the lists of implicants of A and B.

Not always a mark deriving from the product of a remainder $r_i$ of A by a remainder $r_j$ of B becomes a useful prime implicant of core function. However, the output $U$ of the product A*B becomes the input of a subnetwork which will produce the value of core function as its output. Since this subnetwork contains no NOT gates, its output can be written as follows:

$$CF = U * x_1 + U * x_2 + \ldots + y_1 + y_2 + \ldots \tag{16}$$

where, $x_1$, $x_2$, …, $y_1$, $y_2$, …, are products of variables of core functions, that is, products of compatibilities. Notice also that every $U$*$x_i$ and every $y_j$ must be an extended prime implicant of core function.

Analysis of Eq. (16) suggests the following definition: "The value of a Boolean product val(A*B) is the number of minterms of ECF contained in the prime implicants of core function appearing in the result of (16) and deriving from new marks, that is marks different from those already available in A or in B."

In order to identify the best solution from the viewpoint of the value of the considered AND product, first consider the following example relative to CF(4):

$$(a_1 + a_2 + a_3) * (b_1 + b_2 + b_3)$$

where:

$$\begin{aligned} a_1 &= c(1,1;4,1) * c(2,1;4,1) \\ b_1 &= c(3.1;4,1) * c(2,1;4,1) \\ a_2 &= c(1,1;4,2) * c(2,1;4,2) \\ b_2 &= c(3,1;4,2) * c(2,1;4,2) \\ a_3 &= c(1,1;4,3) * c(2,1;4,3) \\ b_3 &= c(3,1;4,3) * c(2,1;4,3) \end{aligned} \tag{17}$$

By proceeding as in the above examples and introducing the value:

$$\begin{aligned} x_1 &= c(1,1;2,1) * c(1,1;3,1) * c(2,1;3,1) \\ &* c(1,1;4,1) * c(1,1;4,2) * c(3,1;4,1) * c(3,1;4,2) \\ x_2, \ldots, y_1, y_2, \ldots &= 0 \end{aligned}$$

in Eq. (16), it is easy to prove that the best implementation of this product generates three prime implicants whose total value is equal to (5/16)·NMT1. A way to improve this value consists in adding suitable compatibilities to remainders appearing in (17) in order that any product $a_i$' * $b_j$' * $x_1$ (where $x_1$ = $c$(1,1;2,1)*$c$(1,1:3,1)*$c$(2,1;3,1)) implies CF(4) as in the following example:

$$\begin{aligned} a_1' = a_1 a_2' &= a_2 * c(1,1;4,1) a_3' = a_3 * c(1,1;4,1) * c(1,1;4,2) \\ b_1' = b_1 b_2' &= b_2 * c(3,1;4,1) b_3' = b_3 * c(3,1;4,1) * c(3,1,4,2) \end{aligned} \tag{18}$$

The total merit is equal to:

$$\begin{aligned} &val(a_1' * b_1') + val(a_2' * b_2') + val(a_3' * b_3') \\ &= (1 + 1/4 + 1/16) \cdot NMT1(4) \end{aligned}$$

A better result can be obtained by multiplying $a_i$ and $b_i$ by suitable complemented compatibilities in order that $a_i$ * $b_j$ is equal to 0 if $i$<>$j$. For example:

$$\begin{aligned} a_1'' = a_1 a_2'' &= a_2 * !c(2,1;4,1) a_3'' = a_3 * !c(2,1;4,1) * !c(2,1;4,2) \\ b_1'' = b_1 b_2'' &= b_2 * !c(2,1;4,1) b_3'' = b_3 * !c(2,1;4,1) * !c(2,1;4,2) \end{aligned} \tag{19}$$

The total merit of this new AND operation is equal to:

$$(1 + 1/2 + 1/4) \cdot NMT1(4)$$

In order to prove that Eq. (18) and (19) represent good solutions, first observe that in a product as ($a_1$+$a_2$+$a_3$ +…) * ($b_1$+$b_2$+$b_3$ +…) a term $a_i$ or $b_j$ might be a remainder or a mark (or a prime implicant). However, in Appendix 1 it is shown that no mark appearing in the list of the products of compatibilities $a_i$ or $b_j$ can be useful in order to produce an increase of the value of Boolean product A * B. Therefore, we can assume that all the $a_i$'s and the $b_j$'s are remainders.

Besides, we might hope that a product as ($a_1$+$a_2$) * ($b_1$+$b_2$) can produce four different marks $a_1$*$b_1$, $a_1$*$b_2$, $a_2$*$b_1$, $a_2$*$b_2$. In Appendix 2 it is shown that four different marks can derive from that product, but the total value is very small and it decreases very quickly with the number n of variables.

Also the total value of the product ($a_1$+$a_2$)*$b$ is very small and it decreases very quickly with the number $n$ of variables, as shown in Appendix 3.

The following proof is devoted to Eq. (18).

Appendixes 1, 2 and 3 show that the best solutions from the viewpoint of the value are characterized by a correspondence one-to-one according which a remainder $a_i$ is associated to a single remainder $b_i$ and vice versa. However, the names $a_i$ and $b_j$ must be corrected and they must become spurious or impure PoC's because every product $a_i * b_j * x_1$ must imply CF(n).

If the prime implicant $I_1$ deriving from $m_1 = a_1 * b_1$ is different from the prime implicant $I_2$ deriving from $m_2 = a_2 * b_2$, one of the compatibilities of $m_1$ must make reference to a variable A which does not appear in any of the compatibilities of $I_2$. Besides, $a_1$ cannot contain all the compatibilities involving A since, otherwise, it would not be a remainder, but it would be a mark.

It follows that the product $a_1 * b_2$ must produce prime implicant $I_1$ since it cannot generate $I_2$ or another prime implicant different from $I_1$. Indeed, we can assume that $m_1$ has a value equal to NMT($n$) and, therefore, it is not spurious and it cannot contain a variable as B which characterizes prime implicant $I_2$ or another prime implicant.

If $a_1 * b_2 * x$ produces prime implicant $I_1$, either $b_2$ or $x$ must contain at least one compatibility involving variable A.

The same analysis which has been developed with reference to product $a_1 * b_2$ can be applied to the product $a_2 * b_1$. From such an analysis we can prove that either $b_1$ or $x$ must contain at least a compatibility relative to a symbol A' different from the symbol A characteristic of $I_1$. Therefore, the value of mark $m_2 = a_2 * b_2$ will be equal to, or less than, $(1+1/4)\cdot$NMT1(4).

By applying the same type of analysis it is easy to prove that, if:

$$val(a_1 * b_1) <= NMT1(4)$$

and:

$$val(a_1 * b_1 + a_2 * b_2) <= (1+1/4)\cdot NMT1(4),$$

then:

$$val((a_1 + a_2 + a_3) * (b_1 + b_2 + b_3)) <= (1+1/4+1/16)\cdot NMT1(4),$$

as shown by Eq. (18).

The proof of Eq. (19) can be developed in a similar way. It is simpler.

This solution is extended to CF(6) in Appendix 4, where the hypothesis:

$$x_1 = c(1,1;2,1) * c(1,1:3,1) * c(2,1*3,1)$$
$$x_2, \ldots, y_1, y_2, \ldots = 0$$

has been assumed.

Notice that, if Eq. (16) contains only $x_1$ as it is usual in many examples, $x_1$ may coincide with one of the $k$ primary composite addendum factors $F_j$ appearing in Fig. 7.

## The Best Implementation of the Product A * B

The best implementation of the product of two Boolean functions A and B (such that $A*B*x_1$ is equal to one or more than one prime implicants of CF($n$)) is characterized by the maximum value according our definition of "value of a product".

According to the statements of Appendixes 1 to 3, we can assume that the prime implicants of CF($n$) by A*B are the following:

$$a_1 * b_1 * x_1$$
$$a_2 * b_2 * x_1$$
$$a_3 * b_3 * x_1$$
$$\ldots\ldots\ldots\ldots$$

However, also every product $a_i * b_j * x_1$, with $i <> j$, must be a (possibly spurious) prime implicant of CF(n) (solution 1), unless $a_i * b_j = 0$ and $a_j * b_i = 0$ (solution 2). Equation (18) and (19) make reference to an example where solution 2 is better then solution 1 from the viewpoint of the value of the product. However, it is easy to prove that solution 2 is always better than solution 1. Indeed, at least two spurious compatibilities must be added to $a_j$ and $b_j$ in order that both $a_i * b_j * x_1$ and $b_i * a_j * x_1$ are prime implicants of CF($n$), while a single compatibilty is always sufficient in order that $a_i * b_j = 0$ and $b_i * a_j = 0$. Besides, sometimes a single impure compatibility can make $a_i * b_j = 0$ and $b_i * a_j = 0$ for a given $j$ and many different values of $i$.

For example, in Appendix 4, a single complemented compatibility (!$c$(4,1;5,1)) can make $a_1$, $a_2$, $a_3$ compatible with $b_4$ and $b_1$, $b_2$, $b_3$ compatible with $a_4$, while compatibility !(2,1;4,1) can make $a_1$, $a_2$, …, $a_9$ compatible with $b_{10}$ and $b_1$, $b_2$,…, $b_9$ compatible with $a_{10}$.

It is very easy to prove that any natural order of the type adopted in Appendix 4 for generating the new marks produces the maximum value of the product of two Boolean functions as $F_1 * F_2$ or $(F_1 * F_2) * F_3$ of Fig. 7. It is also easy to prove on the base of analysis of Appendix 4 that the value of a Boolean product is always less than or equal to $(1+1/2+1/4)^{n-m}\cdot$NMT1(n) where n is the number of variables of CF(n) and m is the number of variables (as [1,1], [2,1], [3,1] in the previous examples) each of which appear in all the prime implicants which will be generated from that product.

## The Value of an "OR" Operation

Theoretically, a mark might derive from the Boolean sum of two or more than two remainders. For example, the mark of CF(4):

$$m = c(1,1;4,1) \star c(2,1;4,1) \star c(3,1;4,1)$$

might derive from the sum of the two remainders:

$$r_1 = c(1,1;4,1) \star c(2,1;4,1) \star !c(1,1;3,1)$$

and:

$$r_2 = c(3,1;4,1) * c(1,1;3,1)$$

Let remainders $r_1$ and $r_2$ be two of the inputs of the OR gate producing mark m and let $U$ be the output of this OR gate. Since the circuit producing $PCA_{MAX}$ does not contain NOT circuits, the value of the circuit producing CF can be written as follows:

$$CF = U * x_1 + U * x_2 + \ldots + y_1 + y_2 + \ldots$$
$$= r_1 * x_1 + r_2 * x_1 + r_1 * x_2 + r_2 * x_2 + \ldots + y_1 + y_2 + \ldots$$

Since $r_1$ and $r_2$ are remainders, every $x_i$ must be a mark. Besides, either there is a $y_k = I(m)$ or one of marks $x_j$ coincides with mark $m$, in order to produce the relative implicant. It follows that the production of a mark as the sum of two remainders is not necessary in order to generate its relative implicant.

## Conclusion

The synthesis of Boolean function CF($n$) can be described as a sequence of Boolean operations finalized to producing new marks in every step. The most important of these operations are Boolean AND's which produce new marks from remainders or already available marks with an increase of the value, where the value is defined by the number of minterms of ECF contained in the output marks not contained in the input marks.

The simplest operation is the AND of input variables of the whole network or of already available remainders. This operation is characterized by a value equal to (or less than) NMT1($n$). The most efficient operation is the AND of impure or spurious compatibilities of the type described in Appendix 4, which is characterized by a value equal to $(1+1/2+1/4)^{n-m}$·NMT1($n$).

Notice that a single AND gate having $i$ inputs can perform ($i$-1) AND operations, but usually these ($i$-1) AND operations are associated to $i$ other gates in addition to the considered AND, that is, the gates (typically, OR gates) feeding this AND gate. Therefore, it is easy to verify that the implementation of the $X$ Boolean functions necessary to synthesize CF(n) requires at least $X$ gates.

Therefore, since the number of minterms of ECF($n$) contained in CF($n$) is equal to $3^n$·NMT1($n$) and the merit of a gate is always less than or equal to $(1+1/2+1/4)^{(n-m)}$· NMT1($n$), the number of gates contained in the considered network is larger than $3^n/((1+1/2+1/4)^{(n-m)}$ and, therefore, it increases exponentially with $n$.

Since the synthesis of core function is an NP-complete problem, this result is equivalent to proving that $P$ and NP do not coincide.

One final note: Also the synthesis of the remainders appearing in all the OR or AND gates requires a number of gates which increases exponentially with $n$. The proof of this property is rather complex and it has been omitted in this study since it is not necessary.

## Appendix 1

In the product A*B of two Boolean functions $A = (a_1 + a_2 + a_3 + \ldots.)$ and $B = (b_1 + b_2 + b_3 + \ldots)$, where the $a_i$'s and $b_j$'s are prime implicants of function $A$ and $B$, respectively, no mark contained in the list of $a_i$'s and $b_j$'s is useful in order to produce an increase of the value of A*B.

Consider the following hypotheses.

### *Hyp 1*

$a_i$ is a mark of CF(n) and $b_j$ is a remainder implying $a_1$. For example, $a_i = c(1,1;2,1)*c(1,1;3,1)*c(2,1 ;4,1)$ is a mark of CF(4) and $b_j = c(1,1;3,1)*c(2,1; 4,1)$ is a remainder implying $a_i$.

In this case, the product $a_i*b_j$ coincides with $a_i$ and has the same value.

### *Hyp 2*

$a_i$ is a mark of CF($n$) and remainder $b_j$ does not imply $a_i$, but $a_i$ and $b_j$ imply the same prime implicant of CF($n$). For example, $a_i = c(1,1;2,1)*c(1,1;3,1)*c(2,1; 4,1)$ and $b_j = c(1,1;2,1)*c(3,1;4,1)$.

In this case, a new mark of CF (4) is generated, but this new mark imply the same prime implicant of CF(4) as $a_i$. Therefore, it does not increase the value of $a_i$.

### *Hyp 3*

$a_i$ is a mark implying a prime implicant $p_1$ of CF and $b_j$ is a remainder not implying $p_1$.

For example, $a_i = c(1,1;2,1)*c(1,1;3,1)*c(2,1;4,1)$ and $b_j = c(1,1;2,1)*c(1,1;3,2)$.

In this case, a new mark spurious $a_i*b_j$ is generated.

This imply the same prime implicant as $a_i$, but its value is less than val($a_i$).

### *Hyp 4*

$a_i$ is a mark and $b_j$ is another mark implying the same prime implicant of CF(n) implied by $a_i$.

For example, $a_i = c(1,1;2,1)*c(1,1;3,1)*c(2,1;4,1)$ (mark of CF(4)) and $b_j = c(1,1;2,1)*c(1,1;3,1)*c(1,1;4,1)$.

In this case, val($a_i$) = val($b_j$) = val($a_i*b_j$).

### *Hyp 5*

$a_i$ is a mark of CF($n$) and $b_j$ is another mark of CF($n$), but $a_i$ and $b_j$ imply two different prime implicants of CF($n$).

For example, $a_i = c(1,1;2,1)*c(1,1;3,1)*c(2,1;4,1)$ and $b_j = c(1,1;2,1)*c(1,1;3,1)*c(2,1;4,2)$ are marks of two different prime implicants of CF(4).

In this case, $a_i * b_j$ can be viewed as a spurious mark of both $a_i$ and $b_j$, but $val(a_i*b_j) < val(a_i)$, $val(a_i*b_j) < val(b_j)$, $val(a_i*b_j) < val(a_i) + val(b_j)$.

## Appendix 2

Consider the product $(a_1 + a_2) * (b_1 + b_2)$ relative to CF(4) where:

$$a_1 = c(1,1;2,1)*c(1,1;3,1)*c(1,1;3,2)$$
$$a_2 = c(1,1;2,2)*c(1,1;3,2)*c(1,1;3,1)$$
$$b_1 = c(2,1;3,1)*c(2,1;4,1)*c(3,1;4,1)*c(2,2;3,1)*c(2,2;4,1)$$
$$b_2 = c(2,2;3,2)*c(2,2;4,1)*c(3,2;4,1)*c(2,1;3,2)*c(2,1;4,1)$$

with $x = c[1,1]*c[4,1]$.

The following four marks of CF(4) are generated:

$$m_1 = a_1 * b_1 involving\ variables([1,1],[2,1],[3,1],[4,1])$$
$$m_2 = a_1 * b_2 involving\ variables([1,1],[2,1],[3,2],[4,1])$$
$$m_3 = a_2 * b_1 involving\ variables([1,1],[2,2],[3,1],[4,1])$$
$$m_4 = a_2 * b_2 involving\ variables([1,1],[2,2],[3,2],[4,1])$$

It is easy to verify that:

$$val(m_1) = val(m_2) = val(m_3) = val(m_4) = (1/8)\cdot NMT1(4)$$

Therefore, the total value of the considered product is ½·NMT1(4).

Now consider the following product $(a_1 + a_2) * (b_1 + b_2)$ relative to CF(5), where:

$$a_1 = c(1,1;2,1)*c(1,1;3,1)*c(1,1;5,1)*c(1,1;3,2)$$
$$a_2 = c(1,1;2,2)*c(1,1;3,2)*c(1,1;5,1)*c(1,1;3,1)$$
$$b_1 = c(2,1;3,1)*c(2,1;4,1)*c(2,1;5,1)*c(3,1;4,1)*c(3,1;5,1)$$
$$*c(4,1;5,1)*c(2,2;3,1)*c(2,2;4,1)*c(2,2;5,1)$$
$$b_2 = c(2,2;3,2)*c(2,2;4,1)*c(2,2;5,1)*c(3,2;4,1)$$
$$*c(3,2;5,1)*c(4,1;5,1)*c(2,1;3,2)*c(2,1;4,1)*c(2,1;5,1)$$

It is easy to verify that:

$$m_1 = a_1 * b_1$$
$$m_2 = a_1 * b_2$$
$$m_3 = a_2 * b_1$$
$$m_4 = a_2 * b_2$$

are four marks implying four different prime implicants of CP(5) and that:

$$val(m_1) = val(m_2) = val(m_3) = val(m_4) = (1/16)\cdot NMT1(5)$$

In more general terms, the product $(a_1 + a_2) * (b_1 + b_2)$ can produce four marks implying four different prime implicants of CF($n$), but the value of one of these marks is:

$$val(m_i) = 1/(2^{n-1})\cdot NMT1(n).$$

Therefore, it decreases very quickly with $n$.

## Appendix 3

Consider the following product:

$$(a_1 + a_2) * b$$

where, $a_1$, $a_2$ and b are remainders of CF(4) which take the following values:

$$a_1 = c(1,1;2,1)*c(1,1;3,1)$$
$$a_2 = c(1,1;2,2)*c(1,1;3,1)$$
$$b = c(2,1;3,1)*c(2,1;4,1)*c(2,2;3,1)*c(2,2;4,1)*c(3,1;4,1)$$

The considered product generates the following marks of CF(4):

$$m_1 = a_1 * b$$
$$m_2 = a_2 * b$$

taking the following values:

$$val(m_1) = ¼\cdot NMT1(4)$$
$$val(m_2) = ¼\cdot NMT1(4)$$

In general, the product $(a_1 + a_2) * b$ where $a_1$, $a_2$ and $b$ are remainders of CF($n$) produces two marks whose total value is equal to $1/(2^{n-1})$·NMT1($n$).

This value is very small and it decreases very quickly with $n$.

## Appendix 4

In order to understand how it is possible to generalize the results of section 8 consider the following Boolean product $(a_1 + a_2 + \ldots)*(b_1 + b_2 + \ldots)$ relative to CF(6):

$a_1$ = $c$(1,1;4,1)*$c$(1,1;5,1)*$c$(1,1;6,1)*$c$(2,1;4,1)*$c$(2,1;5,1)* $c$(2,1;6,1)*$c$(4,1;5,1)*$c$(4,1;6,1)*$c$(5,1;6,1)

$b_1$ = $c$(3,1;4,1)*$c$(3,1;5,1)*$c$(3,1;6,1)*$c$(2,1;4,1)*$c$(2,1;5,1)* $c$(2,1;6,1)*$c$(4,1;5,1)*$c$(4,1;6,1)*$c$(5,1;6,1)

$a_2$ = $c$(1,1;4,1)*$c$(1,1;5,1)*$c$(1,1;6,2)*$c$(2,1;4,1)*$c$(2,1;5,1)* $c$(2,1;6,2)*$c$(4,1;5,1)*$c$(4,1;6,2)*$c$(5,1;6,2)*!$c$(5,1;6,1)

$b_2$ = $c$(3,1;4,1)*$c$(3,1;5,1)*$c$(3,1;6,2)*$c$(2,1;4,1)*$c$(2,1;5,1)* $c$(2,1;6,2)*$c$(4,1;5,1)*$c$(4,1;6,2)*$c$(5,1;6,2)*!$c$(5,1;6,1)

$a_3$ = $c$(1,1;4,1)*$c$(1,1;5,1)*$c$(1,1;6,3)*$c$(2,1;4,1)*$c$(2,1;5,1)* $c$(2,1;6,3)*$c$(4,1;5,1)*$c$(4,1;6,3)*$c$(5,1;6,3)*!$c$(5,1;6,1)* !$c$(5,1;6,2)

$b_3$ = $c$(3,1;4,1)*$c$(3,1;5,1)*$c$(3,1;6,3)*$c$(2,1;4,1)*$c$(2,1;5,1)* $c$(2,1;6,3)*$c$(4,1;5,1)*$c$(4,1;6,3)*$c$(5,1;6,3)*!$c$(5,1;6,1)* !$c$(5,1;6,2)

$a_4$ = $c$(1,1;4,1)*$c$(1,1;5,2)*$c$(1,1;6,1)…!$c$(4,1;5,1)

$b_4$ = $c$(3,1;4,1)*$c$(3,1;5,2)*$c$(3,1;6,1)…!$c$(4,1;5,1)

$a_5$ = $c$(1,1;4,1)*$c$(1,1;5,2)*$c$(1,1;6,2)…!$c$(4,1;5,1)* !$c$(5,2;6,1)

$b_5$ = $c$(3,1;4,1)*$c$(3,1;5,2)*$c$(3,1;6,2)…!$c$(4,1;5,1)* !$c$(5,2;6,1)

$a_6$ = $c$(1,1;4,1)*$c$(1,1;5,2)*$c$(1,1;6,3)…!$c$(4,1;5,1)* !$c$(5,2;6,1)*!$c$(5,2;6,2)

$b_6$ = $c$(3,1;4,1)*$c$(3,1;5,2)*$c$(3,1;6,3)…!$c$(4,1;5,1)* !$c$(5,2;6,1)*!$c$(5,2;6,2)l

$a_7$ = $c$(1,1;4,1)*$c$(1,1;5,3)*$c$(1,1;6,1)…!$c$(4,1;5,1)* !$c$(4,1;5,2)

$b_7$ = $c$(3,1;4,1)*$c$(3,1;5,3)*$c$(3,1;6,1)…!$c$(4,1;5,1) *!$c$(4,1;5,2)

$a_8$ = $c$(1,1;4,1)*$c$(1,1;5,3)*$c$(1,1;6,2)…!$c$(4,1;5,1)* !$c$(4,1;5,2)*!$c$(5.3;6,1)

$b_8$ = $c$(3,1;4,1)*$c$(3,1;5,3)*$c$(3,1;6,2)…!$c$(4,1;5,1)* !$c$(4,1;5,2)*!$c$(5,3;6,1)

$a_9$ = $c$(1,1;4,1)*$c$(1,1;5,3)*$c$(1,1;6,3)…!$c$(4,1;5,1)* !$c$(4,1;5,2)*!$c$(5,3;6,1)*!$c$(5,3;6,2)

$b_9$ = $c$(3,1;4,1)*$c$(3,1;5,3)*$c$(3,1;6,3)…!$c$(4,1;5,1)* !$c$(4,1;5,2)*!$c$(5,3;6,1)*!$c$(5,3;6,2)

First, consider the lines relative to $a_1$, $b_1$, $a_2$, $b_2$, $a_3$, $b_3$. The merits of the marks produced by those lines are, respectively:

$$\mu(a_1 * b_1) = 1 \cdot NMT1(6)$$
$$\mu(a_2 * b_2) = (1/2) \cdot NMT1(6)$$
$$\mu(a_3 * b_3) = (1/4) \cdot NMT1(6)$$

Similarly, the merits of the marks produced by the lines $a_4$, $b_4$, $a_5$, $b_5$, $a_6$, $b_6$, $a_7$, $b_7$, $a_8$, $b_8$, $a_9$ $b_9$ are:

$$\mu(a_4 * b_4) = (1/2) \cdot NMT1(6)$$
$$\mu(a_5 * b_5) = (1/2) \cdot (1/2) \cdot NMT1(6)$$
$$\mu(a_6 * b_6) = (1/2) \cdot (1/4) \cdot NMT1(6)$$
$$\mu(a_7 * b_7) = (1/4) \cdot NMT1(6)$$
$$\mu(a_8 * b_8) = (1/4) \cdot (1/2) \cdot NMT1(6)$$
$$\mu(a_9 * b_9) = (1/4) \cdot (1/4) \cdot NMT1(6)$$

It follows that the total merit of the nine lines considered is equal to:

$$(1+1/2+1/4)^2 \cdot NMT1(6)$$

The set of the lines $a_1$, $b_1$, …, $a_9$, $b_9$ can be extended as follows:

$a_{10}$ = $c$(1,1;4,2)*$c$(1,1;5,1)*$c$(1,1;6,1)* $c$(2,1;4,2)*$c$(2,1;5,1)*$c$(2,1;6,1)* $c$(4,2;5,1)*$c$(4,2;6,1)*$c$(5,1;6,1)*!$c$(2,1;4,1)

$b_{10}$ = $c$(3,1;4,2)*$c$(3,1;5,1)*$c$(3,1;6,1)* $c$(2,1;4,2)*$c$(2,1;5,1)*$c$(2,1;6,1)* $c$(4,2;5,1)*$c$(4,2;6,1)*$c$(5,1;6,1)*!$c$(2,1;4,1) …….

$a_{19}$ = $c$(1,1;4,3)*$c$(1,1;5,1)*$c$(1,1;6,1)* $c$(2,1;4,3)*$c$(2,1;5,1)*$c$(2,1;6,1)* $c$(4,3;5,1)*$c$(4,3;6,1)*$c$(5,1;6,1) * !$c$(2,1;4,1)*!$c$(2,1;4,2)

$b_{19}$ = $c$(3,1;4,3)*$c$(3,1;5,1)*$c$(3,1;6,1)* $c$(2,1;4,3)*$c$(2,1;5,1)*$c$(2,1;6,1)* $c$(4,3;5,1)*$c$(4,3;6,1)*$c$(5,1;6,1)* !$c$(2,1;4,1)*!$c$(2,1;4,2)

It follows that the total merit of the considered lines $a_1$, $b_1$, …, $a_{10}$, $b_{10}$…, $a_{19}$, $b_{19}$…, becomes:

$$(1+1/2+1/4)^3 \cdot NMT1(6)$$

By proceeding along this line of reasoning, it is easy to prove that the merit of a Boolean product of the type of the above presented products is equal to:

$$(1+1/2+1/4)^{n-m} \cdot NMT1(n)$$

where, of course, m is the number of variables (as <1,1>, <2,1>, <3,1> in the preceding examples) each of which appears in all the prime implicants which will be generated from that product.

## Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

## References

Cook, S. (2006). The P versus NP problem. In: Carlson, J., Jaffe, A., & Wiles, A. (Eds.), The Millennium Prize Problem, (pp. 870104), Providence: American Mathematical Society. ISBN-10: 082183679X.

Fortnow, L. (2009). The status of the P versus NP problem. Communications of the ACM, 52(9), 78-86. https://doi.org/10.1145/1562164.1562186

Meo, A. R. (2008) Some Theorems Concerning the Core Function. In: Degano, P., De Nicola R., & Meseguer J. (Eds.), Concurrency, Graphs and Models, (pp. 778-796). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-68679-8_48

Meo, A. R. (2016-2020). On the P versus NP question. Accademia delle Scienze di Torino. https://www.accademiadellescienze.it/attivita/editoria/lavori-di-soci

Meo, A. R. (2018). On the P vs NP question: a proof of inequality. arXiv preprint arXiv:1802.05484. https://arxiv.org/abs/1802.05484

Razborov, A. A. (1985). Lower bounds for the monotone complexity of some Boolean functions. Soviet Mathematics-Doklady, 31, 354-357. https://ci.nii.ac.jp/naid/10003041149/