Graphic Tool for Learning and Application of Decision Diagrams in Reliability Analysis

Lukáš Čajka, Martin Belvončík

***Abstract*—**In this article, we present a tool for manipulation with binary decision diagrams. This tool allows applying binary decision diagrams in reliability engineering, i.e., it allows expressing the structure function of the system in the form of a decision diagram and calculate availability of the whole system. The application's role is to make a binary decision diagram based on the structure function, draw it on the screen and calculate availability of the system. On the screen it is possible to move with individual shapes and modify them, e.g., it is possible to change the size, color, and fonts of individual shapes. The created diagram can also be exported to PNG or SVG file format. The standard input for the application is a structure function defined in PLA format.

***Keywords*—**Boolean function, structure function, reliability, binary decision diagram

I. INTRODUCTION

The main objective of reliability as a science is to qualitatively and quantitatively assess the ability of an object to perform and maintain its functions under different conditions. Simultaneously, one of the main goals of reliability analysis is to increase the reliability of the system. For this purpose, it is necessary to [1]:

- find an appropriate representation and model of the system,
- quantify the created model,
- appropriately model and quantify uncertainties that occur in system behavior.

The reliability theory focuses on forecasting, estimating and optimizing product performance. The product can be viewed as a system composed of components. The relationship between the functionality of the components and the functionality of the system is represented by a structure function.

The structure function is used for the mathematical description of the system and describes the system's performance depending on states of its components. For a system composed of n components, this function has n variables that have two possible states (functioning/failure). The structure function then looks like this [2]:

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

where x_i is a variable representing state of component i, where $i \in \{1,2,\dots,n\}$, and vector $\boldsymbol{x} = (x_1, x_2, \dots, x_n)$ is known as a state vector defining states of all the system components.

Structure function defines system topology. However, knowledge of this function is not enough to perform complex reliability analysis. This task also requires knowledge of the state probabilities of the system components. With this information, we can investigate several reliability measures of the system, such as availability (the probability that the system is in state 1) or unavailability (the probability that the system is in state 0), or analyze importance of the system components [3].

It is a problem or a question of how to represent systems that have many components and how to efficiently express their structure functions. One of the possible solutions to this problem is application of approaches of Boolean logic in reliability analysis. This results from a fact that the structure function can be viewed as a Boolean function [4].

L. Čajka, Faculty of Management Science and Informatics, University of Žilina in Žilina, Slovakia (e-mail: cajka.lukas88@gmail.com).
M. Belvončík, Faculty of Management Science and Informatics, University of Žilina in Žilina Slovakia (e-mail: martin.belvo@gmail.com).

II. Representation of Boolean Function

There exist several ways of how to represent a Boolean function: the truth table, logic expression and Binary Decision Diagram (BDD) [5, 6]. Each of them has some pros and cons.

A. Logic Expression

Logic expression consists of logic constants, variables and functions and logic operators (AND, OR, XOR, NOT). The result of the logic expression is one of the values true and false, which are usually represented as numbers 1 and 0 respectively. Examples of logic expressions are conjunctive normal form or disjunctive normal form [7], such as:

$$(\overline{x_1} \wedge x_2 \wedge x_3) \vee (x_1 \wedge \overline{x_2} \wedge x_3) \vee (x_1 \wedge x_2 \wedge \overline{x_3}) \vee (x_1 \wedge x_2 \wedge x_3), \quad (2)$$

where $\wedge$ denotes logical conjunction (AND), $\vee$ agrees with logical disjunction (OR), and $\overline{\ }$ denotes logical complement (NOT). This expression defines a Boolean function of 3 variables that takes value 1 if not more than one Boolean variable has value 0.

Logic expressions are very convenient for symbolic manipulation, but they are not the best choice for computer implementation and manipulation with Boolean functions because their processing on a computer can be quite complicated.

B. Truth Table

Truth table (Table I) is another typical representation of a Boolean function. This table enumerates all possible combinations of values of the variables of a Boolean function and, for each of them, it contains a corresponding value of the Boolean function. It is very simple for computer implementation and can be processed on a computer very fast. However, its main drawback is a high memory consumption since the size of the table grows exponentially with increasing number of variables (clearly, if a Boolean function has n, then the truth table has to contain 2^n rows). Because of that it can be used just for small functions containing not more than about 30 variables.

Table I Truth table defining the same function as logic expression (2)

x_1	x_2	x_3	$f(\boldsymbol{x})$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

C. Binary Decision Tree

Binary decision tree is a binary tree that has individual variables in the inner nodes, and the decision-making variables in the leaves. Let us suppose we have Boolean variables $x_1, x_2, \ldots, x_n$ that make up the input for the function. We test one of the variables in the root, for example x_1 and we have two subtree, one for the case where $x_1 = 0$ and the other where $x_1 = 1$. Each of the two subtrees now tests another variable, each with two other subtrees, etc. On leaves we have either 0 or 1, which are the outputs of the function expressed by a binary decision tree.

An example of a binary decision tree for function defined by logic expression (2) is in Fig. 1. The solid green lines in this figure agree with situations when a decision variable (Boolean variable) takes value 1 and the red dashed lines with situations when the variable has value 0. Every path from the root (it contains variable x_1) to a leaf agrees with one combination of

values of the Boolean variables x_1, x_2, and x_3, and the value in the leaf at the end of the path agrees with the value of the function for this combination.

As one can see, binary decision tree representing a Boolean function of n variables consists of $2^{n+1} - 1$ nodes ($2^n - 1$ inner nodes and 2^n decision-making nodes (leaves)) [5, 6]. This implies its demands on memory are usually greater than demands of the truth table. Because of that, a binary decision tree is not a good data structure for representation of Boolean functions containing a lot of variables. However, it can be reduced in a special form, which is known as a BDD.

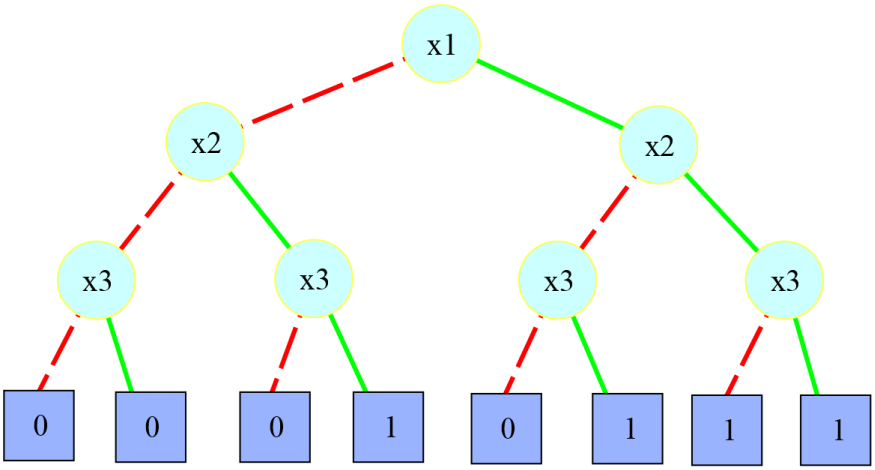

Fig. 1 Binary decision tree defining the same function as logic expression (2)

D. Binary Decision Diagram

BDDs differ from binary decision trees in two directions. Firstly, they allow omitting redundant decision nodes, i.e., nodes that have no influence on the result, e.g., the leftmost node linked with Boolean variable x_3 in Fig. 1. The second improvement is to allow sharing of the same subtrees, e.g., the subtrees defined by two middle nodes linked with Boolean variable x_3 in Fig. 1. This two benefits are achieved by iterative application of two reducing rules, which are known as *elimination of redundant nodes* (Fig. 2) and *merging of isomorphic nodes* (Fig. 3) [5, 6]. For example, if we iteratively apply these two rules on a binary decision tree depicted in Fig. 1, then we obtain a BDD depicted in Fig. 4. As one can see, the diagram is more compact, and it has much less nodes than the tree.

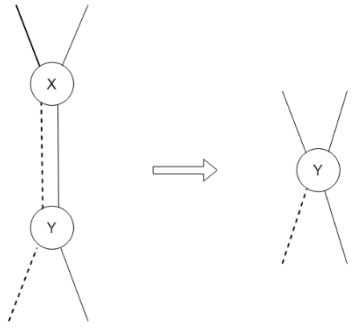

Fig. 2 Rule of elimination of redundant nodes (according to [5])

According to the previous description, a BDD is a quite good choice for representation of Boolean functions because its memory demands are much less than demands of the truth table or binary decision tree, and it can be processed on a computer easier than logic expressions. Thanks to these properties they can be used as an underlying data structure for storing the structure function of systems composed of a huge amount of components. Furthermore, they

can also be used in quantitative reliability analysis because they allow computing system availability and unavailability, which are one of the basic characteristics of systems from reliability point of view, in an efficient way. However, for this purpose, one more information has to be added to the diagram. This information agrees with the state probabilities of the components. In this case, we obtain a new structure, which is known as a probabilistic BDD.

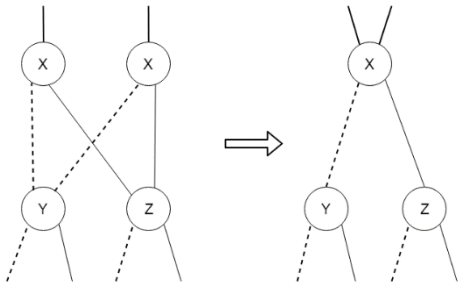

Fig. 3 Rule of merging of isomorphic nodes (according to [5])

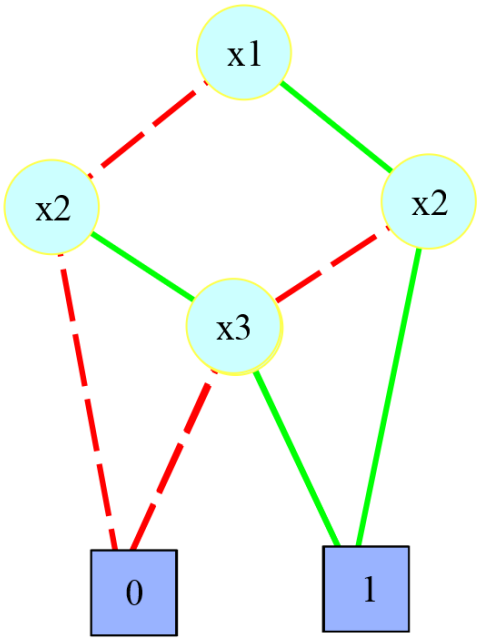

Fig. 4 Binary decision diagram obtained from binary decision tree depicted in Fig. 1

III. Probabilistic Binary Decision Diagram

A BDD is a graphical structure developed for representation of Boolean functions. Since a structure function can be interpreted as a Boolean function, BDD can be used in reliability analysis to represent a structure function. A special type of BDD is a probabilistic BDD [4]. In this kind of BDD, each outgoing edge from an inner node carries additional information that defines the probability that the component (whose behavior is modeled by random variable x_i) associated with the inner node is in a state associated with the outgoing edge.

A probabilistic BDD can be used to calculate the probability of the system state. The probability that the system is in state 0 (does not work) or 1 (works) is simply computable by traversing the BDD and identifying all paths ending in a leaf labeled by number 0 or 1 respectively [4]. This algorithm can be described based on [8] in the following way.

After creating the diagram, for each internal node, we set the probability that a component associated with the node is working. Then we traverse the diagram from the root to leaves and compute the probability that we reach a leaf. At first, we set the probability that we reach the root. This probability agrees with value 1. Then we set the probability of reaching nodes that

are sons of the root. For each node, this probability is set as the product of the probability that we are in the root and the probability that we go from the root to the node. The probability that we go from the root to the specific node agrees with the probability that the component associated with the root is working if we move through the edge associated with state 1 of the component or with the probability that the component is not available if we leave the root by the edge associated with state 0 of the component. When we process all sons of the root, then we move to one of its sons and repeat the previous procedure for this node. After that, we move to another son and repeat the procedure. After processing all nodes at one level, we move to the next level and process all nodes at it. If we visit one node more times (i.e., if a node has more than one parent), we sum all the probabilities of reaching the node. This procedure is repeated until we process all the nodes of the diagram using level order traversal. At the end, the probability that the system is working (available) will be in the leaf denoted by number 1 and the probability that the system is unavailable in the leaf labelled as 0.

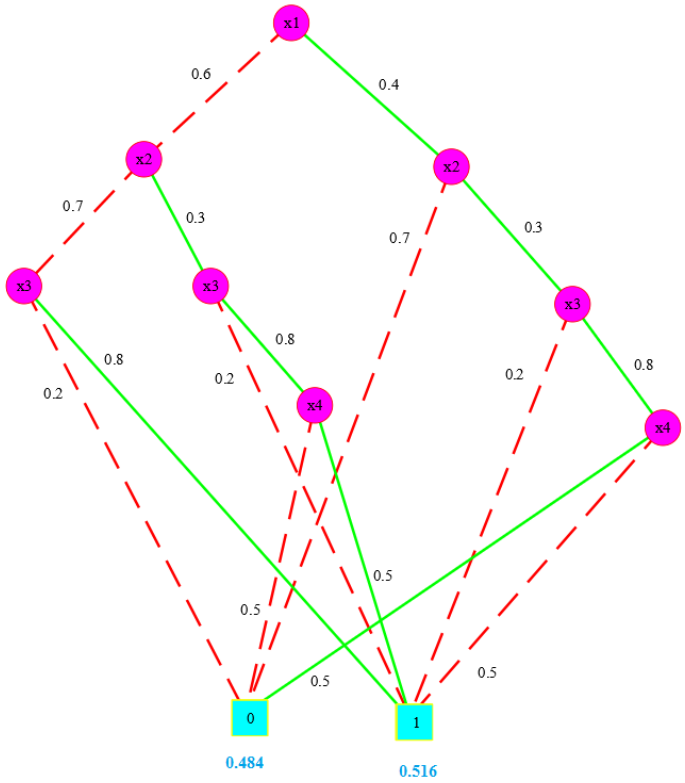

Fig. 5 Example of probabilistic binary decision diagram

IV. Software for Manipulation with Binary Decision Diagrams

BDDs are useful data structure for representation of Boolean functions of high dimensions. They are also useful in reliability analysis of systems composed of a lot of components because they allow representing their structure function in an efficient way. Furthermore, they can also be used to develop efficient algorithms for quantitative reliability analysis. Because of that, they are also taught in the frame of a course on reliability analysis at Faculty of Management Science and Informatics of University of Zilina. However, manipulation with them can be quite complicated without appropriate software support. In a frame of a project course, which is taught in the master study program "informatics" at the faculty, such a software has been prepared in Java. A screenshot of the software can be viewed in Fig. 6.

As you can see, the probabilistic BDD depicted in Fig. 6 is not a real BDD because it contains more than two leaves. However, all other properties of a BDD are satisfied. This form of a BDD can be denoted as an expanded BDD [4]. We choose this special form of BDD because it usually contains fewer crossing lines, what improves its readability.

The software that we created in the frame of the project course allows:

- defining own function,
- automatic creation of a binary decision tree,
- automatic creation of a BDD,
- moving with individual objects on the screen,
- changing specific properties of each object,
- attaching state probabilities to the components linked with internal nodes of a BDD,
- calculating the probability that the system defined by a structure function expressed in a BDD will be working or fail,
- generating the truth table for a BDD,
- loading a Boolean function in PLA file format [9],
- exporting the resulting tree/diagram to PNG or SVG file format.

Thanks to the possibility of exporting the diagram into PNG or SVG file format, it can be used not only for learning of BDDs but also for preparing pictures for teaching materials or for print publications.

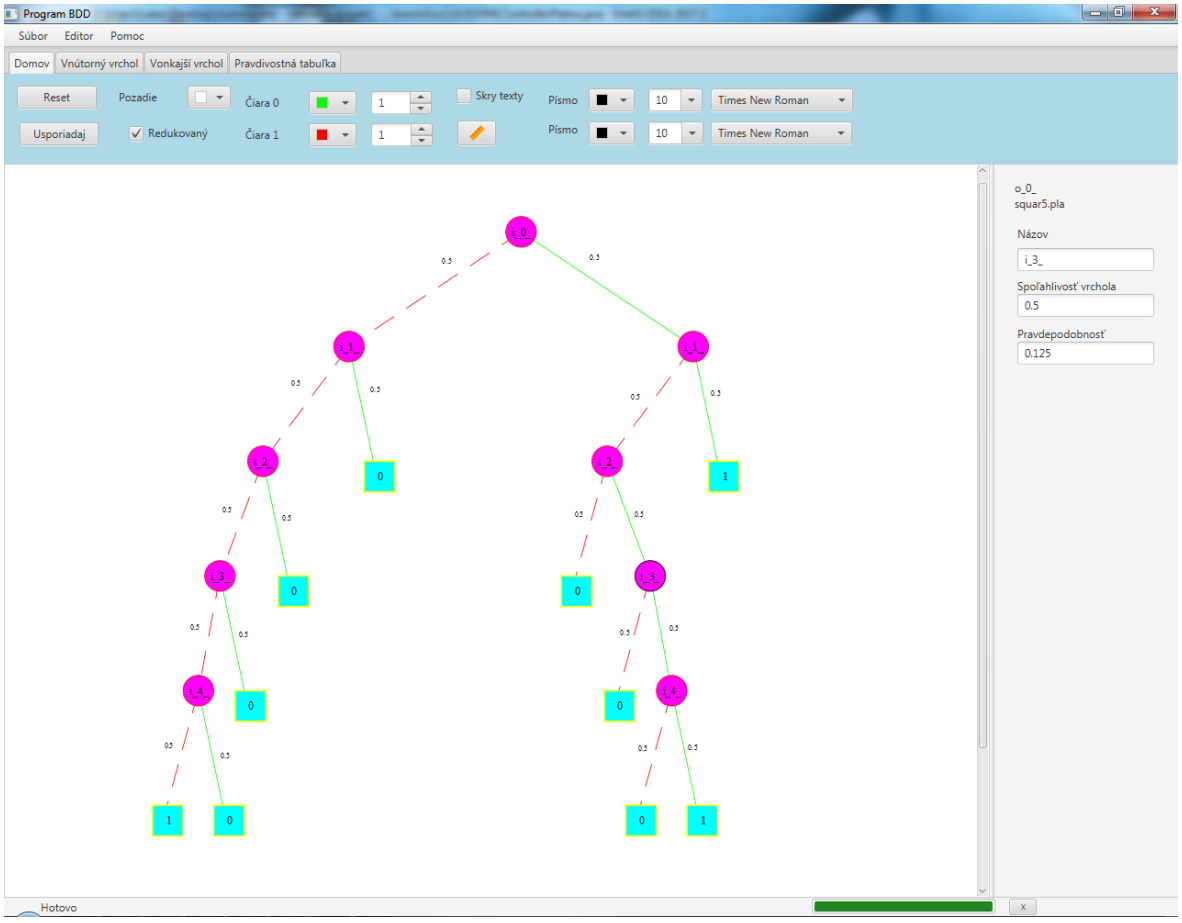

Fig. 6 Software for manipulation with binary decision diagrams

V. Conclusion

BDDs are one of efficient structures for representation of Boolean functions. They can be processed on a computer easier than algebraic formulae, and they have much less demands on memory than the truth table. They can be used in several tasks or research fields, such as logic design or reliability analysis. In reliability analysis, they allow representing the structure function of systems composed of a lot of components. Thanks to this, they can be used to develop new methods for quantitative analysis of complex systems. However, manipulation

with BDDs requires special software because they are not suitable for hand calculations. Because of that, we created a graphic application that allows creating BDDs, manipulating with them and use them in quantification of system availability. The application will be used as a support tool for teaching a course on reliability analysis at Faculty of Management Science and Informatics of University of Zilina.

REFERENCES

[1] E. Zio, "Reliability engineering: Old problems and new challenges," *Reliability Engineering & System Safety*, vol. 94, no. 2, pp. 125–141, Feb. 2009.

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

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

[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] T. Sasao and M. Fujita, Eds., *Representations of Discrete Functions*. Norwell, MA: Kluwer Academic Publishers, 1996.

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

[7] P. Rusnak, "Transformation of Boolean expression into disjunctive or conjunctive normal form," *Central European Researchers Journal*, vol. 3, no. 1, pp. 43–49, 2017.

[8] J. Kostolny, E. Zaitseva, P. Rusnak, and M. Kvassay, "Application of multiple-valued logic in importance analysis of k-out-of-n multi-state sSystems," in *2018 IEEE 48th International Symposium on Multiple-Valued Logic (ISMVL)*, 2018, pp. 19–24.

[9] "ESPRESSO(5OCTTOOLS)." [Online]. Available: http://www.ecs.umass.edu/ece/labs/vlsicad/ece667/links/espresso.5.html. [Accessed: 12-Jun-2018].