Transformation of Boolean Expression into Disjunctive or Conjunctive Normal Form

Patrik Rusnak

***Abstract*—**Reliability is an important characteristic of many systems. One of the current issues of reliability analysis is investigation of systems that are composed of many components. Structure of such systems can be defined in the form of a Boolean function. A Boolean function can be expressed in several ways. One of them is a symbolic expression. Several types of symbolic representations of Boolean functions exist. The most commonly known are disjunctive and conjunctive normal forms. These forms are often used not only in reliability analysis but also in other fields, such as game theory or logic design. Therefore, it is very important to have software that is able to transform any kind of Boolean expression into one of these normal forms. In this paper, an algorithm that allows such transformation is presented.

***Keywords*—**reliability, Boolean function, normal forms.

I. Introduction

Investigation of system reliability is a complex problem consisting of many steps. One of them is creation of a model of the system. Since the system is usually composed of several components, a special map defining the dependency between operation of the components and operation of the system has to be known. This map is known as structure function and, for a system consisting of n components, it has the following form [1]:

$$\phi(x_1, x_2, \dots, x_n) = \phi(\boldsymbol{x})\colon\ \{0,1\}^n \to \{0,1\}, \tag{1}$$

where set $\{0,1\}$ is a set of possible states at which the components and the system can operate (state 1 means that the component/system is functioning while state 0 agrees with a failure of the component/system), x_i is a variable defining state of the i-th system component, for $i = 1,2,\dots,n$, and $\boldsymbol{x} = (x_1, x_2, \dots, x_n)$ is a vector of components states (state vector).

Based on the properties of the structure function, two classes of systems can be recognized – coherent and noncoherent. A system is coherent if its structure function is monotonic, i.e. there are no circumstances under which a failure of any system component can result in a repair of the system. If this condition is not met, the system is noncoherent.

Most of the systems studied by reliability engineers are coherent and, therefore, a lot of methods of reliability analysis are based on the assumption that the structure function is monotonic. Typical examples are methods of importance analysis [2], which focuses on ranking the components with respect to their influence on the system operation. However, real noncoherent systems also exist. Some of them are k-to-l-out-of-n systems that are working if at least k but not more than l components are working [3, 4] or logic circuits [5]. Reliability analysis, especially importance analysis, of such systems requires development of new methods that will take the incoherencies into account. One of the possible ways is to use tools related to the analysis of Boolean function. One of such tools is logical differential calculus.

Logical differential calculus allows investigating dynamic properties of Boolean functions [6]. Boolean derivative is the central term of this tool. Several types of Boolean derivatives exist but, for the purpose of reliability analysis, the most important one is Direct Partial Boolean Derivative (DPBD) [7]. For a Boolean function $\phi(\boldsymbol{x})$, it is defined as follows [6, 7]:

P. Rusnak, University of Zilina, Faculty of Management Science and Informatics, Zilina, Slovakia (e-mail: pattrik.rusnak@gmail.com).

$$\partial\phi(j \to \bar{j})/\partial x_i(s \to \bar{s}) = \begin{cases} 1, & \text{if } \phi(s_i, \boldsymbol{x}) = j \text{ AND } \phi(\bar{s}_i, \boldsymbol{x}) = \bar{j} \\ 0, & \text{otherwise} \end{cases}, \quad \text{for } s, j \in \{0,1\}, \tag{2}$$

where $(s_i, \boldsymbol{x}) = (x_1, x_2, \dots, x_{i-1}, s, x_{i+1}, \dots, x_n)$. In reliability analysis, this derivative can be used to find situations in which a failure/repair of a given system component results in the failure/repair of the system. Quantification of such situations can be performed to rank importance of the system components [7].

Definition (2) implies four different DPBDs with respect to variable x_i exist. For coherent systems, only DPBDs $\partial\phi(1 \to 0)/\partial x_i(1 \to 0)$ and $\partial\phi(0 \to 1)/\partial x_i(0 \to 1)$ are relevant since there exist no situation in which a failure (repair) of a component can result in system repair (failure). However, this is not true for noncoherent systems and, therefore, DPBDs $\partial\phi(1 \to 0)/\partial x_i(0 \to 1)$ and $\partial\phi(0 \to 1)/\partial x_i(1 \to 0)$ have to be taken into account too [8].

DPBDs can be computed numerically (if the analyzed function is defined by the truth table) or symbolically (if the function is defined by a symbolic expression). Numerical calculation can be implemented using the computer easily. However, it can be applied only to functions of few variables since defining a big Boolean function using the truth table requires a huge amount of memory. This problem can be solved using the symbolic representation. This solution requires creation of complex software that is able to manipulate with different forms of symbolic expressions. Development of such tool has been considered in [9].

The software described in [9] will implement methods needed for reliability analysis. The methods should be based primarily on symbolic manipulation with Boolean (and also with multiple-valued logic) functions. The principal part of the software is a parser that is able to transform a symbolic expression into the form of a multi way tree that can be processed on the computer much more easily than the original string representing the symbolic expression. Normal forms [10] are one of the most commonly used Boolean expressions. In order to work with them, the software has to be able to transform various kinds of Boolean expressions into a specific normal form. In this paper, an algorithm for such transformation is presented. The algorithm assumes that a Boolean expression is represented in the form of a multi way tree. After applying the algorithm, we obtain a new tree that will agree with one of the normal forms that are used in Boolean algebra.

II. Boolean Normal forms

Boolean function can be represented in many forms, and one of them is a symbolic form. A Boolean function can be expressed in a number of symbolic forms, some of which have a specially defined format and are called normal forms [10]. One of the normal forms is a disjunctive normal form or a sum of products. This form consists of elementary conjunctions. An elementary conjunction of n Boolean variables $x_1, x_2, \dots, x_n$ has the following form [11]:

$$P = \bigwedge_{i \in \mathbb{A}} x_i \wedge \bigwedge_{j \in \mathbb{B}} \bar{x}_j \text{, where } \mathbb{A} \cup \mathbb{B} \subseteq \{1,2,\dots,n\} \text{ and } \mathbb{A} \cap \mathbb{B} = \emptyset. \tag{3}$$

Examples of elementary conjunctions are expressions such as $1, \bar{x}_1, x_2, x_1 \wedge \bar{x}_2, \bar{x}_1 \wedge x_2 \wedge x_3$.

Logical expression $\phi(x_1, x_2, \dots, x_n)$ that has the following form [11]:

$$\phi(x_1, x_2, \dots, x_n) = \bigvee_{k=1}^{l} P_k = \bigvee_{k=1}^{l} \left(\bigwedge_{i \in \mathbb{A}_k} x_i \wedge \bigwedge_{j \in \mathbb{B}_k} \bar{x}_j \right), \tag{4}$$

is then known as Disjunctive Normal Form (DNF) composed of l elementary conjunctions denoted as P_k for $k = 1,2,\dots,l$. Examples of logical expressions in DNF are expressions such as $\phi(x_1,x_2,x_3) = x_1 \vee x_2 \wedge \bar{x}_3, \phi(x_1,x_2) = \bar{x}_1 \wedge x_2 \vee x_1 \wedge \bar{x}_2, \phi(x_1,x_2,x_3) = x_1 \wedge x_2 \vee \bar{x}_1 \wedge \bar{x}_2 \vee x_1 \wedge x_2 \wedge \bar{x}_3$.

Another commonly used form is a conjunctive normal form also called product of sums. This form consists of elementary disjunctions (they are known as clauses). For n Boolean variables $x_1, x_2, \dots, x_n$, an elementary disjunction is defined as follows [11]:

$$S = \bigvee_{i\in\mathbb{A}} x_i \wedge \bigvee_{j\in\mathbb{B}} \bar{x}_j \text{, where } \mathbb{A} \cup \mathbb{B} \subseteq \{1,2,\dots,n\} \text{ and } \mathbb{A} \cap \mathbb{B} = \emptyset. \tag{5}$$

Examples of elementary disjunctions are expressions such as $0, \bar{x}_1, x_2, x_1 \vee \bar{x}_2, \bar{x}_1 \vee x_2 \vee \bar{x}_3$.

Logical expression $\phi(x_1,x_2,\dots,x_n)$ that has the following form [11]:

$$\phi(x_1,x_2,\dots,x_n) = \bigwedge_{k=1}^{l} S_k = \bigwedge_{k=1}^{l}\left(\bigvee_{i\in\mathbb{A}} x_i \wedge \bigvee_{j\in\mathbb{B}} \bar{x}_j\right), \tag{6}$$

is Conjunctive Normal Form (CNF), where S_k is the k-th elementary disjunction for $k = 1,2,\dots,l$. Examples of logical expressions in CNF are expressions such as $\phi(x_1,x_2,x_3) = (x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_3), \phi(x_1,x_2) = (x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2), \phi(x_1,x_2,x_3) = x_1 \vee \bar{x}_2 \vee x_3$.

These normal forms find their usage in a number of fields, such as game theory, artificial intelligence, logic programming, logic design, and reliability analysis. For example, a special DNF, called the Horn clause (condition), is used in proposition logic and logic programming [11], DNF and CNF are used in the area of logic circuits to layout the logic gates from the logic function of circuit [12], or DNF is used in the reliability analysis, specifically in symbolic fault-tree analysis [13].

III. Transformation to normal forms

The previous text indicates that normal forms are important in a number of fields. Therefore, it is necessary for the software tool introduced in [9] to transform the expressions of Boolean functions into these basic forms.

Before the transformation of the stored Boolean function into DNF or CNF is performed, it is firstly necessary to ensure that Bool's negation operation (NOT) is only in front of variables or constants. For this purpose, it is necessary to transform the Bool operation that is the operand of the NOT operation. This will transform the operation into its equivalent that is not preceded by operation NOT [14]. The specific transformations of the Boolean operations used in the software tool can be seen in Table I, where a and b represent operands of the Boolean operations.

It is also necessary to ensure that Boolean operations such as exclusive or (XOR), equivalence (EQV), Sheffer stroke (NAND) and Peirce's arrow (NOR) will be replaced with equivalent expressions containing only basic Bool operations, thus AND, OR and NOT [15]. All transformations can be seen in Table II, where a and b represent the operands of Boolean operations.

TABLE I TRANSFORMATION OF NOT OPERATION

Before transformation	After transformation
NOT(NOT a)	a
NOT(a OR b)	a NOR b
NOT(a AND b)	a NAND b
NOT(a NOR b)	a OR b
NOT(a NAND b)	a AND b
NOT(a EQV b)	a XOR b
NOT(a XOR b)	a EQV b

TABLE II TRANSFORMATION OF BOOLEAN OPERATIONS

Before transformation	After transformation
a NAND b	NOT a OR NOT b
a NOR b	NOT a AND NOT b
a EQV b	(a AND b) OR (NOT a AND NOT b)
a XOR b	(a AND NOT b) OR (NOT a AND b)

After performing all of the previous transformations, we get a logical expression that characterizes a function and contains only basic Boolean algebra operations AND, OR and NOT. Also, NOT operation is located just above variables and constants, becuase the transformation are performed in top-down manner. An example of the transformation process for logical expression $(1 \text{ NAND } x_1) \text{ XOR } (0 \text{ NOR } x_2)$ can bee seen on Fig. 1, where red circles depict unwanted nodes for DNF or CNF.

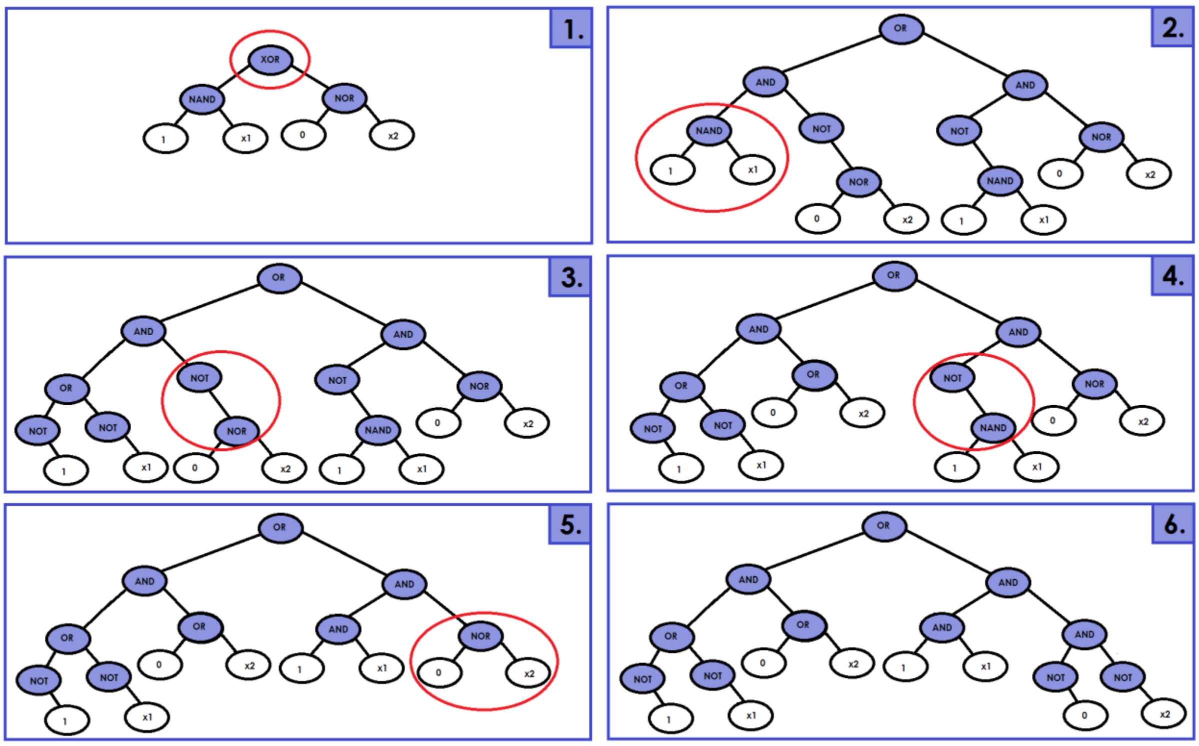

Fig. 1 Illustration of the transformation process

In order to ensure that the expression will be in DNF (i.e. disjunctions of elementary conjunctions) or CNF (i.e. conjunctions of elementary disjunctions), it is necessary to perform the transformation of the arrangement of AND and OR operations [14]. In total, there are three cases needed to be addressed by the relevant transformation. It is also needed to point out that all cases are for DNF, but if the AND operation is changed to OR operation and vice versa, these cases are also usable for CNF.

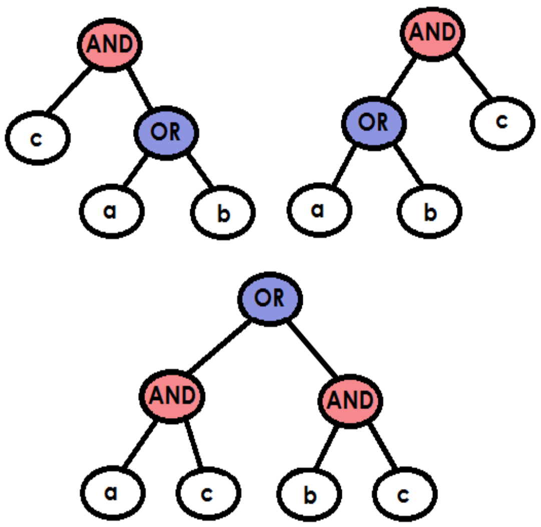

Fig. 2 Case one and its transformation

The first case corresponds to situation c AND $(a$ OR $b)$, or through the commutativity of the operation AND also to the situation$(a$ OR $b)$ AND c. In this case, it is necessary to multiply the operands a and b of the OR operation with another operand c of the AND operation based on the distribution law of the Boolean algebra. This means that the resulting expression $(a$ AND $c)$ OR $(b$ AND $c)$ will be correct for DNF [14].

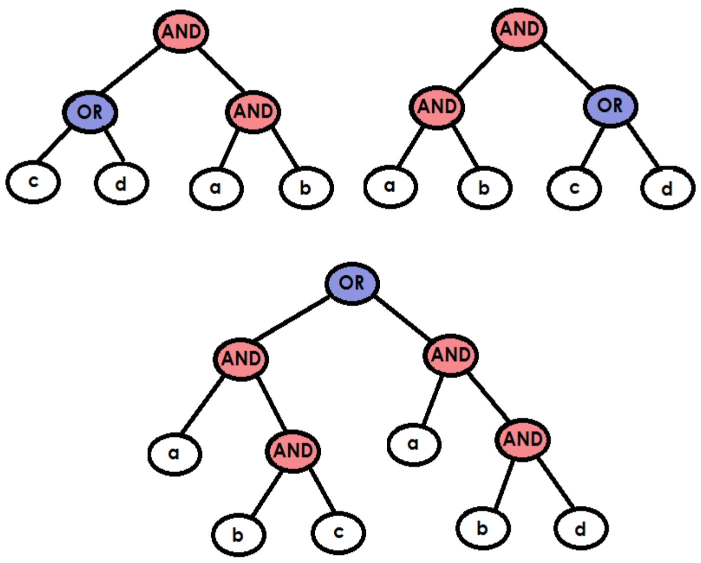

Fig. 3 Case two and its transformation

The second case corresponds to the occurrence of situation $(c$ OR $d)$ AND a AND b, possibly due to the commutativity of the AND operation also the occurrence of situation

a AND b AND$(c$ OR $d)$. In this case, it is necessary to multiply the operands c and d of the OR operation with operands a and b of the AND operation based on the distribution law. This means that the obtained expression $(a$ AND b AND $c)$ OR $(a$ AND b AND $d)$ will already be correct for DNF [14].

Finally, the third case corresponds to situation $(a$ OR $b)$ AND $(c$ OR $d)$. In this case, it is necessary, on the basis of the distribution law, to multiply the individual operands of OR operations between themselves. After multiplication, the created expression $(a$ AND $c)$ OR $(a$ AND $d)$ OR $(b$ AND $c)$ OR $(b$ AND $d)$ will already be correct for DNF.

After performing all the transformations for the symbolic expression of the Boolean function, the result expression is in DNF or CNF. This transformation process is very intuitive and easy to implement, so it was chosen for a software tool.

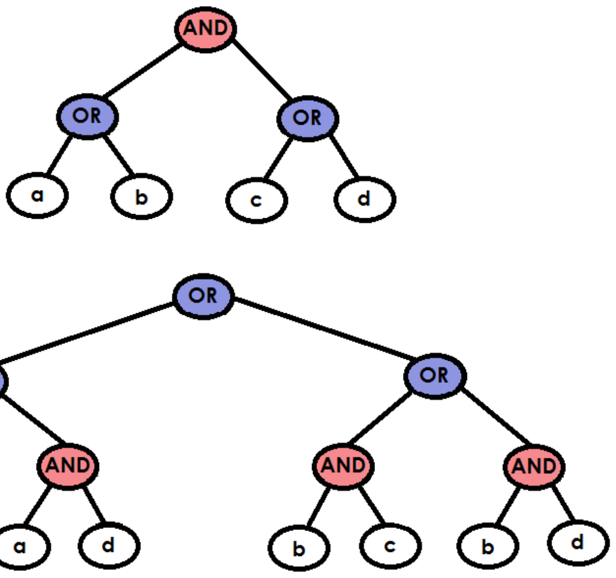

Fig. 4 Case three and its transformation

IV. Conclusion

Boolean functions are very powerful tool, which can be used in many fields, such as game theory, propositional theory, logic programming and reliability analysis. They can be represented by many forms. One of them is symbolic expression, which can be easily read by humans but not computers. The special symbolic expressions, called normal forms, consist only of specific Boolean operations and are used in a number of fields. However, retrieving these forms from a symbolic expression of Boolean function may be hard and time consuming. Because of that, the software introduced in [9] was extended by other functionalities that allow obtain normal forms, specifically DNF and CNF, from any symbolic representation of Boolean function.

In order to receive Boolean normal forms, a transformation process is needed. In this paper, we presented the transformation of symbolic expressions of a Boolean function by replacing Boolean operation into a form that contains only basic Boolean algebra operations AND, OR, and NOT, which is located just in front of variables or constants, and by transforming three specific cases described above that are not valid for DNF or CNF. This transformation process is intuitive and also effective for tree structure in which the symbolic expression is stored in the software.

References

[1] M. Rausand and A. Høyland, *System Reliability Theory*, 2nd ed. Hoboken, NJ: John Wiley & Sons, Inc., 2004.

[2] W. Kuo and X. Zhu, *Importance Measures in Reliability, Risk, and Optimization: Principles and Applications*. Chichester, UK: Wiley, 2012.

[3] S. J. Upadhyaya and H. Pham, "Analysis of noncoherent systems and an architecture for the computation of the system reliability," *IEEE Transactions on Computers*, vol. 42, no. 4, pp. 484–493, Apr. 1993.

[4] M. Kvassay, E. Zaitseva, V. Levashenko, and J. Kostolny, "Binary decision diagrams in reliability analysis of standard system structures," in *2016 International Conference on Information and Digital Technologies (IDT)*, 2016, pp. 164–172.

[5] M. Kvassay, E. Zaitseva, V. Levashenko, and J. Kostolny, "Reliability analysis of multiple-outputs logic circuits based on structure function approach," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 3, pp. 1–1, Mar. 2016.

[6] S. N. Yanushkevich, D. M. Miller, V. P. Shmerko, and R. S. Stankovic, *Decision Diagram Techniques for Micro- and Nanoelectronic Design Handbook*, vol. 2. Boca Raton, FL: CRC Press, 2005.

[7] E. N. Zaitseva and V. G. Levashenko, "Importance analysis by logical differential calculus," *Automation and Remote Control*, vol. 74, no. 2, pp. 171–182, Feb. 2013.

[8] M. Kvassay, E. Zaitseva, J. Kostolny, and V. Levashenko, "Reliability analysis of noncoherent systems based on logical differential calculus," in *Risk, Reliability and Safety: Innovating Theory and Practice*, Taylor & Francis Group, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742: CRC Press, 2017, pp. 1367–1374.

[9] P. Rusnak "Parser of Input Data in Reliability Analysis based on Logical Differential Calculus" *CERes Journal*, vol. 2, pp. 11-16, Dec. 2016.

[10] S. E. Whitesitt, *Boolean Algebra and Its Applications*, Courier Corporation, 2012.

[11] A. Horn, "On sentences which are true of direct unions of algebras," *Journal of Symbolic Logic*, vol. I, no. 16, pp. 14–21, 1951.

[12] Y. Crama and P. L. Hammer, *BOOLEAN FUNCTIONS - Theory, Algorithms, and Applications*. Cambridge University Press, 2011.

[13] U. Niessen-Gillhaus, W. Schneeweiss, "A practical comparison of several algorithms for reliability calculations," *Reliability Engineering & System Safety*, vol. 31, pp. 309-319 1991.

[14] A. Ligeza, *Logical Foundations for Rule-Based Systems*. Springer, 2006.

[15] S.T. Karris, *Digital Circuit Analysis and Design with Simulink Modeling and Introduction to CPLDs and FPGAs*. Orchard Publications, 2007.