Discussion of a 3SAT Logical Network

Marek Malinowski
Warsaw University of Technology
Branch in Plock
Department of Physics and Mathematics
ul. Łukasiewicza 17, 09-400 Płock
e-mail: m.malinowski@pw.plock.pl

Abstract: The title discussion is preceded by an overview of results of studies conducted using a model of logical networks. The presented results permit the conclusion that there exists a monotone polynomial network for a 3SAT problem. In particular it is claimed that (1) a 3SAT problem can be treated as a conjunction of elementary 3SAT problems, that (2) for any elementary 3SAT problem there is a universal logical network of constant size (number of gates), and finally that (3) for the conjunction of elementary 3SAT problems there exists a polynomial network, and that it is a monotone network.

Key words: logical networks, 3SAT problem, computational complexity

Make a good start
[in the field of problem solving theory]

I. INTRODUCTION: Formulating the Argument

Resolving the nature of the relationship of including classes $P \subseteq NP$ is based on parameterization of the complexity of classes due to a computation model (a Turing machine), a mode of calculation (determinism and non-determinism), resources (time and memory) and constraints (***O(.)***notation).

Approved method of parameterization has become the basis for the concept of the Cook-Levin theorem. It sets out the original ***NP-complete*** problem, and indirectly suggests that, if demonstrated to be a polynomial algorithm for a 3SAT problem (or any other ***NP-complete*** problem), then $\boldsymbol{N = NP}$. On the other hand, if the lower exponential restriction is proved, then $\boldsymbol{P \neq NP}$.

In the course of the study, some other simple models, equivalent to a Turing machine model, were added to the mode of parameterization of complexity of classes, which was defined at the very beginning. It is known, for example, that Turing machines can be simulated by using logical networks. The proof of a theorem that combines the complexity of logical networks with the complexity of time logic can be found in the book by Michael Sipser [1].

The usefulness of the model of logical networks while resolving the problem of *P versus NP* results from close relationships of logical networks and the *SAT* satisfiability problem.

Firstly, the *SAT* problem has been formulated in terms of propositional calculus. Since clauses can be expressed in the form of logical functions, the *SAT* can be defined by logical functions, and these, in turn, can be conveniently represented in the form of logical networks.

Secondly, the proof of Cook-Levin theorem can alternatively be performed using just logical networks [1, pp. 401-408].

And finally, the logical networks are in the form of practical implementation of digital circuits that compute logical functions [2].

It is also known that every problem in class ***P*** has a polynomial network. Unfortunately, since there are polynomial network for unresolved problems, the reverse of this statement is not true.

Therefore, a concept of monotone polynomial networks is introduced. And further, by claiming that a monotone polynomial network which represents a certain problem exists if and only if this problem belongs to class ***P***, they are associated with polynomial calculations.

The above statement can be regarded as equivalent to the Cook-Levin theorem. It indicates that if there exists a monotone polynomial for some ***NP-complete*** problem, then ***P = NP***. On the other hand, if it is proved that ***NP-complete*** problems do not have monotone polynomial networks, then ***P ≠ NP***.

In spite of some assertions (cf. Razborow's theorem) which favor the hypothesis that ***NP-complete*** problems do not have polynomial networks that are either monotonous or non-monotonous (**Hypothesis B** [3, p. 287]), it will be further argued that there exists a monotone polynomial network for the original ***NP-complete*** problem, which is *3SAT*.

The term of ***NP*** problem itself speaks against **Hypothesis B**. It says that the verification (checking) of the certificate of solution to the problem takes place in deterministic polynomial time. It can therefore be assumed that the verification process will be carried out by a polynomial network. This situation corresponds to the statements and descriptions of the network and their relationships with problems CIRCUIT SAT and CIRCUIT VALUE and the definition of a monotone network[1], which will be referred to at a later stage (Fact 2.7).

Subsequently, there will be exploited results of research in the model of logical networks that have already been addressed in literature.

II. LOGICAL FUNCTIONS AND LOGICAL NETWORKS

The basic statements referring to the model of logical networks can be clearly presented by a number of facts, most of which have been taken from the book by Christos Papadimitriou [3].

[1] In fact, by definition, a monotonous network is intended to constitute a "special" kind of certificate – a word in which all variables have a fixed logical value *true*.

Fact 2.1: Equivalence of logical formulae and logical functions [3, p. 95].

Each formula φ with variables $x_1, \ldots, x_n$ is the *n*-arguments logical function *f* and conversely, any *n*-arguments logical function *f* can be expressed as a formula φ containing variables $x_1, \ldots, x_n$.

Fact 2.2: Representation of logic functions by logical networks[3, pp. 95-96].

Any logic function can be represented as a logical network, defined as a graph $C=(V,E)$ where $V=\{1, \ldots, n\}$ are the gates of graph *C*, whereas *E* is the set of edges (i, j) at $i < j$ (a-cyclicity condition).

The syntax of the network additionally links type $s(i) \in \{true,false\} \cup \{x_1, \ldots, x_n\} \cup \{\vee, \wedge, \neg\}$ to each gate $i \in V$ and is characterized by the degree of input (number of falling edges) equal to *0*, *1* or *2*. The vertex which has no outgoing edges, called the output gate of the network, calculates the logic function.

If we consider networks that have multiple outputs, they can calculate several functions simultaneously. Based on such a definition of a network, it is possible to construct three types of networks:

- a network without variables (Fig. 1 a), i.e. with gates $V \in \{true, false\} \cup \{\vee, \wedge, \neg\}$;
- a network with variables and no constants (Fig. 1 b), i.e. with gates $V \in \{x_1, \ldots, x_n\} \cup \{\vee, \wedge, \neg\}$;
- a network with variables and constants (Fig. 1 c), i.e. with gates $V \in \{true, false\} \cup \{x_1, \ldots, x_n\} \cup \{\vee, \wedge, \neg\}$..

Note 2.1: Each multi-output network can be reduced in an elementary way by means of a *Cext* expansion network to a network with one output. For example, for any *m* output network, a *Cext* expansion network can be composed of not more than (*m-1*) AND gates in a sequential or parallel arrangement.

By allowing multi-output networks, each of the types of networks can, by analogy with combination systems, be treated as a multi-pole expansion. Circuit diagrams of selected types of network are shown in Figure 1.

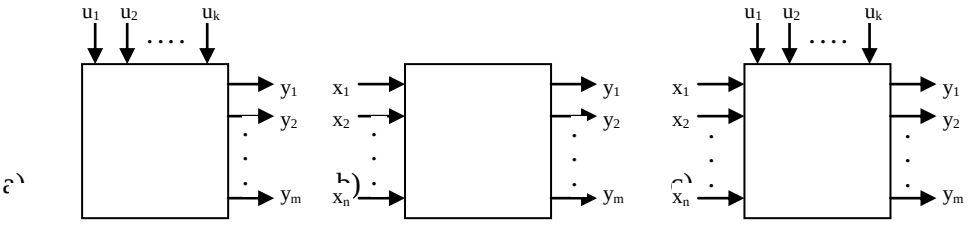

Fig. 1. Circuit diagrams of selected types of logical networks, where:
X – the vector of input signals (variables $x_1, x_2, \ldots, x_n$)

U – the vector of constants $u_i \in \{true,false\}$ $i=1, 2, \ldots, k$;
Y - the vector of outputs, $y_j \in \{true,false\}$ $j=1, 2, \ldots, m$ is the calculated value of a function (not necessarily a function of all n variables.)

Subsequent statements involve logical networks describing problems of different classes of computational complexity.

Fact 2.3: The network as a logical *CIRCUIT VALUE* problem [3, p. 97].

Network *C* without variables (i.e. the network of gates $V(i) \in \{true,false\} \cup \{\vee, \wedge, \neg\}$) (a circuit diagram of Figure 1 a) defines the problem *CIRCIUT VALUE* $\in$ ***P***.

Fact 2.4: Logical network as a problem of *CIRCUIT SAT* [3, p. 97].

Network *C* with variables (Fig. 1 b, c) where it should be noted that there is such a valuation of variables that the output gate returns *true*, defines the problem *CIRCIUT SAT*.

Showing [3, p. 179] a polynomial reduction of *CIRCUIT SAT* to *3SAT* in *log (n)* memory, it is assumed that *CIRCUIT SAT* belongs to the class ***NP-complete*** problems.

The subsequent quoted statements are the result of research involving computational complexity and the complexity of logical networks. The complexity of the network is defined as the size of the network expressed by the number of gates of this network.

Fact 2.5: Polynomial networks [3, pp. 285-286].

The network has a polynomial size, if there is a family of networks $C=\{C_0, C_1, \ldots\}$ for which it is true, firstly, that the size of C_n is equal to at most $p(n)$ for some fixed polynomial p, and secondly, that for each combination of variables $x_i \in \{0,1\}$ $i=1,2, \ldots, n$ satisfying the formula represented by this network, the value of the network is *true* (in terms of the language of Turing machines we would say that for the input words that the machine accepts - the value of the network is *true*).

Fact 2.6: Any problem in class ***P*** has a polynomial network [3, p. 286].

Unfortunately, the reverse is not true. Not every polynomial network is a representation of a problem belonging to class ***P***.

Note 2.2: It has been shown that there exist unresolved problems that do have polynomial networks. This ensures that it cannot be concluded that ***P = NP***, even though the network of *C* defining the problem of *CIRCUIT SAT* (i.e. the network with variables when it should be noted that there is such a valuation of variables that the output gate returns *true*) must have a polynomial size so that the reduction can be done in polynomial time. And if so, *CIRCUIT SAT* would belong to the class ***P***. On the other hand, if *CIRCUIT SAT* is reduced to *3SAT*, and thus in terms of ***COMPLETENESS*** is not worse than *3SAT*, then also *3SAT* would fall into ***P***.

Combining polynomial networks with polynomial calculations is performed by introducing the concept of a monotone network.

Fact 2.7: Polynomial monotone networks [3, p. 287].

Family of networks $C=\{C_0, C_1, \dots\}$ is monotone if for the variables $x_i \in \{1\}$ $i=1, 2, \dots, n$ it is possible to construct a network C_n in the *log n* memory.

This leads to the conclusion that a monotone family of polynomial networks for any problem the family represents exists if and only if this problem belongs to class ***P***. Therefore, if for any problem ***NP*** - complete there is a monotone polynomial network, the resolution of the ***P*** *versus* ***NP*** issue takes the form ***P = NP***.

III. A 3SAT MONOTONE POLYNOMIAL NETWORK

Judging by the facts in the preceding section it can be argued that *3SAT* monotone networks exist. The scheme presented below consists of the following sequence:

- the various clauses in the record of *CNF* problem *3SAT* are elementary *3SAT* problems,
- for each elementary *3SAT* there is a universal network of fixed size (the number of gates),
- all-purpose networks representing elementary problems *3SAT* (clauses) are combined into a multi-output polynomial network which, after its enlargement by the network C_{ext} transforming a network to a single-output network, decides on *CNF* satisfiability for the *3SAT* formula.

The veracity of particular elements of the above scheme leads to the conclusion that there exists a monotone polynomial network for the problem *3SAT*.

3.1 Elementary *3SAT*

A logical function assigns a logical value of variables *y* to logical values of variables *x*, and is described formally by using logical expressions. Logical expressions can be written in many different forms, which are equivalent to each other.

Logical expressions are defined as follows:

- 0, 1, x_i i $\neg x_i$ are logical expressions, and where φ_1 and φ_2 are logical expressions, the $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$ are logical expressions;
- a logical expression can have only the form described above, but instead of x_i there may be other logical variables.

We know all the logical expressions describing the functions of two variables (there are 16 of them altogether). With their help we can describe any function of *n* variables. If the function of *i* variables is denoted by f^i, then $f^2(f^i(x_1, x_2, \ldots, x_i), x_{i+1}) = f^{i+1}$. Thus, for example, $f^3 = f^2(f^2(x_1, x_2), x_3)$, and because there can occur any variables, it may take the form of $f^3 = f^2(f^2(x_r, x_s), x_t)$.

If φ is a logical expression that describes the function of *n* variables in the form:

$$\varphi = f_1^3(x_r, x_s, x_t) \wedge f_2^3(x_r, x_s, x_t) \wedge \ \ldots \ \wedge f_m^3(x_r, x_s, x_t) \qquad (1)$$

where: $x_r, x_s, x_t \in \{x_1, x_2, \ldots, x_n\}$ and $r \in \{1, 2, \ldots n\}$, $s \in (r+1, r+2, \ldots, n\}$ and $t \in \{s+1, s+2, \ldots n\}$, ie. $r < s < t$,

then, leaving the designation of arguments of each function, it can be assumed that φ is a function of *m* variables, whose role shall be performed by the functions f_i^3, $i=1, 2, \ldots, m$.

$$\varphi = f^m(f_1^3, f_2^3, \ldots, f_m^3) = f^2(f^{m-1}(f_1^3, f_2^3, \ldots, f_{m-1}^3), f_m^3)$$

Expanding the final form of the expression, we finally get:

$$\varphi = f^2(f^2(\ldots f^2(f^2(f_1^3, f_2^3), f_3^3) \ldots, f_{m-1}^3), f_m^3) \qquad (2)$$

The resulting form of the expression shows that $f_1^3, f_2^3, \ldots f_m^3$ can be treated independently as arguments to the next (*m-1*) two-argument f^2 functions (they are all functions implemented by AND gates), to start with the most deeply nested.

In terms of logical networks, this corresponds to a situation in which $f_1^3, f_2^3, \ldots f_m^3$ are independent pieces of the network with *m* outputs, so that the output of *j-teenth* output resolves satisfiability of *j-teenth* function $f_j^3(x_r, x_s, x_t)$.

Fact 3.1 Elementary 3SAT

If we assume that the function φ described by the expression *(1)* is a normal product of the *CNF* corresponding to the definition of the problem *3SAT*, then the functions f_i $i=1, 2, \ldots, m$ are represented by clauses of three different variables, then each of the eight possible forms of the clauses of the three variables may be treated as an elementary *3SAT*.

In propositional calculus, the clause is defined as an alternative valuation of literals defining logical variables, and the conjunction of literals is called an implicant. Equivalent terms of these concepts in terms of elementary logic functions are an alternative conjunction and an elementary conjunction, respectively. The combination of eight definable clauses (elementary alternatives) and eight implicants (elementary conjunctions) of a function of three variables is presented in Table 1.

Table 1. Overview of elementary alternatives and conjunctions of a function of three variables.

Valuation of variables x_3	x_2	x_1	elementary alternatives (clauses)	elementary conjunctions (implicants)
0	***0***	***0***	$(x_3 \vee x_2 \vee x_1)$	$(\overline{x_3} \wedge \overline{x_2} \wedge \overline{x_1})$
0	***0***	***1***	$(x_3 \vee x_2 \vee \overline{x_1})$	$(\overline{x_3} \wedge \overline{x_2} \wedge x_1)$
0	***1***	***0***	$(x_3 \vee \overline{x_2} \vee x_1)$	$(\overline{x_3} \wedge x_2 \wedge \overline{x_1})$
0	***1***	***1***	$(x_3 \vee \overline{x_2} \vee \overline{x_1})$	$(\overline{x_3} \wedge x_2 \wedge x_1)$
1	***0***	***0***	$(\overline{x_3} \vee x_2 \vee x_1)$	$(x_3 \wedge \overline{x_2} \wedge \overline{x_1})$
1	***0***	***1***	$(\overline{x_3} \vee x_2 \vee \overline{x_1})$	$(x_3 \wedge \overline{x_2} \wedge x_1)$
1	***1***	***0***	$(\overline{x_3} \vee \overline{x_2} \vee x_1)$	$(x_3 \wedge x_2 \wedge \overline{x_1})$
1	***1***	***1***	$(\overline{x_3} \vee \overline{x_2} \vee \overline{x_1})$	$(x_3 \wedge x_2 \wedge x_1)$

3.2 A Universal 3 SAT Elementary Logical Network

For eight of the various clauses of the three variables there can be 255 functions defined (without constant ***TRUE***) that are expressed as a conjunction of clause combinations of one, two, etc., up to eight out of eight, and each of them be assigned a number. Every function, so defined, is expressed in a canonical conjunctive normal form (*CNF*).

By using a dual way of defining functions, owing to implicants, we obtain 255 functions (without constant ***FALSE***) that are expressed as an alternative of an appropriate combination of implicants, and then each function will be expressed in a canonical disjunctive normal form (*DNF*).

If constructing a logical network that represents functions of three variables, even if we confine ourselves to deal with 8 functions described by a single clause, is being implemented directly on the basis of expression clauses, then, although not complicated, they will vary in size. For example, for function $f^3(x_r, x_s, x_t) = (x_r \vee x_s \vee x_t)$ 3 OR gates are enough, but for the function $f^3(x_r, x_s, x_t) = (\neg x_r \vee \neg x_s \vee \neg x_t$, the network, in addition to the 3 OR gates, will also include 3 more NOT gates.

Because of the varied size of an elementary network *3SAT CNF*, it is not a favorable factor for the task of constructing a logical network for the whole *CNF 3SAT*.

It turns out that at the expense of increasing the size of the logical network, it is possible to operate a universal logical network that can represent any logic function of three variables. The functions can equally be defined in both *CNF* and *DNF* forms.

Such a universal network for the case of elementary *3SAT CNF* is a logical network that represents a constant function *TRUE*, and for the case of elementary *3SAT DNF*, it is a network representing the function of a constant *FALSE*. In both cases, the basis for constructing a network system is a setup of multiplexer 8/1.

The type of gates and the way of their combination in the logical network is defined by a scheme configuration, as presented in Figure 2.

The input signals of variables x_r, x_s, x_t, will be provided onto the address input system. Constant signals (vectors $U=[u_1, u_2, \ldots, u_8]$, interpreted as a binary number of the function) will be made onto the data input of the multiplexer. The output signal from OR gate is determined by the table of positions (Table 2), which clearly shows that the signal at the output corresponds to the signal being fed at the input, defined by the current state of the address inputs.

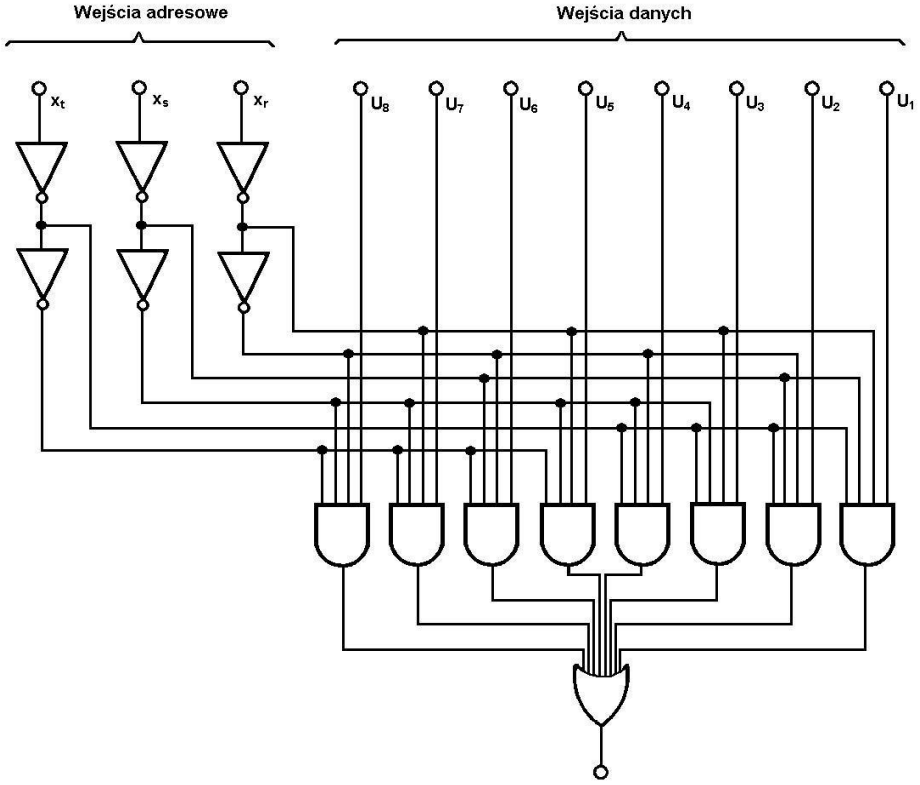

Figure 2. The setup of multiplexer 8/1

Table 2. The table of positions of the multiplexer 8/1

Address inputs	x_r	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*
	x_s	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*
	x_t	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*
Output data		u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8

A diagram of multiplexer 8/1, as shown in Figure 2, allows for the specification of the size corresponding to its logical network. Each four-input AND gate, occurring in the circuit, can be replaced by 3 two-input AND gates. Further, each eight-input OR gate can be replaced with 7 OR gates. Ultimately, the size of the logical network that reproduces the layout of the multiplexer will be the size of a total of 48 gates, including 3 input gates of variables, 8 constant input gates, 24 two-input AND gates, 7 two-input OR gates and 6 NOT gates.

Based on the preceding discussion of the logical network of elementary 3SAT, we suggest formulating the following assertion:

Fact 3.2*:* Any logical function *f* of three different variables x_r, x_s, x_t, described by the expression in one of the two canonical normal forms, conjunctive (CNF) or disjunctive (DNF), has a universal logical network of fixed size *k* gates. It takes the form of a multi-pole, as shown in Figure 3, which corresponds to the circuit diagram of Figure 1 c.

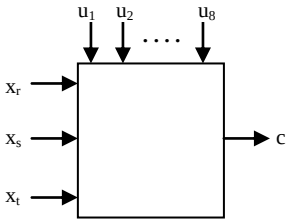

Figure 3 Circuit diagram of a universal network of a logical function of three variables.

3.3 The Design of a 3SAT Network

In order to examine all possible 255 functions as defined by the *CNF* expression, one would have to operate on 255 different vectors of constant $U=[u_1, u_2, \ldots, u_8]$.

Since we have assumed that we will treat functions described by expressions having the form of individual clauses as elementary *3SAT*, we can confine ourselves to examining eight functions defined by eight different clauses, and then while constructing the network, we will be using eight fixed vectors *U*, which are clearly determined by the form of clauses.

$$\boldsymbol{U} \quad = \quad \begin{cases} [0,1,1,1,1,1,1,1] & \text{if} \quad (x_3 \vee x_2 \vee x_1) \\ [1,0,1,1,1,1,1,1] & \text{if} \quad (x_3 \vee x_2 \vee \overline{x_1}) \\ [1,1,0,1,1,1,1,1] & \text{if} \quad (x_3 \vee \overline{x_2} \vee x_1) \\ [1,1,1,0,1,1,1,1] & \text{if} \quad (x_3 \vee \overline{x_2} \vee \overline{x_1}) \\ [1,1,1,1,0,1,1,1] & \text{if} \quad (\overline{x_3} \vee x_2 \vee x_1) \\ [1,1,1,1,1,0,1,1] & \text{if} \quad (\overline{x_3} \vee x_2 \vee \overline{x_1}) \\ [1,1,1,1,1,1,0,1] & \text{if} \quad (\overline{x_3} \vee \overline{x_2} \vee x_1) \\ [1,1,1,1,1,1,1,0] & \text{if} \quad (\overline{x_3} \vee \overline{x_2} \vee \overline{x_1}) \end{cases}$$

The individual vectors are constant *false* to their subsequent positions. In order to determine the number of positions in which they occur in the vector *U*, it is sufficient to interpret the negative literals in a clause as the digit "1" of the binary, for example $(\bar{x_3} \vee \bar{x_2} \vee x_1) \equiv 110_2 = 6_{10}$.

Note 3.1: It should be made clear that the determination of vectors *U* is directly affected only by the number of negative literals and it is indifferent which combination of three variables we deal with. This can be any combination of three variables allowed by the formula *(1)* defining *3SAT CNF*.

Note 3.2: If all constant inputs will be given signals *true* ($u_i=1$), then the considered multiplexer scheme performs the function *TRUE*. In turn, if all constant inputs are given signals *false* ($u_i=0$), the multiplexer performs the function F*ALSE*.

In view of the observations made in **Note 3.1** and **Note 3.2**, we assume that the network performing function *TRUE* for the three variables will become the basis for constructing an entire logical network *3SAT CNF*. In addition, saving the expression *(2)* in the form of operators, and using the prefix operator AND, we shall obtain a form that clearly defines how to design a C_{ext}. expansion network.

$$3SAT\ CNF = AND(AND(\ \ldots\ AND(AND(f_1^3, f_2^3), f_3^3) \ldots\ , f_{m-1}^3), f_m^3)$$

As a result, the design of the network *3SAT CNF* can be reduced to an *m*-fold replication of an elementary network *3SAT* represented by the network performing the function *TRUE* , and to establishing a constant *false* on an appropriate position of vector U_i $(i=1, 2, \ldots, m)$.

Assuming that the way of numbering address input gates and outputs of data is identical in each segment representing elementary *3SAT* network, and using the

designation of the multi-pole (Fig. 3), the resulting network can be represented as a circuit diagram, which is illustrated by Figure 4.

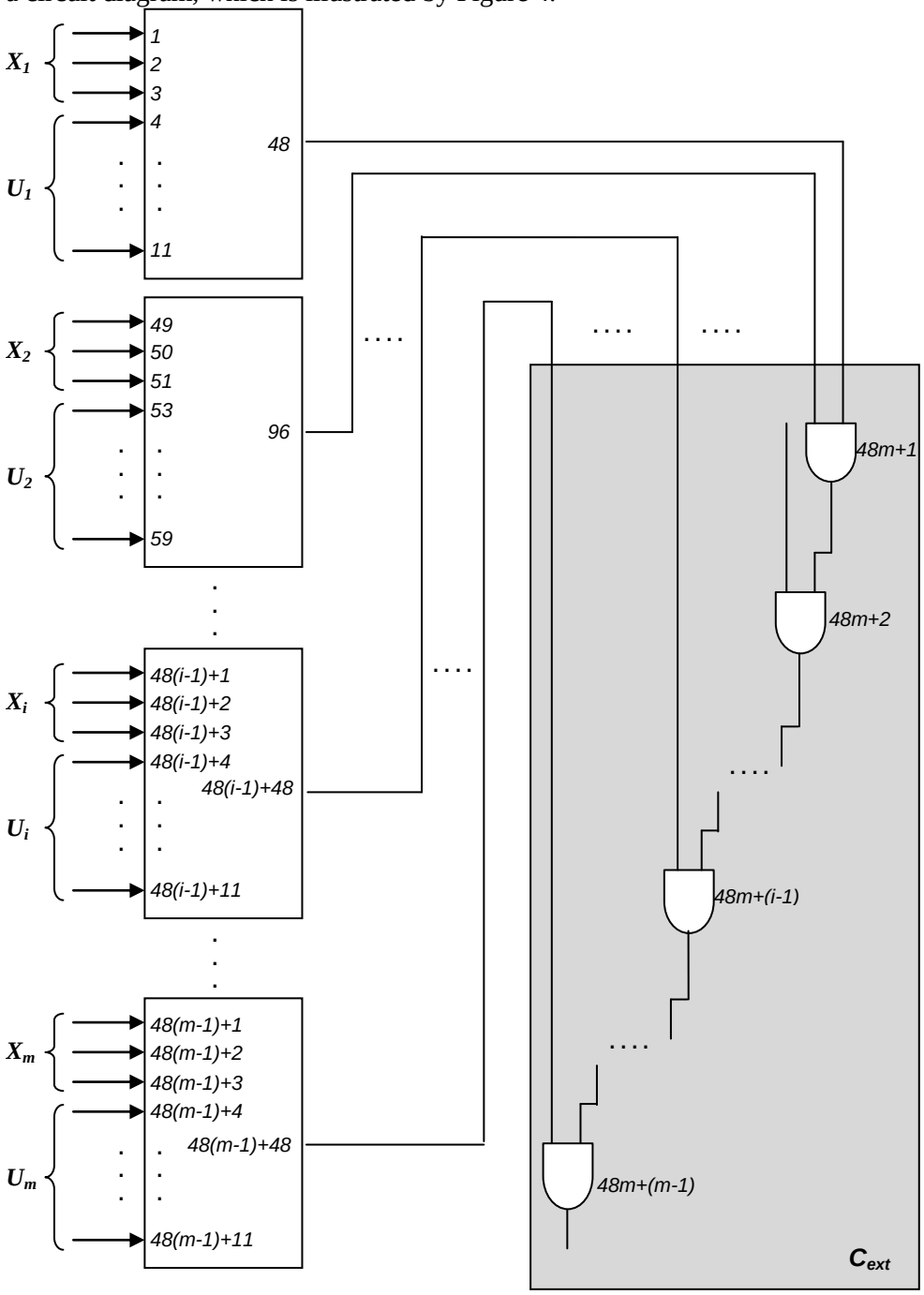

Figure 4. Circuit diagram of a 3SAT CNF logical network

The size of such a network constructed in terms of the *m* number of clauses written as *3SAT CNF* is linear and is equal to (*49· m-1*) goals.

To determine the size of the network with respect to *n* number of variables, one needs to take into account the case of *3SAT CNF* where clauses in all combinations of the three *n* variables can appear. The number of such combinations amounts to *(n-2) · (n-1) · n / 6*, and thus their number is expressed in a polynomial of the third degree.

Considering the fact that for each three combination of *n* variable there are eight different clauses, thus, in the worst case, the number of clauses in the formula *3SAT CNF* will reach *4/3 · · n (n-1) · (n-2)* at its most.

Note 3.3: In fact, the worst case can be limited to a situation in which in the formula *3SAT CNF* there is a maximum of seven clauses of the same combination of variables. This can easily be taken for granted based on the fact that if only for one combination of the three variables written as *CNF* there occurred eight clauses, the whole formula would not be satisfiable (*CNF* of eight clauses is defined as a function constant *FALSE*).

Consequently, the size of *3 SAT CNF* logical network constructed according to a scheme of multiplexer circuit 8/1 is not greater than *49 · 4/3 · · n (n-1) (n-2)* goals.

Thus, the network has a polynomial size. In addition, it is a network that is easy to use as a monotone network. In order to do this, it is sufficient that all the multiplexer address inputs *Xi (i = 1, 2, ..., m)* provide constant *true*. It has to be remembered, though, that the address inputs of multiplexers are given variables of *xr, xs, xt ∈ {x1, x2, ..., xn}* where *r ∈ {1, 2, ... n-2}, s ∈ (r 1, r 2, ..., n-1}* and *t ∈ {s 1, s 2, ... n}*, ie. *r <s <t*, appearing in various clauses, whereas providing valuations of *xi = true* is required by the definition of a monotone network. For the vector of valuations of variables *[x1, x2, ..., xn]* that specifies any other process of valuation of these variables, the network remains decisive.

It only remains to prove that the construction of the network can be done in *n log* memory. The process of duplication of *3SAT* elementary networks (multiplexer 8/1) requires one variable to remember the *m* number of clauses and one counter for the indexation of the subsequent *3SAT* elementary networks and their inputs. Yet another counter is needed to calculate the number of constant inputs which should be given a constant value f*alse*. Connections to respective gates require only direct operations on indices and they are, therefore, easy to perform in the logarithmic memory.

IV. SUMMARY

The hypothesis of the existence of monotone polynomial networks for the *3SAT CNF* problem, as formulated at the beginning, leads directly to the claim of equality of classes of ***P*** and ***NP*** problems.

To demonstrate the veracity of the given hypothesis has been an essential part of this research. It has been described that by using the duality of logical functions, it is possible to treat the conjunction of clauses, written in *3SAT CNF* , as a conjunction of elementary functions of three variables. It has been shown that using the elements of the theory of combinational circuits, it is possible to define a universal logical network for any elementary logic function of three variables. Such a universal network is characterized by a constant size expressed by the 48 logic gates AND, OR, and NOT. In particular, this applies to all of eight possible forms of clauses of three variables. It has been demonstrated that the task of such a universal logical network can be performed by a network constructed on the basis of a multiplexer circuit 8/1.

Furthermore, it has been shown that the logical network of complete *3SAT CNF*, in the worst case, is characterised by the size expressed by $O(n^3)$. And finally, it has been demonstrated that such a network can be constructed in the logarithmic memory, and that it can be a monotone one.

Choosing a *3SAT* problem to demonstrate the existence of a monotone polynomial network is not accidental[2]. First of all, the problem of *SAT* satisfiability is of primary importance for analysis in terms of *COMPLETENESS*. All results in relation to it, therefore, refer to both ***NP - complete*** problems (*3SAT* problem) as well as the ***P - complete*** problems (2SAT problem). Moreover, a *3SAT* problem was chosen in order to avoid difficulties similar to those that occurred in the process of verification and checking of the solution to the problem of **KNAPSACK**.

It cannot be denied that the description and analysis of the network structure (and the whole family of networks) is correct unless (1) firstly, the definition of the family of monotone network is questioned, including the condition that each network of the family was constructed in the *n log* memory, and that it was decisive for the input words, such that $xi = 1$ for each $i = 1, ..., n$, where n is as

[2] In subsequent papers prepared for publication the construction of a monotone polynomial network for the problem MULT (n) will be described.

defined in [3, p. 285][3] a length of the word describing a method of how to value the variables of the 3SAT formula, mapped by a logical network, and not, as in [3, p.43][4], the length of the word describing the problem in terms of a Turing machine; (2) and secondly, it is challenged that a Turing machine model and a logical network model are equivalent, and an error is pointed out in the theorem quoted in the introduction that a complexity of logical networks is combined with the complexity of time logic [1, pp. 401-405].

In fact, the C_n logical network design is nothing else but a polynomial reduction of the *3SAT CNF* formula *φ (x1, x2, ..., xn*) to the problem of *CIRCUIT VALUE* as is the reduction of any language $L \varepsilon P$ to *CIRCUIT VALUE*, described in [3, pp. 184-186]. In both cases, it is a process of duplicating of the identical elementary networks and then connecting the corresponding input and output goals.

Apart from the fact that structurally a *3SAT* polynomial network does not differ from a *2SAT* network (in 2SAT networks multiplexers 8/1 are replaced by multiplexers 4/1), yet another aspect of a monotone polynomial network is worth mentioning. The fact that for a *3SAT* a logical network of a polynomial number of goals can be constructed paves the way for "daring" attempts to construct hardware implementation for an algorithm to *2SAT* and *3SAT* solutions; and "even more daring" attempts to develop algorithms for poly-logarithmic time paralleled with total polynomial work. And if these attempts turned out a success, it would solve another interesting issue, namely ***NC*** (Nick's Class) *versus* ***P***.

Having demonstrated that there exists a monotone network for *3SAT CNF* is equivalent to the claim that there exists a polynomial algorithm for *3SAT CNF*. An attempt to design such an algorithm, regardless of its practical importance, may be looked upon as part of the verification of the above findings.

REFERENCES

[1] Sipser, Michael. 2009. Introduction to the Theory of Computation. WNT Warsaw
[2] Traczyk, Wiesław. 1986. Digital Circuits. Theoretical Basis and Methods of Synthesis. WNT Warsaw
[3] Papadimitriou, Christos. 2007. Computational Complexity. WNT Warsaw

[3] "... we know that a logical network of *n* input variables can compute any logical function of *n* variables. Equivalently, we may think that the network accepts some words of the *n* length of *{0, 1} **, and rejects the others. In this context, the words *x = x1 ... xn ε {0, 1} ** are treated as a valuation of input variables, ... "

[4] "To solve this problem by using a Turing machine, we must first decide how we will write (represent) the example by means of words."