A Theorem on monotonic Boolean functions and its application to "P=NP?" question

Memoria del Socio nazionale ANGELO RAFFAELE MEO
presentata nell'adunanza del 22 novembre 2006

Abstract. *A theorem is proved according which there exists always an implementation of an isotonic Boolean function containing only AND or OR gates whose cost is less than the double of the cost of its minimum cost implementation with AND, OR and NOT gates.*

The theorem is applied to proving the disequality of the known classes P and NP.

Keywords: isotonic Boolean function, AND, OR and NOT gates, classes P and NP.

Riassunto. *Si dimostra un teorema secondo il quale esiste sempre un'attuazione di una funzione booleana isotona contenente solo circuiti AND e OR, il cui costo è inferiore al doppio del costo della sua attuazione di costo minimo con circuiti AND, OR e NOT.*

Il teorema è applicato alla dimostrazione della diseguaglianza delle note classi P e NP.

Parole chiave: funzione booleana isotona, circuiti AND, OR e NOT, classi P e NP.

1. Introduction

In a preceding paper an NP-complete problem has been discussed pertaining to a function, called "core function", which plays an important role in the well known Boolean satisfiability problem. In that paper, it was proved that the minimal Boolean implementation of the core function contains a number of gates which increases with the number of variables exponentially.

In this paper the same analysis is performed on the basis of the following theorem on Boolean isotonic functions.

2. A theorem of Boolean isotonic functions

Let $\mathbf{f(x_1,x_2, ..., x_n)}$ be an isotonic Boolean function, that is a Boolean function which can be implemented with only **AND** and **OR** gates.

Let $\mathbf{I_{min}}$ be one of its minimum cost implementations among all the implementations of **f**, the cost being defined as the total number of **AND**, **OR** or **NOT** gates. Let $\mathbf{C_{min}}$ be the cost of $\mathbf{I_{min}}$.

Theorem

There exists always an implementation **J** of **f** containing only **AND** and **OR** gates such that cost (**j**) < = 2 * $\mathbf{C_{min}}$.

Proof

Let us divide the gates of implementation $\mathbf{I_{min}}$ of **f** into different levels.

At level **l** we place the gates whose inputs coincide all with the complemented or uncomplemented input variables $\mathbf{x_i}$ **or** $\mathbf{!x_i}$ (where $!\mathbf{x_i}$ denotes the complement of variable $\mathbf{x_i}$).

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 $\mathbf{I_{min}}$ into **J** by deleting **NOT** gates and adding new **AND** or **OR** gates as follows.

We start from level **1**.

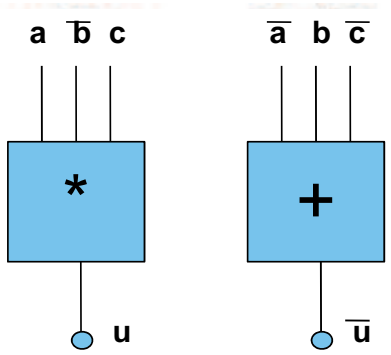

Figure 1: The transformation of the gates of level **1**.

For any level **1 AND** gate we join an **OR** gate whose inputs are the complemented inputs of the considered **AND** gate (Fig. 1). Similarly, for any level 1 **OR** gate we join 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 $\mathbf{I_{min}}$ we shall join an **OR** gate whose inputs are the complements of the inputs of the corresponding **OR** gate, in both the cases when these inputs coincide with input variables of **f** or with outputs of level **1** gates (Fig. 2).

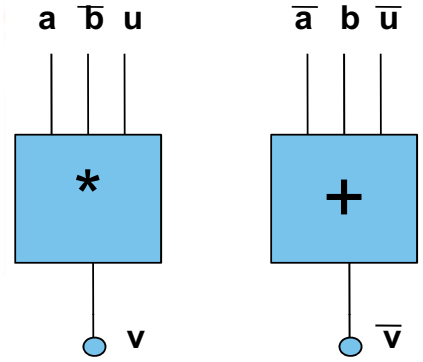

Figure 2: The transformation of the gates of level **1**.

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

As concerns **NOT** gates, any level **1 NOT** gate will be removed since the output of that gate is already available as the output of a level **1** gate.

As an example, the two level subnetwork of Fig. 3 will be transformed into the subnetwork of Fig. 4. Notice that at the outputs of **J** not only the outputs **u** and **w** of $\mathbf{I_{min}}$ will be available, but also their complements !**u** and !**v**.

The preceding operations will be applied to all the levels of implementation $\mathbf{I_{min}}$, in the order of increasing levels. It is apparent that the number of gates of **J** is less than twice the number of gates of $\mathbf{I_{min}}$.

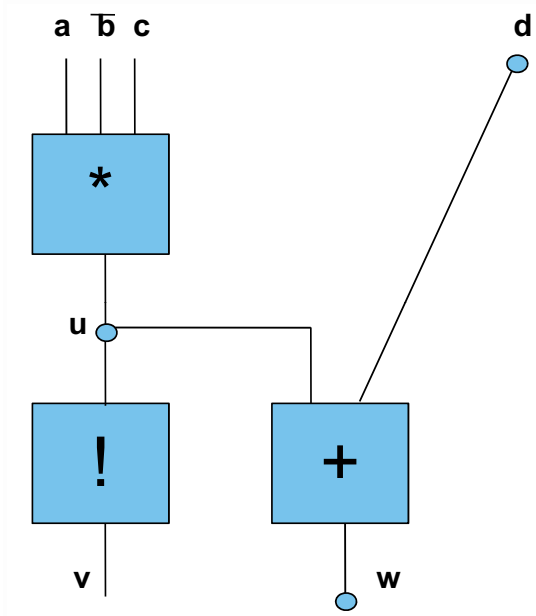

Figure 3: A two level subnetwork.

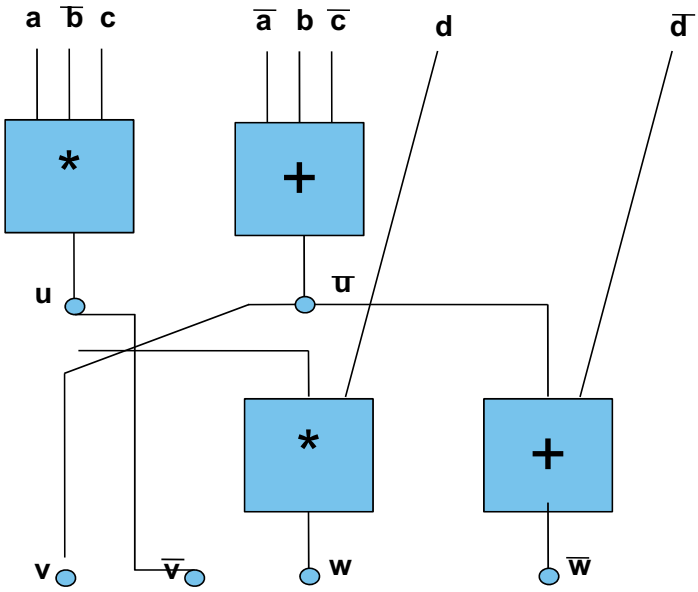

Figure 4: The transformation of subnetwork of figure 3.

3. The question "P = NP?"

The simple theorem proved in the preceding section will be applied to the analysis of a well known problem, namely the question "**P** = **NP**?". It will used to simplify the sketch of a proof presented in a previons paper [1].

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 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, con-

versely, 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 paper will be based on a 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, the operators **AND**, **OR** and **NOT**, and parentheses, is there an assignment of **TRUE** and **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. In this case, the analysed expressions will be of the type:

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

where:
t is the number of clauses or triplets;
each x_{ij} is a variable with or without a **NOT** operator in front of it;
each variable can appear multiple times in the expression.
If the deterministic Turing machine is assumed as the computational model, with {0,1,b} as its set of input symbols, the input data appearing on the tape at the beginning of computation can represent the data of expression (1) in the following way:

…b b <binary code of number of variables> <separator>

$$
\begin{array}{lllll}
s_{11} & n_{111} & n_{112} & \ldots & n_{11m} \\
s_{12} & n_{121} & n_{122} & \ldots & n_{12m} \\
s_{13} & n_{131} & n_{132} & \ldots & n_{13m} \qquad (2) \\
s_{21} & n_{211} & n_{212} & \ldots & n_{21m} \\
\ldots & \ldots & \ldots & \ldots & \ldots \\
s_{t3} & n_{t31} & n_{t32} & \ldots & n_{t3m}\ b\ b \ldots
\end{array}
$$

where:
b is the blank symbol;
t is the number of triplets;
$\mathbf{s_{ij}}$ denotes the sign of variable $\mathbf{x_{ij}}$ (with $\mathbf{s_{ij} = 1}$ denoting that $\mathbf{x_{ij}}$ is preceded by operator **NOT**);
$\mathbf{n_{ijk}}$ denotes the **k**-th component of the binary code $<\mathbf{n_{ij1}}\ \mathbf{n_{ij2}} \ldots \mathbf{n_{ijm}}>$ representing the name of variable $\mathbf{x_{ij}}$;
the binary code of the number $\mathbf{n_v}$ of variables is needed in order to determine the number **m** of binary digits necessary to represent the names of variables according the rule **m**=minimun integer not less than $\mathbf{log_2\ n_v}$;
the binary code of the separator must not be included in the binary code of the number of variables.

Notice that, by neglecting the bits of the binary code of the number of variables and the bits of the separator, the number of input bits on the tape will be

$$\mathbf{n = t * 3 * (1 + \text{minimum integer not less than } \log_2 3t)} \qquad (3)$$

since the maximun value of the number of variables is **3t**.

The properties of Turing machines processing the bit string described by (2) will be analyzed in this paper with reference to a family $\{\mathbf{C_n}\}$ of Boolean circuits. $\mathbf{C_n}$ has **n** binary inputs and produces the same binary output as the corresponding Turing machine.

The equivalence between a deterministic Turing machine **M** processing some input $\mathbf{x \in \{0,1\}n}$ and an **n**-input Boolean circuit $\mathbf{C_n}$ is well known. It is also known that the number of gates, or **AND, OR, NOT** operators, appearing in circuit $\mathbf{C_n}$, is polynomial in the running time $\mathbf{T_M(n)}$ of the corresponding Turing machine. We shall prove here that the number of gates of the circuit $\mathbf{C_n}$ increases exponentially with **n**; it follows that also $\mathbf{T_M(n)}$ increases exponentially.

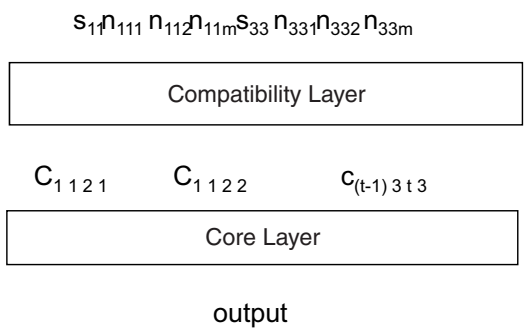

Figure 5: Decomposition of Boolean circuit C_n into the compatibility layer and core layer.

Boolean circuit $\mathbf{C_n}$ has **n** inputs which the binary data described in (2) are applied to. (Of course, the binary code of the number of variables and the separator are not needed). The output of $\mathbf{C_n}$ 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 $\mathbf{C_n}$ will be decomposed into two processing layers as shown in Fig. 5.

A variable **j** of triplet **i** will be defined as "compatible" with variable **k** of triplet **h** when, and only when, either the sign $\mathbf{s_{ij}}$ of the former variable is equal to the sign $\mathbf{s_{hk}}$ of the latter or the name $<\mathbf{n_{ij1}}\ \mathbf{n_{ij2}}\ \ldots\mathbf{n_{ijm}}>$ of the former is different from the name $<\mathbf{n_{hk1}}\ \mathbf{n_{hk2}}\ \ldots\mathbf{n_{hkm}}>$ of the latter. From that definition it follows that two not compatible variables have different signs and the same name; therefore, their **AND** is identically **FALSE**.

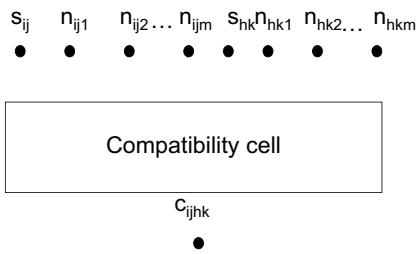

Figure 6: Compatibility Cell.

The compatibility layer is composed of **3t(3t-3)/2** identical cells, one for each pair of variables belonging to different triplets.

As shown in Fig. 6, the inputs of a cell will be the sign $\mathbf{s_{ij}}$ and the name $<\mathbf{n_{ij1}}\ \mathbf{n_{ij2}}\ \ldots\mathbf{n_{ijm}}>$ of variable **j** of triplet **i**, and the sign $\mathbf{s_{hk}}$ and the name $<\mathbf{n_{hk1}}\ \mathbf{n_{hk2}}\ \ldots\mathbf{n_{hkm}}>$ of variable **k** of triplet **h**. The ouput of the same cell $\mathbf{c_{ijhk}}$ will be **TRUE** when, and only when, the two variables are compatible between themselves.

Variable $\mathbf{c_{ijhk}}$ will be called a compatibility variable or simply a compatibility.

The core layer processes only the **3t(3t-3)/2** compatibility variables $\mathbf{c_{ijhk}}$ and produces the global result of computation.

As the circuit $\mathbf{C_n}$**,** also the global Boolean function implemented by $\mathbf{C_n}$ 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):

$$\mathbf{c_{ijhk} = s_{ij} * s_{hk} + !s_{ij} * !s_{hk} +}$$

$$+ \mathbf{n_{ij1}} * !\mathbf{n_{hk1}} + !\mathbf{n_{ij1}} * \mathbf{n_{hk1}} + \tag{4}$$

$$+ \mathbf{n_{ij2}} * !\mathbf{n_{hk2}} + !\mathbf{n_{ij2}} * \mathbf{n_{hk2}} +$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$\mathbf{n_{ijm}} * !\mathbf{n_{hkm}} + !\mathbf{n_{ijm}} * \mathbf{n_{hkm}}$$

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)**. The core function can be determined by proceeding as follows.

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

$$\mathbf{1i_1, 2i_2, \ldots, ti_t} \tag{5}$$

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

be the indexes <number of triplet, number of variable in the triplet> of the selected variables. Let **Π** be the product of all the compatibility variables relative to all the pairs of the selected variables:

$$\mathbf{\Pi = c_{1\,i1\,2\,i2} * c_{1\,i1\,3\,i3} * \ldots * c_{t\text{-}1\,it\text{-}1\,t\,it}} \tag{6}$$

The core function can be defined as the sum

$$\mathbf{\Sigma\Pi} \tag{7}$$

of the products (6) relative to all the selections (5). For example, in the case of 3 triplets, the core function can be defined as follows:

$$\begin{aligned}\mathbf{CF(3)} = {} & \mathbf{c_{1121} * c_{1131} * c_{2131}} + \\ & \mathbf{c_{1121} * c_{1132} * c_{2132}} + \\ & \mathbf{c_{1121} * c_{1133} * c_{2133}} + \\ & \mathbf{c_{1122} * c_{1131} * c_{2231}} + \\ & \mathbf{...(other\ 22\ products)...} \\ & \mathbf{c_{1323} * c_{1333} * c_{2333}}\end{aligned} \tag{8}$$

It is easy to prove that Eq. (**1**) is **TRUE** when, and only when, the core function takes the value **TRUE.**

Notice that the processing work of a cell of Fig. 6 increases with the logarithm of the number of the variables since such is the increment of the code of the name of a variable. Therefore, the total processing work of the compatibility layer can be written as:

$$\mathbf{K * 3t * (3t - 3) * \log_2 3t}$$

where **3t * (3t – 3)/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 polinomially reduces an **NP**-complete problem **(3SAT)** to the problem solved by the core layer, the core function describes a new **NP**-complete problem. This will be the problem discussed in the following sessions.

Properties of the core function

In the previons paper [1] the following properties of a core function have been proved.

Property 1

The core function CF is totally isotonic.

Property 2

Any product (6) is a prime implicant of the core function (that is, a product of variables which is included by no other).

Property 3

Since the different selections of each of variables (5) are **3**, the number of prime implicants of the core function is equal to $\mathbf{3^t}$**.** Each of these prime implicants is essential and is the product of **t(t-1)/2** compatibilities.

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

Definition of spurious term

A pair of compatibility variables **c**hklm and **c**pqrs 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**1121 and **c**1231 is a spurious pair since the triplet **1** is associated to two different indexes of variables (**1** and **2**).

A spurious term is a product of compatibility variables countaining the elements of one or more than one spurious pair.

For example, the term

$$\mathbf{c}_{1121} * \mathbf{c}_{1231} * \mathbf{c}_{2131}$$

is a spurious term since it countains the elements of the spurious pair $\{\mathbf{c}_{1121}, \mathbf{c}_{1231}\}$

Definition of impure term

A term **t** of the core function **CF** (namely, a product of literals implying **CF**), which contains one or more complemented variables, will be defined as an impure term. 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 the variables listed in (**6**) satisfying the property that each of the indexes of triplet appears at least once in some variable. The product of the variables of such a subset will defined as a "mark" or a "not spurious mark" of the prime implicant which it is part of.

For example, in the case of **CF** (4), the product

$$\mathbf{m} = \mathbf{c}_{1a2b} * \mathbf{c}_{1a3c} * \mathbf{c}_{1a4d} \tag{9}$$

(where **a, b, c, d** are elements of **{1,2,3}**)

is a "not spurious mark" of the prime implicant

$$\mathbf{p} = \mathbf{c}_{1a2b} * \mathbf{c}_{1a3c} * \mathbf{c}_{1a4d} * \mathbf{c}_{2b3c} * \mathbf{c}_{2b4d} * \mathbf{c}_{3c4d} \tag{10}$$

since all the indexes of triplet appear at least once in (9).

Notice that a product of variables may be the mark of more than one prime implicant, if it a spurious term. In this case it will be called a spurious mark. Besides, it might be impure.

Definition of extended prime implicants

A term **t** of the core function which contains all the uncomplemented literals of a prime implicant will be defined as an "extended prime implicant" or as an "impure or spurious prime implicant" of **CF**.

Notice that an extended prime implicant can be viewed as a mark.

Definition of remainder

A term (namely a product of complemented or uncomplemented variables) which is neither a mark nor an extended prime implicant will be called a "remainder". Generally, a remainder can be associated to one or more prime implicants.

For example,

$$\mathbf{r} = \mathbf{c}_{2b3c} * \mathbf{c}_{2b4d} * \mathbf{c}_{3c4d} \tag{11}$$

is a remainder of the prime implicant (10). As will be shown shortly, a remainder **r** "belongs" to more than one prime implicant, in the sense that

$$\mathbf{r} <= \mathbf{p}_i$$

for a set $\{\mathbf{p}_i\}$ of prime implicants of the core function.

For example, in the case of **CF(3)**, $\mathbf{c}_{2131}$ is a remainder of the prime implicants

$$\begin{aligned} \mathbf{p}_1 &= \mathbf{c}_{1121} * \mathbf{c}_{1131} * \mathbf{c}_{2131} \\ \mathbf{p}_2 &= \mathbf{c}_{1221} * \mathbf{c}_{1231} * \mathbf{c}_{2131} \\ \mathbf{p}_3 &= \mathbf{c}_{1321} * \mathbf{c}_{1331} * \mathbf{c}_{2131} \end{aligned} \tag{12}$$

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 (6) is completely defined.

We shall write

$$\mathbf{p} = \mathbf{I(m)}$$

to state that **p** is the prime implicant specified by **m**.

The same equation will be written to state that **p** is one of the prime implicants specified by the spurious mark **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 $\mathbf{c_{2131}}$, as shown by (12), since a single index of triplet is missing in that remainder. In general, if **z** triplets are not involved in **r**, there are **3z** different ways of involving the missing triplets. Hence the following property.

Property 5

A not spurious remainder **r** in which the indexes of **z** triplets are missing corresponds to $\mathbf{3^z}$ different prime implicants.

Property 6

Let $\mathbf{P_1}$ and $\mathbf{P_2}$ be two products of variables such that $\mathbf{P_1} * \mathbf{P_2}$ is equal to a prime implicant **P** of a core function. Either $\mathbf{P_1}$ or $\mathbf{P_2}$ is a mark of **P.**

4. The reference architecture

The theorem proved in Section 2 makes it possible to simplify the proof of the disequality of **P** and **NP**.

Indeed, we shall consider only the implementations of the core function with **AND** and **OR** gates only (isotonic implementations). We shall prove that no isotonic implementation exists which increases not exponentially with the number of variables; hence we shall derive that no implementation of general type (with **NOT** gates also) exists which increases with the number of variables exponentially.

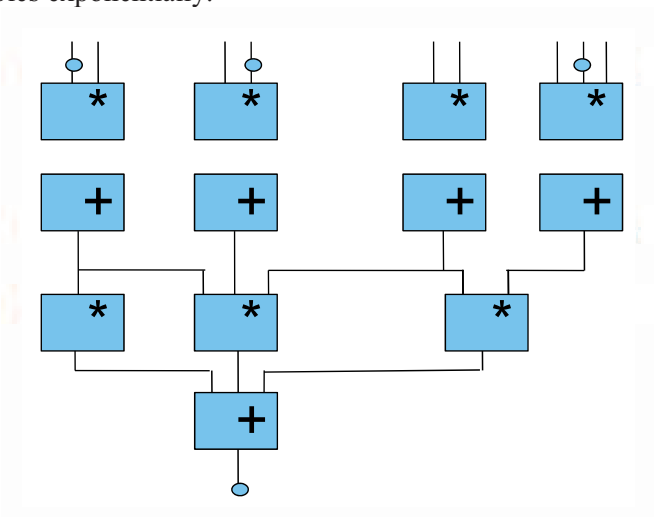

Figure 7: The reference architecture.

Fig. 7 shows the related architecture characterized by a number of subnetworks each of which has the structure shown by Fig. 8. As an alternative,

the network of Fig. 7 might be composed by a single network of the type of Fig. 8.

Each of the circuits presented in Fig. 8 will be called a "primary composite addendum (**PCA**)" and each of the **OR** gates together with its input gates will be called a "primary factor of a primary composite addendum (**PCAF**)".

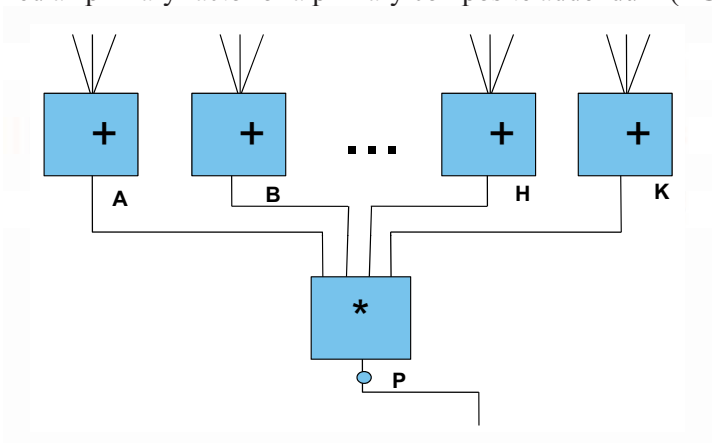

Figure 8: The structure of a PCA.

Consider the Boolean values of the functions implemented by the circuits preceding nodes **A, B, H, K, P,**

val (A)

val (B)

.....

val (H)

val (P) = val (A) * val (B) *...* val (H) * val (K)

Each of these Boolean values can be expressed as the sum of its prime implicants and, therefore, as a sum of terms of the type:

$$(M_1 + M_2 + M_3 + ... + R_1 + R_2 + R_3 + ...)$$

where the M_i 's are marks and the R_i 's are remainders.

It is easy to prove that the following four different cases may occur

1.

$$\mathbf{val\ (A) * val\ (B) *...*val\ (H) = M_{11} + M_{12} + M_{13} +...}$$

and

$$\mathbf{val\ (K) = M_{21} + M_{22} + M_{23} +...+ R_{21} + R_{22} + R_{23} +...}$$

2.

$$\mathbf{val\ (A) * val\ (B) *...* val\ (H) = M_{11} + M_{12} + M_{13} +...}$$

and

$$\mathbf{val\ (K) = M_{21} + M_{22} + M_{23} +....}$$

3.

$$\mathbf{val\ (A) * val\ (B) *...*val\ (H) = M_{11} + M_{12} + M_{13} +...+ R_{11} + R_{12} + R_{13} +...}$$

and

$$\mathbf{val\ (K) = M_{21} + M_{22} + M_{23} +...}$$

4.

$$\mathbf{val\ (A) * val\ (B) *...* val\ (H) = M_{11} + M_{12} + M_{13} +...R_{11} + R_{12} + R_{13} +...}$$

and

$$\mathbf{val\ (K) = M_{21} + M_{22} + M_{23} +...+ R_{21} + R_{22} + R_{23} +...}$$

but, for any **i, j,**

$$\mathbf{R_{1i} * R_{2j} = 0}$$

Indeed, if any $\mathbf{R_{1i} * R_{2j}}$ were different from **0** identically, val (**P**) would be different from the core function.

For the sake of brevity, we shall restrict analysis to case **1**, leaving the reader the task of analyzing the other three cases by the same arguments we shall apply to case **1**.

As mentioned a prime implicant can be interpreted as a mark; so, in the preceding relationships some mark $\mathbf{M_{ij}}$ might be a prime implicant.

5. The merit of a PCA

The Boolean function implemented by a **PCA** is the sum of prime implicants, some of which are extended.

An extended prime implicant contains only a fraction of the minterms contained in the corresponding not extended prime implicant. Therefore, we shall define the merit of a not extended prime implicant as the value 1 and the merit of an extended prime implicant as the fraction of minterms of the corresponding prime implicants it covers.

Finally, the merit of a **PCA** will be defined as the sum of the merits of all its extendended and not extended prime implicants.

6. The structure of PCA's and PCAF's

Any product $\mathbf{M_{1i} * M_{2j}}$ or $\mathbf{M_{1i} * R_{2j}}$ must coincide with a prime implicant (or with an extended prime implicant) of the core function.

First consider the products of the type $\mathbf{M_{1i} * R_{2j}}$. It is easy to prove that any product of this family must coincide with one of the prime implicants $\mathbf{I(M_{1i})}$ generated by $\mathbf{M_{1i}}$.

Notice that a spurious mark might generate a term of the core function which is the extension of two or more prime implicants. However, this case will not be considered here since the number of minterms contained in such an extension decreases very quickly with the number of variables of core function.

In what follows we shall need a new definition.

Definitions of complete and nearly complete marks

A mark **M** will be defined as "complete in a given variable of a given triplet" if it contains all the compatibilities involving that variable. For example the mark

$$\mathbf{c_{1121} * c_{1131}}$$

of **CF(3)** is complete in nodo 11.

A mark **M** will be defined as "nearly complete in a given variable of a given triplet " if it contains nearly all the compatibilities involving that variable, the exceptions being at maximum equal to **e*N** where **e** is arbitrarily small and **N** the total number of compatibilities.

On this definition the following theorem is based.

Theorem

Consider two products of the following type:

$$\mathbf{M_{11} * R_{21} = I(M_{11})}$$

$$\mathbf{M_{12} * R_{22} = I(M_{12})}$$

and assume that no complemented variables appear in all the terms involved.

If $\mathbf{(M_{11})}$ is different from $\mathbf{I\ (M_{12})}$, then $\mathbf{M_{11}}$ and $\mathbf{M_{12}}$ must be nearly complete in two different variables.

Proof

For the sake of brevity we shall present here only a not formal proof.

Assume that $\mathbf{M_{11}}$ is complete in a given variable of a given triplet and $\mathbf{M_{12}}$ is nearly complete in another variable with the exception of a compatibility **c.**

Since also $\mathbf{M_{12} * R_{21}}$ must be equal to $\mathrm{I}(\mathbf{M_{12}})$, then $\mathbf{R_{21}}$ must contain that compatibility **c**. As a consequence the product $\mathbf{M_{11} * R_{21}}$ will contain **c** and, therefore, it will be spurious.

Its merit will be **1/2.**

Assume now that $\mathbf{M_{11}}$ is complete in a given variable and $\mathbf{M_{12}}$ is nearly complete in another variable with the exceptions of two compatibilities $\mathbf{c_1}$ and $\mathbf{c_2}$. In this case $\mathbf{R_{21}}$ must contain both $\mathbf{c_1}$ and $\mathbf{c_2}$, and its merit will be 1/4.

As a synthesis, the merit decreases very quickly with **e*N.**

The previons theorem can be extended to the products of the type $\mathbf{M_{1i} * M_2}$ j according the following theorem.

Theorem

Consider the products

$$\mathbf{M_{11} * M_{21}} \quad \text{and} \quad \mathbf{M_{12} * M_{22}}$$

$\mathbf{M_{11}} * \mathbf{M_{21}}$ and $\mathbf{M_{12}} * \mathbf{M_{22}}$ must be equal to the prime implicants specified by $\mathbf{M_{11}}$ and $\mathbf{M_{12}}$, respectively. Besides, $\mathbf{M_{11}}$ and $\mathbf{M_{12}}$ must be nearly complete in two different variables.

As a consequence of the preceding theorems we can state the following theorem.

Theorem

If condition **1** of Section 4 is satisfied and the hypothesis is assumed that no complemented variables occur, **val (A) * val (B)* ... *val** (**H**) (Fig. 8) can be expressed as a sum of nearly complete marks or prime i implicants.

7. The decomposition of a PCA in factors

Consider again the function

$$\mathbf{val(A) * val(B) * ...*val(H)}$$

under the hypothesis that it is the sum of nearly complete marks and no complemented variable occur. We are interested now in determing **val(H).**

Assume

$$\mathbf{val(A) * val(B)*... = (a+b+c+...)}$$

$$\mathbf{val(H) = (m+n+o+...)}$$

where **a, b, c, ..., m, n, o, ...** are the prime implicants of the Boolean functions they represent.

Of course,

$$\mathbf{(a+b+c+...) * (m+n+o+...) = M_{11} + M_{12} + ...}$$

Let

$$\mathbf{a * m = M_{11}}$$

$$\mathbf{b * n = M_{12}}$$

Consider

$$\mathbf{a * n}$$

Since **a*n** will be one of addenda of ($\mathbf{M_{11}}$ + $\mathbf{M_{12}}$ **+...**) each of which must be nearly complete in one of its variables, only one of the two following conditions is possible.

1.1) **a** is nearly complete in the variables characteristic of $\mathbf{M_{11}}$;

1.2) **n** is nearly complete in the variables characterist of $\mathbf{M_{12}}$.

Now consider the product **b * m.**

Since also **b *m** will be one of the addenda of ($\mathbf{M_{11}}$ + $\mathbf{M_{12}}$ **+...**), only one of the two following conditions is possible.

2.1) **b** is nearly complete in the variables characteristic of $\mathbf{M_{12}}$;

2.2) **m** is nearly complete in the variables characteristic of $\mathbf{M_{11}}$.

The combinations of the cases 1.1, 1.2, 2.1, 2.2 produce the following conclusive alternatives.

3.1) **a** is nearly complete as $\mathbf{M_{11}}$ and **b** is nearly complete as $\mathbf{M_{12}}$;

3.2) **a** and **m** are nearly complete as $\mathbf{M_{11}}$;

3.3) **b** and **n** are nearly complete as $\mathbf{M_{12}}$;

3.4) **m** is nearly complete as $\mathbf{M_{11}}$ and **n** is nearly complete as $\mathbf{M_{12}}$.

In the cases **3.1** and **3.4** (**a + b+...**) and (**m + n +...**) have the same type of structure as ($\mathbf{M_{11}}$ + $\mathbf{M_{12}}$ **+...**); in the cases **3.2** and **3.3** (**a + b +...**) and (**m + n +...**) contain a number of nearly complete terms less by a unit than the number of near complete marks contained in ($\mathbf{M_{11}}$ + $\mathbf{M_{12}}$ **+...**).

The same line of reasoning apply also to the decomposition in factors of (**a + b +...**) or (**m + n +...**). Therefore, we can state the following theorem.

Theorem

In the decomposition of a **PCA** in factors of the type shown in Fig. 7 and Fig. 8 the number of the generated factors is equal to or larger than the number of prime implicants produced by that **PCA**. As a consequence also the merit of PCA is less than the number of the corresponding prime implicants.

8. The decomposition of a PCAF in addenda

Let **val (A)** be the Boolean function implemented by the subnetwork whose output is node **A** of the circuit of Fig. 8. Node **A** is a **PCAF** resulting

from $\mathbf{n_1}$ decomposition operations of **PCA P** in factors. Assume that **val (A)** is equal to

$$\mathbf{R_1 + R_2 + ... + R_p + M_1 + M_2 + ... + M_q}$$

and, therefore, it contains **q** nearly complete marks.

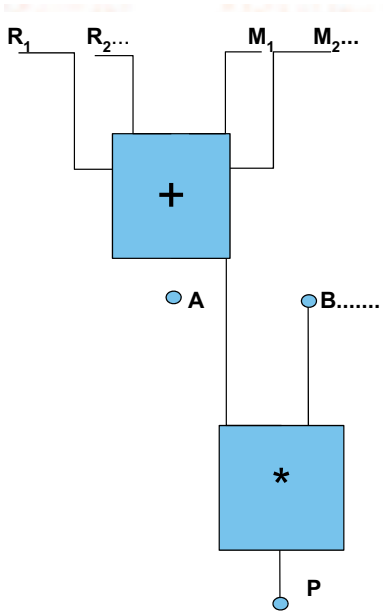

Figure 9: A structure of a PCAF.

A might be the output of an **OR** gate whose inputs take the values $\mathbf{R_1}$, $\mathbf{R_2}$,..., $\mathbf{M_1}$, $\mathbf{M_2}$.., as shown in Fig. 9.

In this case each of the value $\mathbf{M_1}$, $\mathbf{M_2}$,... can be implemented with an **AND** gate having a subset of the input variable $\mathbf{x_1}$, $\mathbf{x_2}$,... of **CF** as its inputs. It is apparent that the total number of gates implementing **PCA P** is larger than $\mathbf{n_1 + q}$ and, therefore , it is larger than the number of prime implicants implemented by **P.**

Alternatively **A** might be the output of an **OR** gate whose inputs take values of the type $\sum \mathbf{R} + \sum \mathbf{M}$ as shown by the example of Fig. 10. Also in this case, an addendum as $(\mathbf{R_1} + \mathbf{M_1} + \mathbf{M_2})$ can be decomposed in factors by using a number of gates larger than **2**, the number of nearly complete marks. Also in this case the total number of gates implementing **PCA P** is larger than the number of prime implicants implemented by **P.**

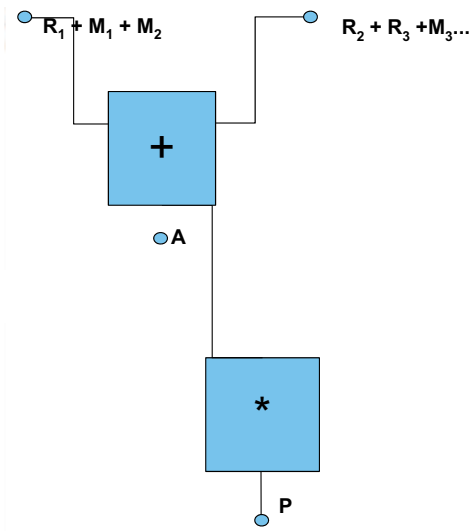

Figure 10: A second type of structure of a PCAF.

9. The role of complemented variables

So far the hypothesis has been assumed that no complemented variables are contained in marks or remainders. Such a simple hypothesis will be removed in this section.

Let us return to the first decomposition in factors shown in Fig. 7 and Fig. 8. The relationships:

$$\mathbf{val(A) * val(B) *....*val(H) = M_{11} + M_{12} +...}$$

$$\mathbf{val(K) = M_{21} + M_{22} + ...+ R_{21} + R_{22} +....}$$

presented in Section **4** are still valid, but now marks and remainders may contain complemented variables.

Subdivide the set of marks $\mathbf{M_{1i}}$ into the following subsets:

marks containing no complemented variables;
marks containing one complemented variable $!\mathbf{c_{11}}$;
marks containing one complemented variable $!\mathbf{c_{12}}$;
.....
marks containing two complemented variables $!\mathbf{c_{11}}\ !\mathbf{c_{12}}$;
marks containing two complemented variables $!\mathbf{c_{11}}\ !\mathbf{c_{13}}$;
.....

marks containing three complemented variables $!\mathbf{c_{11}}\ !\mathbf{c_{12}}\ !\mathbf{c_{13}}$;
and so on.

Of course, some of these subsets may be empty.

Let **m** be the number of elements contained in the largest subset.

The evaluations discussed in previous sections apply to first subset and, with minor changes, to each of the other subsets. So, a single decomposition can reduce the number of marks of all the subsets by one unit. It follows that the number of gates contained in a **PCA** will be

$$\mathbf{cost > = m}$$

Now we can evaluate the merit of the considered **PCA**. The merit of a prime implicant generated by the marks of the first subset which contain no complemented variables may be equal to **1**, but the merit of each of the prime implicants generated by the marks of the other subsets decreases very quickly with the number of complemented variables contained in the generated mark. Indeed, the merit of each of the prime implicants generated by a mark containing one complemented variable is less than (or equal to) 1/2; the merit of each of the prime implicants generated by a mark containing two complemented variables is less than (or equal to) 1/4; and so on. If **n** is the number of complemented variables used to subdivide the set of marks into disjoint subsets, the total merit of a PCA will be less than the following limit:

$$\mathbf{m * 1 + m * n * 1/2 +}$$
$$\mathbf{+ m * bin(n,2) * (1/2)^2 +}$$
$$\mathbf{+ m * bin(n,3) * (1/2)^3 +}$$
$$\mathbf{.....}$$

where **bin** (**n,i**) denotes the number of combination of **n** elements **i** by **i**.

10. Conclusion

From the analysis presented in previous section we can deduce an upper bound for the merit of a gate, which is specified by the following value;

$$\mathbf{1 + n/2 + bin(n,2) * (1/2)^2 +}$$

$$\mathbf{+ bin(n,3) * (1/2)^3 + ...}$$

But **n** is less than **N**, the total number of compatibilities, and $\mathbf{3^N}$ is the number of prime implicants of the core function.

It follows that the number of gates necessary to implement the core function, represented by $\mathbf{3^N}$ divided by the above specified merit of a gate, increases with **N** exponentially. This amounts to proving the dis equality sign between **P** and **NP.**

References

[1] MEO A.R., *On the minimization of core function*, Acc. Sc. Torino Memorie Sc. Fis. 29 (2005), 155-178, 7 ff.

[2] SIPSER M., *Introduction to the theory of computation*, PWS Publ. Comp., 1997.

[3] SMALE S., *Mathematical problems for the next century*, Mathematical Intelligencer, 20 (2) (1998), pp. 7-15.

[4] GAREY M.R. and JOHNSON D.S., *Computers and intractability* (1979). New York, W.H. Freeman and Co.

[5] ASSER G., *Das Reprasentantenproblem im Pradikatenkalkul der ersten Stufe mit Identitat*, Zietschr. f. Mathematisches Logik u. Grundlagen der Math. 1 (1955), pp. 252-263.

[6] COOK S.A., *The complexity of theorem proving procedures*, in: *Proc. 3rd Annual ACM Symp. on Theory of Computing* (1971), ACM Press, pp. 151-158.

[7] COOK S.A., *Computational complexity of higher type functions*. In Ichiro SARAKE editor – *Proceeding of the International Congress of Mathematicians* – Kyoto, Japan, 1991, Springer-Verlag, pp. 55-60.

[8] GODEL K., *A letter to J.von Neumann*, in *Sipser's article listed above*, or in: *Arithmetic, Proof Theory and Computational Complexity*, eds. CLOTE P. and KRAJICEK J., Oxford Press (1993).

[9] KARP R.M., *Reducibility Among Combinatorial Problems*, *Complexity of Computer Computations*, pp. 85-103, Plenum Press, 1972.

[10] LEVIN L., *Universal Search Problems* (in Russian), *Problemy Peredachi Informatsii* (= *Problems of Information Transmission*), 9 (3), pp. 265-266 (1973). A partial English translation in: TRAKHTENBROT B.A., *A Survey of Russian Approaches to Perebor (Brute-Force Searches) Algorithms*, Annals of the History of Computing, 6 (4), (1984), pp. 384-400.

[11] SCHOLZ H., *Problem #1: Ein ungelostes Problem in der symbolisches Logik*, J. Symbolic Logic, 17 (1952), p. 160.

[12] SIPSER M., *The history and status of the P versus NP question*, in: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing* (1992), ACM Press, pp. 603-618.

[13] BAKER T., GILL J. and SOLOVAY R., *Relativizations of the P =? NP question*, SIAM J. of Computing, Vol. 4 (1975), pp. 431-442.

[14] BOPPANA R., and SIPSER M., *Complexity of finite functions*, in: *Handbook of Theoretical Computer Science*, ed. VAN LEEUWEN J. (1990), pp. 758-804.

[15] KRAJICEK J., *Bounded arithmetic, propositional logic, and complexity theory*, Encyclopedia of Mathematics and Its Applications, Vol. 60, Cambridge University Press (1995).

[16] PUDLAK P., *The lengths of proofs*, in: *Handbook of Proof Theory*, ed. S.R. BUSS, North-Holland (1998).