UNIVERSITY OF ŽILINA
FACULTY OF MANAGEMENT SCIENCE AND INFORMATICS

RELIABILITY ANALYSIS OF NON-COHERENT SYSTEMS BASEDON STRUCTURE FUNCTION METHODS

Dissertation thesis

Registration number: 28360020213007

Study program:	Applied Informatics
Field of study:	Informatics
Workplace:	Department of Informatics
	Faculty of Management Science and Informatics, University of Žilina
Supervisor:	doc. Ing. Miroslav Kvaššay, PhD.

Žilina, 2021 **Ing. Peter Sedláček**

UNIVERSITY OF ŽILINA
FACULTY OF MANAGEMENT SCIENCE AND INFORMATICS

RELIABILITY ANALYSIS OF NON-COHERENT SYSTEMS BASEDON STRUCTURE FUNCTION METHODS

Dissertation thesis

Registration number: 28360020213007

Study program:	Applied Informatics
Field of study:	Informatics
Workplace:	Department of Informatics
	Faculty of Management Science and Informatics, University of Žilina
Supervisor:	doc. Ing. Miroslav Kvaššay, PhD.

Žilina, 2021 **Ing. Peter Sedláček**

Acknowledgement

I would like to thank my supervisor specialist, Professor Elena Zaitseva, PhD., my supervisor Associate Professor Miroslav Kvassay, PhD. and my other colleagues for their advises, suggestions and cooperation.

Abstract

Spoľahlivosť je jednou z charakteristík systémov, ktoré majú veľkú dôležitosť v súčasnosti, keďže zlyhania systémov môžu mať fatálne následky. V našej práci sa zameriavame na analýzu nekoherentných viacstavových systémov reprezentovaných vo forme štruktúrnej funkcie. Táto funkcia nám umožňuje popísať závislosť funkčnosti systému od funkčnosti jednotlivých komponentov. Keďže reálne systémy často pozostávajú z veľkého množstva komponentov, je potrebné reprezentovať štruktúrnu funkciu efektívnym spôsobom. Za týmto účelom používame viac-hodnotový rozhodovací diagram. Samotná analýza systému popísaného pomocou štruktúrnej funkcie pozostáva z hľadania kritických stavov a výpočtu charakteristík systému a jeho komponentov. Jedným z tradičných nástrojov na túto analýzu je orientovaná parciálna logická derivácia. V rámci našej práce sme navrhli jej využitie pre analýzu nekoherentných viac-stavových systémov. Ďalším relevantným problémom v spoľahlivostnom inžinierstve je problém neúplne definovanej štruktúrnej funkcie, keďže pri mnohých reálnych systémoch nemáme k dispozícii úplnú informáciu o analyzovanom systéme. Za týmto účelom je možné využiť nástroje dolovania dát na vytvorenie rozhodovacieho stromu. Tento je následne možné redukovať na štruktúrnu funkciu vo forme rozhodovacieho diagramu. V rámci práce popísujeme algoritmus redukcie rozhodovacieho stromu na rozhodovací diagram. V ďalšej časti práce sa venujeme analýze spoľahlivosti softvéru. Existuje viacero prístupov k analýze softvéru. V našej práci sme sa rozhodli analyzovať zdrojový kód, ten reprezentovať vo forme abstraktného syntaktického stromu a z neho vytvoriť spoľahlivostný model. Použitie všetkých navrhnutých prístupov a metód je demonštrované na konkrétnych príkladoch, najmä systémov s ľudským faktorom, čo je typický príklad nekoherentného systému.

Kľúčové slová: analýza spoľahlivosti, viac-hodnotový rozhodovací diagram, nekoherentný systém, ukazovatele dôležitosti, neúplne definovaná štruktúrna funkcia, spoľahlivosť softvéru

Abstract

Reliability is one of system characteristics that have great importance in these days as consequences of system failure can be fatal in some cases. In our thesis, we are focusing on analysis of non-coherent multi-state systems represented in a form of structure function. This function allows us to express dependency of system performance on performance of its components. As real systems usually consists of large amount of components, it is necessary to represent structure function in an efficient way. For this purpose, we are using multi-valued decision diagrams. The analysis of a system described in a form of structure function consists of search for critical states and calculation of system characteristics and characteristics of system components. Direct partial logical derivative is one of typical tools used to perform this analysis. In our work, we propose its usage for analysis of non-coherent systems. The other relevant problem in reliability engineering is the problem of incompletely specified structure function, as in many real systems we do not have full information about analysed system. One of the approaches to solve this is the usage of methods of data mining to construct decision tree. It is possible to reduce this tree into structure function in a form of decision diagram. In our work, an algorithm for this reduction is described. The next part of our thesis focuses on software reliability. We decided to analyse source code, represent it in a form of abstract syntax tree and use it to construct reliability model. The usage of the proposed approaches and methods is demonstrated in specific cases, mainly for systems with human factor, as they are typical examples of non-coherent systems.

Keywords: reliability analysis, multi-valued decision diagram, non-coherent system, importance measures, incompletely specified structure function, software reliability

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

A Availability

AST Abstract Syntax Tree

BDD Binary Decision Diagram

BI Birnbaum's Importance

BSS Binary-State system

DAM Domain Analysis Model

DD Decision Diagram

DPLD Direct Partial Logical Derivatives

DT Decision Tree

FT Fault Tree

IDPLD Integrated Direct Partial Logical Derivatives

LDC Logic Differential Calculus

MCS Minimal Cut Set

MCV Minimal Cut Vector

MDD Multi-Valued Decision Diagram

MSS Multi-State system

MTTF Mean time to failure

MVL Multi-Valued Logic

NHPP Non-Homogeneous Poisson Process

R Reliability

RBD Reliability Block Diagram

SF Structure Function

SI Structure Importance

SRGM Software Reliability Growth Model

TD Truth Density

TT Truth Table

U Unavailability

UML Unified Modeling Language

Notation

n number of system components

x_i i-th system component state

$\boldsymbol{x}$ vector of system components states

m number of system states

m_i number of i-th system component states

$p_{i,s}$ probability system component x_i is in state s

q_i probability system component x_i is in state 0

$\boldsymbol{p}$ vector of system components state probabilities

$\phi(\boldsymbol{x})$ Structure Function

Introduction

System reliability is an important characteristic of any system in these days. The reliability analysis is a complex process. The first step of this process is the creation of a mathematical representation or mathematical model of the investigated system [4, 5, 6]. The mathematical model is constructed depending on the specifics of system analysis and properties of the investigated system. Depending on number of system performance levels, mathematical models can be divided into two types depending on the detail of the analysis [4]: Binary-State Systems (BSS) and Multi-State Systems (MSS).

Both of these types can be coherent or non-coherent depending on influence of system component degradation (failure) on the system functioning [7, 8, 9]. The degradation or failure of a component of coherent system cannot result into the system performance level improvement and all components of coherent system are relevant to the system functioning [10, 8]. Non-coherent system reliability does not increase monotonically if functioning/reliability of some components increase. Coherent systems are intensively studied in contrast to non-coherent systems in reliability engineering. Of course, coherent systems dominate in engineering applications, but non-coherent systems represent very often control applications [11, 12], multi-tasking applications [13], and applications with limited resources [14]. Most studies of non-coherent systems are developed for BSS. One of the first formalisms in non-coherent system reliability analysis has been definition of k-to-l-out-of-n system [15]. This type of system has been used for the investigations of a multiprocessor system [14], influence of weather conditions to power systems [16], or performability analysis of large-scale distributed systems [17]. There are other examples of non-coherent BSS in reliability analysis, such as H.264 video coding standard [7], nuclear systems [18], gas supply system with safety features [19], logical circuits [20, 21], traffic light systems [22], power control systems [23], or liquid level control systems [24].

The reliability analysis of non-coherent system requires special methods. It is caused by the non-monotonic impact of component failure on the functioning of such systems. Methods for the evaluation of non-coherent BSS availability/reliability, frequency indices and MTTF have been developed and presented in [22, 15, 14, 17, 25]. The

studies of importance analysis of non-coherent BSS have been discussed in papers [7, 12, 19, 26]. The most often approach for non-coherent BSS is based on analysis of prime implicants. The prime implicants have been used for the development of methods for non-coherent system evaluation in [27, 12, 28, 29]. Prime implicants have been proposed in [29] as analogy of minimal cut sets to define minimum combinations of failures that cause failure of a non-coherent system. Fault tree and minimal cut set are efficient tool for analysis of coherent system, but they are not equal with prime implicants and do not take into account non-monotonic influence of components failure of non-coherent system. Fault tree and minimal cut sets have been generalized for non-coherent BSS analysis in [25, 30, 20, 31]. The definition of prime implicants is complex and time-consuming problem if the number of system components increases. One of the proposed decisions is application of Binary Decision Diagrams (BDD) for the system representation [26, 17, 32, 31].

Non-coherent MSS are not studied intensively unlike BSS. There were studies of non-coherent MSS in which some theoretical aspects were considered [33, 34, 35, 36], but they have not been further developed. This is caused by some difficulties/ambiguities in the theoretical interpretation of non-coherent MSS and computational complexity of MSS analysis. The conceptions of coherent and non-coherent BSS were generalized for MSS in [34, 35, 36]. According to these studies, MSS is coherent if all its components are relevant to the system and degradation of any component cannot result in the system performance level improvement. The condition of components relevance was not clearly defined in these studies and authors of papers [33, 34, 35, 36, 37] proposed some interpretation of this condition. Every of the proposed definition of component relevance for MSS has own distinctive specifics. The more detailed analysis of these definitions is presented in sections 2.1 and 2.2 of this thesis. The calculation of indices and measures of non-coherent MSS has been discussed in [38, 39, 40]. Bossche in [38, 39] considered the frequency evaluation of reliability of non-coherent MSS based on fault tree with application of prime implicants. The author has generalized the conception of prime implicants from non-coherent BSS and used modified approach of Boolean logic for analysis of MSS. The methods of Multiple-Valued Logic (MVL) are used in definition of importance measures of non-coherent MSS in [40], but the authors have proposed intuitive calculation of importance measure and has not analysed the

theoretical background and definitions of non-coherent MSS.

The mentioned led us to set the following goals, we will investigate within our research in this thesis:

1. The investigation of methods for reliability analysis of multi-state non-coherent systems.

2. The investigation of methods for the reduction of computational complexity of structure function analysis.

3. The investigation of methods for construction of structure function based on incompletely specified data.

Except these goals we decided to expand our research on software reliability and investigate the possibility to represent software in a form of structure function.

In this thesis, we propose new approach for non-coherent MSS analysis based on mathematical methods of MVL. This approach is development of studies of coherent MSS, which have been presented in [6, 41]. Similar to the studies in [6, 41], the investigated system is represented by structure function, which maps all possible system components states to system performance levels. The structure function according to [6] can be interpreted as MVL function, and this fact allows us to use MVL mathematical methods for analysis and evaluation of MSS. In particular, the analysis of critical system component states is considered in this thesis. The critical component states are defined for every system component. For these states the specified change of component state results in the change of the system performance level. According to previous studies in [6], the critical system state can be computed by methods of Logical Differential Calculus, in particular, Direct Partial Logical Derivatives (DPLD). In study [41], the derivatives named Integrated Direct Partial Logical Derivatives (IDPLD) have been developed for calculation of Importance Measures of coherent MSS. Other modification of DPLD in [42] allows definition of minimal cut/path sets of coherent MSS. In this thesis, we develop DPLD-based approach for the computation of critical component states of non-coherent system. The development of this approach is based on analysis of definition of non-coherent MSS proposed in investigation [33, 34, 35, 36, 37].

The organization of this thesis is following. Chapter 1 summarizes the existing research in reliability analysis with focus on structure function, its construction and

basic representation. It also contains description of logic differential calculus as a method for analysis of structure function. The last part of this chapter presents basic terms from quantitative analysis, such as reliability, structure importance etc.

The main focus of chapter 2 is on our research in reliability analysis of non-coherent systems. In the first part, we are describing structure function of non-coherent systems, its classification and an example of this system. This example is in the next section used to demonstrate usage of direct partial logical derivatives to its analysis. Within this section, new type of logic derivative is proposed. This derivative is suitable for analysis of non-coherent systems, specifically for analysis of critical states for individual components. The next section describes one of the effective representations of structure function - multi-valued decision diagram, together with algorithms of its creation for incompletely specified structure function.

Chapter 3 is focused on software reliability, description of traditional models used to analyse this kind of systems and our proposed method. This method uses source code to create reliability model - fault tree - that can be easily transformed into structure function. This allows us to use the same approach to calculate software reliability as to analyse other systems, i.e. DPLD etc.

Finally, the chapter 4 contains demonstration of all proposed methods. There are 4 case studies, namely anesthesia examination, patient with hepatitis survival chance, bike crashes and example of software reliability.

1. Basic definition and properties in Reliability engineering

Evaluation of system reliability is a complex process whose result is information about system and its characteristics from reliability point of view, such as reliability, importance measures, critical states etc. The whole analysis is adjusted depending on characteristics we want to obtain. This adjustment consists of the selection of a mathematical representation (mathematical model) of analysed system. There are different mathematical models in reliability analysis, and two criteria are taken into account in the creation of the mathematical model, namely:

- number of system states;
- background mathematical approach, which determines algorithms and methods used for system evaluation.

The number of system states (performance levels) and number of its components states are resulted by requirements for analysis detail. According to the number of system performance levels, there are two types of mathematical representations that are known as Binary-State System (BSS) and Multi-State System (MSS).

BSS is a mathematical representation of system with two performance levels - i.e. system is either working or not. This mathematical representation is used if system is binary-state from its nature [21, 43], or we are analysing consequences of a system failure [5].

MSS allows defining in system mathematical representation more than two performance levels and describing gradual degradation of system performance from fully working to fully broken [44, 4, 45]. MSS allows performing the system analysis in more detail but computational complexity of this analysis increases and special methods should be developed for quantitative analysis of system associated with this type of mathematical representation. Such methods correlates with background methods used in quantitative analysis of system reliability, for example, stochastic methods, methods of Boolean logic or algebra logic [46, 44].

Depending on the mathematical background used to analyse system, there are several models, such as Structure Function (SF), Markov model, Universal generating function, Fault Tree (FT) etc. The mathematical model that correlates with algebra logic is structure function. The structure function expresses dependency of the change of a system performance level on the change of a performance level of its components. SF can be used for both BSS and MSS and allows us to represent system of any topological complexity.

1.1 Structure Function

Let us consider a MSS with n components. The system has m performance levels. The i-th component of this system has m_i states. The system performance levels and the i-th system component states are changing from 0 for representation of failure to $m-1$ and m_i-1 respectively for indication of perfect functioning. The dependency of system performance on states of its components can be expressed using Structure Function (SF) in form [47, 6]:

$$\begin{aligned}\phi(x_1, x_2, ..., x_n) = \phi(\boldsymbol{x}) :&\{0, 1, ..., m_1 - 1\} \times \{0, 1, ..., m_2 - 1\} \times ... \\ &\times \{0, 1, ..., m_n - 1\} \to \{0, 1, ..., m - 1\},\end{aligned} \tag{1.1}$$

where x_i is the i-th system component state, $i \in \{1, 2, ..., n\}$, $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ is a state vector.

The structure function (1.1) represents heterogenous MSS. If $m_i = m$ for any i, $i \in \{1, 2, ..., n\}$, SF can be expressed in the following form:

$$\phi(x_1, x_2, ..., x_n) = \phi(\boldsymbol{x}) : \{0, 1, ..., m - 1\}^n \to \{0, 1, ..., m - 1\}. \tag{1.2}$$

In case $m = m_i = 2$, the mathematical model expressed by SF (1.1) is a structure function of BSS.

The probability of state j of the i-th system component can be defined as:

$$\begin{aligned}p_{i,j} &= \Pr\{x_i = j\} \\ q_i = p_{i,0} &= \Pr\{x_i = 0\}.\end{aligned} \tag{1.3}$$

1.1.1 Representations of structure function

Structure function can be expressed in different forms. Let us consider the following example that will be used to describe different forms of structure function. System consists of 3 components and can be described using 3 states. The first component x_1 is binary-state and the others, i.e. x_2 and x_3 are 3-state. In case that component x_1 is in state 0, resulting system state can be expressed as $\min(x_2, x_3)$. In case that component x_1 is in state 1, system state can be described as $\max(x_2, x_3)$. Structure function of this system can be represented as following:

- Truth Table (TT) - This representation can be seen in Tab. 1.1. Each row of this table consists of specific set of components states and corresponding system state. The main disadvantage of this representation is its large dimension even for relatively small systems. In general, dimension of truth table can be calculated as a multiplication of number of each component states. Dimension of Truth Table for this example is $2 \times 3 \times 3 = 18$ rows.

- Reliability Block Diagram (RBD) - In this representation, each system component is represented as a block connected to other in series or parallel way. In case of BSS, the series connected components corresponds to AND function and parallel corresponds to OR function. In case of BSS, Reliability Block Diagram can be used to fully describe analysed system. In case of MSS, there is necessary to use multiple Reliability Block Diagrams - one for each state. Alternative approach is to define different functions for series and parallel connections, for example MIN function for series connected blocks [48, 44]. RBD of our example can be seen in Fig. 1.1 and Fig. 1.2.

- Binary and Multi-valued Decision Diagram (BDD and MDD) - In this representation, SF is in form of rooted acyclic graph. Each non-sink node represents system component and sink node represents system state. A path from the root to a sink node expresses specific combination of system components states and corresponding system state for this combination of components. Structure Function in this form can be seen in Fig. 1.3.

- Minimal Cut Set (MCS) and Minimal Cut Vector (MCV) - Minimal Cut Set can

Table 1.1: Structure Function represented in form of Truth Table

x_1	x_2	x_3	$\phi(\boldsymbol{x})$
0	0	0	0
0	0	1	0
0	0	2	0
0	1	0	0
0	1	1	1
0	1	2	1
0	2	0	0
0	2	1	1
0	2	2	2
1	0	0	0
1	0	1	1
1	0	2	2
1	1	0	1
1	1	1	1
1	1	2	2
1	2	0	2
1	2	1	2
1	2	2	2

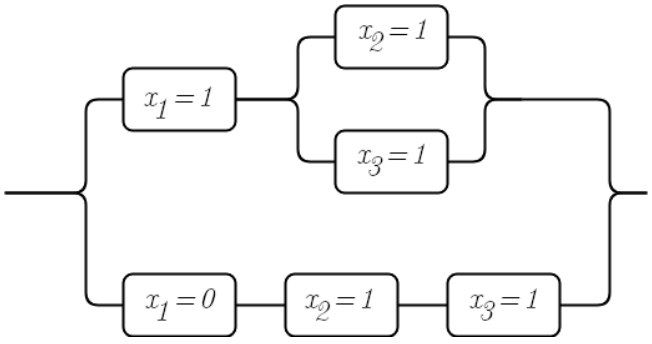

Figure 1.1: Structure Function for state 1 represented in form of Reliability Block Diagram

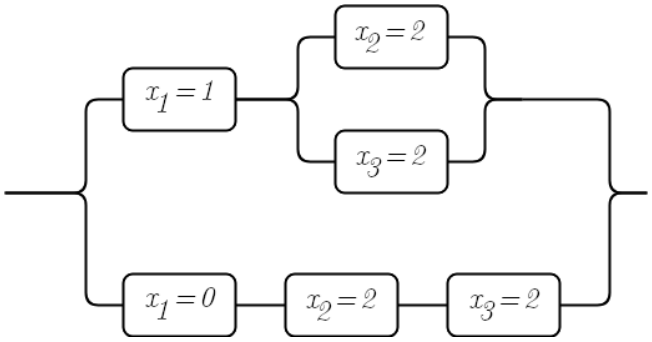

Figure 1.2: Structure Function for state 2 represented in form of Reliability Block Diagram

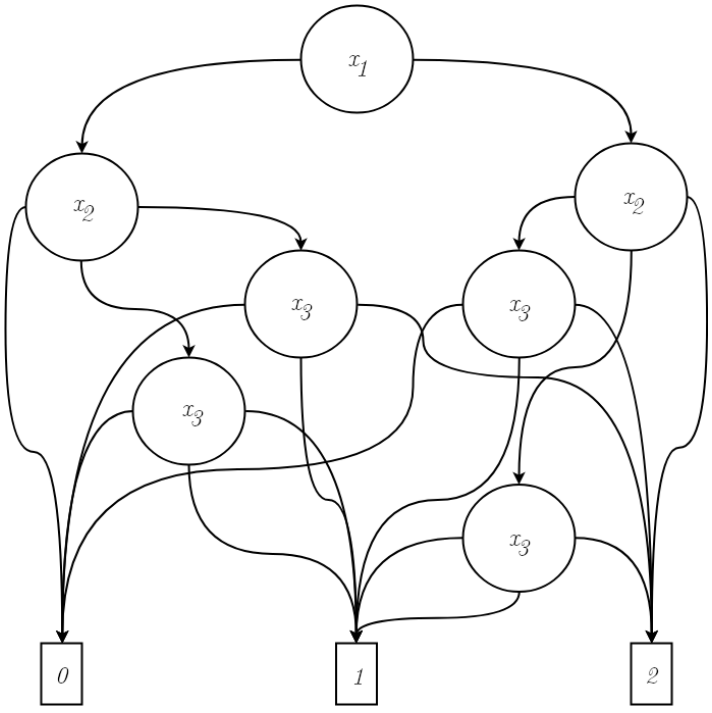

Figure 1.3: Structure Function represented in form of MDD

be described as a minimal set of components, whose simultaneous failure leads to system failure. This is in case of BSS. In case of MSS, minimal cut vector is used as minimal state vector (vector of system components states) that describes a situation where a repair of any failed component leads to system functioning [49].

Many real systems consist of large number of components, which leads to large complexity of structure function. Therefore, it is necessary to use effective representation of Structure Function, for example MDD.

1.2 Logic differential calculus

The logic differential calculus has been elaborated for analysis of dynamic properties of logical functions. Logic differential calculus is formed by different approaches and methods for analysis of a logical function as Boolean function and MVL functions. The interpretation of a structure function 1.1 as an MVL function allows us to use a mathematical methods of MVL for analysis and evaluation of MSS. Authors of [50, 51] have proposed to use Direct Partial Logical Derivatives (DPLD) to evaluate importance of system components. The definitions of importance measures based on DPLD have been introduced in [41]. However, this research has been implemented for coherent MSS. In this thesis, the DPLD-based evaluation of non-coherent system is considered.

DPLD has been defined for Boolean functions and then generalized for MVL functions in [52]. In algebra logic, DPLD of MVL function with respect to variable x_i indicated the change of the function value from j to h depending on this variable change from s to r. In terms of reliability analysis, DPLD of the structure function allows defining the change of the system performance level from j to h depending on the i-th component state change from s to r [50]:

$$\frac{\partial\phi(j \to h)}{\partial x_i(s \to r)} = \begin{cases} 1, & \text{if} \quad \phi(s, \boldsymbol{x}) = j \quad \text{and} \quad \phi(r, \boldsymbol{x}) = h \\ 0, & \text{otherwise} \end{cases} \tag{1.4}$$

for $s, r \in \{0, 1, ..., m_i - 1\}, s \neq r, j,h \in \{0, 1, ..., m - 1\}, j \neq h$.

All possible changes of the system performance level can be indicated based on the DPLD (1.4) for the specified component state change from s to r if this derivative is computed for all possible changes of the system performance levels that are defined

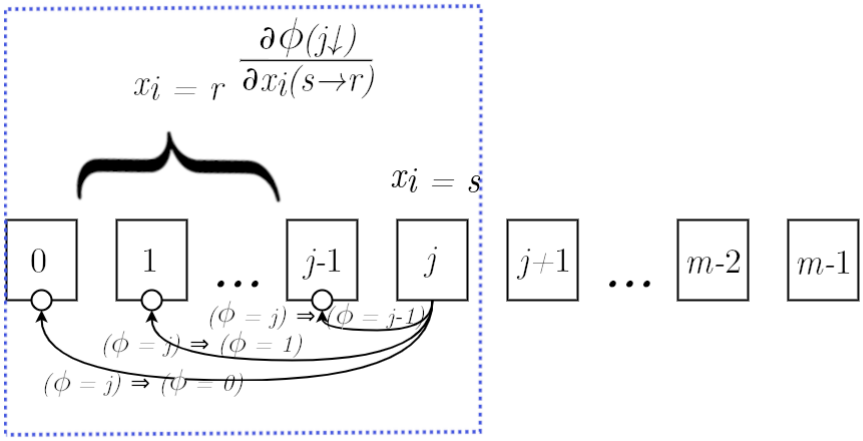

Figure 1.4: Ilustration of DBLD according to Equation 1.5

by parameters j and h. This implies calculation of large set of DPLD. New types of DPLD have been introduced in paper [41] to analyse set of system performance levels depending on the indicated change of the system component. These derivatives were named as integrated DPLD (IDPLD). There have been introduced three types of IDPLD. In paper [41], the integrated DPLD are defined for the i-th component change from s to $s-1$ for the system degradation analysis. The integrated DPLD for the component change from s to $s+1$ allows analysing improvement of the system performance levels [41]. The integrated DPLD for the i-th component change from s to r can be defined in the similar way.

The IDPLD of type I identifies state vectors at which degradation of the i-th component from state s to r results in degradation of system performance level from j to any state $h < j$:

$$\frac{\partial\phi(j\downarrow)}{\partial x_i(s\to r)} = \bigvee_{h=0}^{j-1} \frac{\partial\phi(j\to h)}{\partial x_i(s\to r)} = \begin{cases} 1, & \text{if} \quad \phi(s,\boldsymbol{x}) = j \quad \text{and} \quad \phi(r,\boldsymbol{x}) < j \\ 0, & \text{otherwise} \end{cases} \tag{1.5}$$

for $j \in \{1, 2, ..., m-1\}$.

Another version of the IDPLD of the type I for coherent system allows finding state vectors for which degradation of the i-th component from state s to r leads to the system performance level degradation from any state $h > j$ to j:

$$\frac{\partial\phi(\downarrow j)}{\partial x_i(s\to r)} = \bigvee_{h=j+1}^{m-1} \frac{\partial\phi(h\to j)}{\partial x_i(s\to r)} = \begin{cases} 1, & \text{if} \quad \phi(s,\boldsymbol{x}) > j \quad \text{and} \quad \phi(r,\boldsymbol{x}) = j \\ 0, & \text{otherwise} \end{cases} \tag{1.6}$$

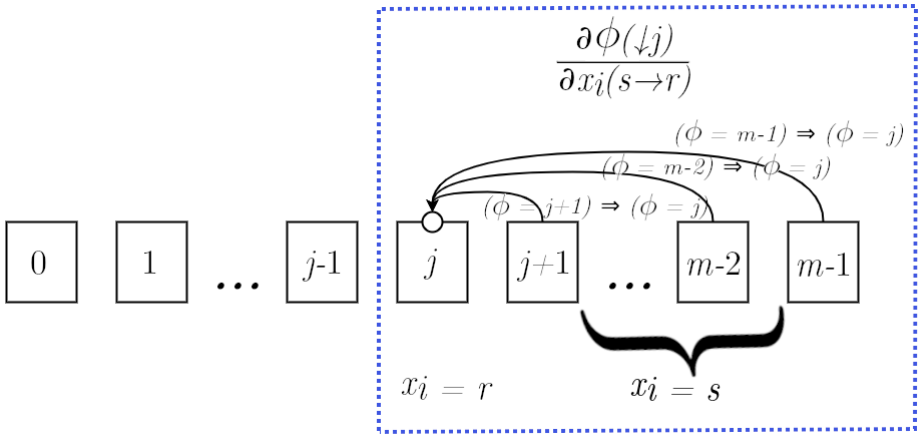

Figure 1.5: Illustration of DPLD according to Equation 1.6

for $j \in \{1, 2, ..., m-1\}$.

The difference between these two versions of IDPLD of the type I defined by (1.5) and (1.6) is illustrated in Fig. 1.4 and 1.5. The IDPLDs of the type I allow investigation of the component degradation influence for the specified system performance level j which is possible from this performance level (1.5) or from other system performance level to j according to (1.6). The analysis of all possible influences of the i-th component state change from s to r is implemented by the IDPLD of the II types. This is ilustrated in Fig. 1.6. This derivatives is defined as the join of the IDPLD of the type I as:

$$\frac{\partial\phi(\downarrow)}{\partial x_i(s\to r)} = \bigvee_{j=1}^{m-1} \frac{\partial\phi(j\downarrow)}{\partial x_i(s\to r)} = \bigvee_{j=0}^{m-2} \frac{\partial\phi(\downarrow j)}{\partial x_i(s\to r)} = \begin{cases} 1, & \text{if} \quad \phi(s_i, \boldsymbol{x}) > \phi(r_i, \boldsymbol{x}) \\ 0, & \text{otherwise} \end{cases} \tag{1.7}$$

The IDPLD of type III for the system performance level j identifies all state vectors for which the i-th component state change from s to r causes changes of the system performance level greater than or equal to j to value less than j:

$$\frac{\partial\phi(h_{\geq j}\to h_{<j})}{\partial x_i(s\to r)} = \bigvee_{h_u=j}^{m-1} \bigvee_{h_d=0}^{j-1} \frac{\partial\phi(h_u\to h_d)}{\partial x_i(s\to r)} = \begin{cases} 1, & \text{if} \quad \phi(s_i, \boldsymbol{x}) \geq j \\ & \text{and} \quad \phi(r_i, \boldsymbol{x}) < j \\ 0, & \text{otherwise} \end{cases} \tag{1.8}$$

where $j \in \{1, 2, ..., m-1\}$ and notation $h_{\geq j}(h_{<j})$ means that all system states that are greater than or equal to (less than) j are taken into account. Illustration of IDPLD of type III can be seen in Fig. 1.7.

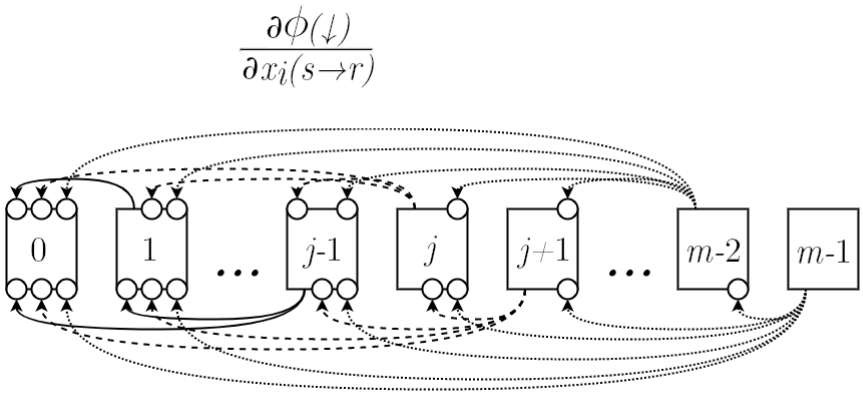

Figure 1.6: Ilustration of DBLD according to Equation 1.7

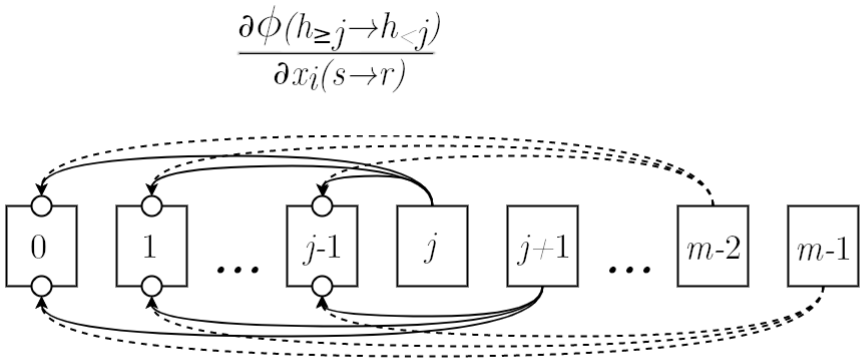

Figure 1.7: Ilustration of DPLD according to Equation 1.8

The IDPLDs (1.5) - (1.8) can be used to define critical system states. Let us consider a simple service system introduced in [41]. This system can be in one of three states, depending on a number of customers that can be satisfied: state 0 (no satisfied customer), state 1 (some customers can be served), state 2 (all potential customers will be satisfied). The system is composed of 3 components – 2 service points (components 1 and 2) and an infrastructure (component 3). The service point 1 is a major service center, and it has 3 possible states. State 0 agrees with the failure, state 1 with situations in which it is not perfectly working (i.e. it is not able to serve all its customers), and state 2 with situations when it is perfectly functioning. The service point 2 represents a minor center that can be either functioning (state 1) or failed (state 0). The quality of infrastructure is modeled using 3 states – from 0 (the infrastructure is poor quality) to 2 (the infrastructure is perfect). All potential customers can be served if only both of the service points are perfectly working. The structure function of the system is in Table 1.2.

The IDPLDs (1.5) - (1.8) are defined for the analysis of coherent MSS degradation. The IDPLD can be defined for the analysis of coherent MSS improving also in similar way [41]:

- The derivative $\partial\phi(j\uparrow)/\partial x_i(s \to r)$ allows defining the system state which are critical for improvement of a performance level j if the i-th system component modified from state s to r $(s < r)$;
- The derivative $\partial\phi(\uparrow j)/\partial x_i(s \to r)$ permits to indicate the critical state for which the improving of the i-th system component from state s to r $(s < r)$ results in the reaching of system performance level j;
- The derivative $\partial\phi(\uparrow)/\partial x_i(s \to r)$ is used to define critical state for the system improving depending on i-th component change from state s to r $(s < r)$;
- The derivative $\partial\phi(h_{<j} \to h_{\geq j})/\partial x_i(s \to r)$ allows computing the system state which are critical for reaching performance level j of system availability if the i-th component state change from s to r.

For the generalization of these derivatives for non-coherent system analysis we have to take into account the possibility of the specified system performance level j

Table 1.2: The structure function of simple service system

Component state		x_3		
x_1	x_2	0	1	2
0	0	**0**	**0**	**0**
0	1	**0**	**0**	**1**
1	0	**0**	**1**	**1**
1	1	**0**	**1**	**2**
2	0	**0**	**1**	**2**
2	1	**0**	**2**	**2**

Table 1.3: Nonzero elements of IDPLDs $\partial\phi(j\downarrow)/\partial x_i(s \to r)$

System state	Components and theis states						
	The 1-st component			The 2-nd component	The 3-rd component		
	$1 \to 0$	$2 \to 0$	$2 \to 1$	$1 \to 0$	$1 \to 0$	$2 \to 0$	$2 \to 1$
1	$(1 \to 0)01$ $(1 \to 0)02$ $(1 \to 0)11$	$(2 \to 0)01$		$0(1 \to 0)2$	$10(1 \to 0)$ $11(1 \to 0)$ $20(1 \to 0)$	$01(2 \to 0)$ $10(2 \to 0)$	$01(2 \to 1)$
2	$(1 \to 0)12$	$(2 \to 0)02$ $(2 \to 0)11$ $(2 \to 0)12$	$(2 \to 1)02$ $(2 \to 1)11$	$1(1 \to 0)2$ $2(1 \to 0)1$	$21(1 \to 0)$	$11(2 \to 0)$ $20(2 \to 0)$ $21(2 \to 0)$	$11(2 \to 1)$ $20(2 \to 1)$

Table 1.4: Nonzero elements of IDPLDs $\partial\phi(\downarrow j)/\partial x_i(s \to r)$

System state	Components and theis states						
	The 1-st component			The 2-nd component	The 3-rd component		
	$1 \to 0$	$2 \to 0$	$2 \to 1$	$1 \to 0$	$1 \to 0$	$2 \to 0$	$2 \to 1$
0	$(1 \to 0)01$ $(1 \to 0)02$ $(1 \to 0)11$	$(2 \to 0)01$ $(2 \to 0)02$ $(2 \to 0)11$		$0(1 \to 0)2$	$10(1 \to 0)$ $11(1 \to 0)$ $20(1 \to 0)$ $21(1 \to 0)$	$01(2 \to 0)$ $10(2 \to 0)$ $11(2 \to 0)$ $20(2 \to 0)$ $21(2 \to 0)$	$01(2 \to 1)$
1	$(1 \to 0)12$	$(2 \to 0)12$	$(2 \to 1)02$ $(2 \to 1)11$	$1(1 \to 0)2$ $2(1 \to 0)1$			$11(2 \to 1)$ $20(2 \to 1)$

Table 1.5: Nonzero elements of IDPLDs $\partial\phi(h_{\geq j} \to h_{<j})/\partial x_i(s \to r)$

System state	Components and theis states						
	The 1-st component			The 2-nd component	The 3-rd component		
	$1 \to 0$	$2 \to 0$	$2 \to 1$	$1 \to 0$	$1 \to 0$	$2 \to 0$	$2 \to 1$
1	$(1 \to 0)01$ $(1 \to 0)02$ $(1 \to 0)11$	$(2 \to 0)01$ $(2 \to 0)02$ $(2 \to 0)11$		$0(1 \to 0)2$	$10(1 \to 0)$ $11(1 \to 0)$ $20(1 \to 0)$ $21(1 \to 0)$	$01(2 \to 0)$ $10(2 \to 0)$ $11(2 \to 0)$ $20(2 \to 0)$ $21(2 \to 0)$	$01(2 \to 1)$
2	$(1 \to 0)12$	$(2 \to 0)02$ $(2 \to 0)11$ $(2 \to 0)12$	$(2 \to 1)02$ $(2 \to 1)11$	$1(1 \to 0)2$ $2(1 \to 0)1$	$21(1 \to 0)$	$11(2 \to 0)$ $20(2 \to 0)$ $21(2 \to 0)$	$11(2 \to 1)$ $20(2 \to 1)$

degradation and improving depending on the change of the i-th system component state from s to r [12, 53].

1.3 Coherent and Non-coherent systems

Depending on the system behavior, systems can be divided into two classes - **coherent** and **non-coherent** [11, 34, 7]. The dominant class of the systems is coherent for which any component failure can cause the system fault. The structure function (1.1) has to meet certain conditions to represent coherent systems. Non-coherent systems are investigated mainly for BSS. The following two conditions are required for a BSS to be a coherent [34, 7, 12]:

1. the SF is monotonically non-decreasing: $\phi(1_i,\boldsymbol{x}) \geq \phi(0_i,\boldsymbol{x})$ for any component i where $\phi(1_i,\boldsymbol{x}) = \phi(x_1, ..., x_{i-1}, 1, x_{i+1}, ..., x_n)$ and $\phi(0_i,\boldsymbol{x}) = \phi(x_1, ..., x_{i-1}, 0, x_{i+1}, ..., x_n)$;

2. Each system component is relevant to the system: $\phi(1_i,\boldsymbol{x}) \neq \phi(0_i,\boldsymbol{x})$ for some $\boldsymbol{x}$.

Most of the technical systems are coherent. A BSS is interpreted as a non-coherent if its structure function doesn't agrees with at least one of the two considered conditions [7, 12]. Examples of non-coherent systems are gas supply systems considered in [12, 7], logical circuits and networks [21], systems including human factor [54] or software components [30]. One of the relevant problem in non-coherent BSS investigation is

the importance analysis [7, 12, 19]. Authors of the paper [51] propose to use Logical Differential Calculus for a calculation of the importance indices of BSS. As was shown in [53, 21], this approach can be used for non-coherent BSS importance analysis too. At the same time mathematical approach of Logical Differential Calculus is generalised for MVL too and is efficient in evaluation of MSS. The methods for coherent MSS analysis based on Logical Differential Calculus are considered, for example, in [6, 41].

1.4 Quantitative analysis

Structure function can be used to analyse system from reliability point of view. This include calculus of system characteristics such as reliability and availability. Except system characteristics, it is possible to calculate characteristics of individual system components, such as importance measures.

1.4.1 System characteristics

Reliability is the basic system characteristic. It can be defined as a probability system is working.

$$R = \Pr\{\phi(\boldsymbol{x}) = 1\}. \tag{1.9}$$

In case of MSS, it is possible to express also probabilities for other performance levels. In this case, availability is used instead.

Availability and unavailability can be defined as a probability system is at least in j-th performance level. Similarly can be defined unavailability as probability system perform in less than j-th performance level.

$$\begin{aligned} A^{\geq j}(\boldsymbol{p}) &= Pr\{\phi(\boldsymbol{x}) \geq j\}, \\ U^{\geq j}(\boldsymbol{p}) &= Pr\{\phi(\boldsymbol{x}) < j\} \end{aligned} \tag{1.10}$$

where $\boldsymbol{p}$ is a vector of system components state probabilities (1.3). In case of BSS, availability is probability, system is working and unavailability probability system is in failure state [55, 45, 44].

In many cases it is important not only to know characteristics of system but also how is system performance level influenced by individual components. For that reason, Importance Measures is calculated.

Table 1.6: IMs calculation

	SI	BI
x_1	0.125	0.28
x_2	0.125	0.32
x_3	0.125	0.44

1.4.2 Structure Importance SI_i

Structure importance express influence of system component to system performance from topological point of view. Structure importance can be calculated as following:

$$\mathrm{SI}_i = \mathrm{TD}\left(\frac{\partial\phi(j \to k)}{\partial x_i(r \to s)}\right) \tag{1.11}$$

where x_i is the i-th component of system, j, k, r, s are states in which system or system component can be and TD is function defined as relative number of cases, function $\frac{\partial\phi(j\to k)}{\partial x_i(r\to s)}$ is true to all possible cases [56]. This function is DPLD and corresponds to equation 1.4. In case change of system component performance level from state r to state s causes change of system performance level from state j to state k this function takes value 1 otherwise this function takes value 0.

1.4.3 Birnbaum's Importance BI_i

BI_i is similar to SI_i with one essential difference namely BI_i takes into account also probabilities component of system is in state i. BI_i can be calculated as following [56, 57, 58]:

$$\mathrm{BI}_i = \Pr\left\{\frac{\partial\phi(j \to k)}{\partial x_i(r \to s)} = 1\right\}. \tag{1.12}$$

For example in Table 1.6 SI and BI for system with Structure Function defined in Table 2.5 is presented. This indices are calculated for components probabilities $p_{1,1} = 0.8$, $p_{2,1} = 0.7$ and $p_{3,1} = 0.6$.

2. Non-coherent systems

The previous chapter sums basic terms in reliability engineering. In this chapter, the main focus will be on non-coherent MSS systems, that are investigating more deeply. There was also the main focus on these systems in our work. In the first part, described in section 2.1 we will describe SF of non-coherent MSS. Section 2.2 describes the usage of DPLD as a tool to analyse these systems. As there can be large dimension of structure function in real systems, it is necessary to represent it in efficient way. For this purpose we decide to use MDD and represent it in form of vector of neighbours. This is present in Section 2.3 together with algorithm to create this MDD from DT. The last part of this chapter present in section 2.4 describes the problem of incompletely specified structure function.

2.1 Structure function of non-coherent MSS

Similarly to BSS for MSS are considered two types of systems in point of view of them behavior: coherent and non-coherent systems. The conception of coherent and non-coherent systems has been discussed in the first investigations in MSS reliability analysis [33, 34, 35, 36]. The coherence conception is generalized from the conception of coherent BSS [10]: MSS is coherent if all its components are relevant to the system and degradation of any component can not result the system performance level improving. The mathematical interpretation of this definition suppose that the MSS structure function is monotonically increasing (non-decreasing) and all its variables are relevant (the change of the variable value results in the change of the structure function value). Therefore there are two conditions for coherent MSS structure function: (1) the structure function increase and (2) the relevance of the variables of the structure function. In papers [33, 34, 35, 36] these conditions have been investigated for homogeneous MSS with the structure function (1.2). But in practice there is a small class of homogeneous system with the structure function (1.2). Most mathematical representations of systems appear as a heterogeneous structure function (1.1). The condition of the increasing for heterogeneous structure function (1.1) is similar with the condition of the homogeneous structure function increasing and is defined as:

$$\phi(s_i,\boldsymbol{x}) \geq \phi((s-1)_i,\boldsymbol{x}), \tag{2.1}$$

for any component i, where $\phi(s_i,\boldsymbol{x}) = \phi(x_1, ..., x_{i-1}, s, x_{i+1}, ..., x_n)$ and $\phi((s-1)_i,\boldsymbol{x}) = \phi(x_1, ..., x_{i-1}, s-1, x_{i+1}, ..., x_n)$ and $s \in \{1, ..., m_{i-1}\}$.

The condition of relevance can have some interpretations for MSS. Papers by Barlow & Wu [8], El-Neweihi, Proschan, Sethuraman [9] and Natvig [37] have investigated the basic conceptions for the theory of the coherent MSS and introduced some definitions of the components relevance for MSS. In particular, in [8] the definition of components relevance allows forming the class of coherent MSS for series and parallel systems only. This relevance of components are defined based on minimal path set. Griffith [35] propose two new conceptions of relevancy that are weaker than the one in [8] and extend the class of coherent MSS. Natvig [37] has considered two types of relevancy. One of them is similar to the conception introduced in [35], and other type according to [37] is defined by the use of special type of binary structure function. Author of paper [59] introduced weaker concepts of the MSS components relevance than in researches [35, 37]. Authors of papers [34, 33] reviewed and systematized proposed conceptions of components relevance for homogeneous MSS. Some of definitions of MSS components relevance according to [34, 33] are shown in Table 2.1. These definitions can be generalized for heterogeneous MSS (Table 2.1). Natvig [44] considered the known definitions of MSS components relevance. Some of these definitions are very similar and most of the functions of one class belongs to other class, and classes differ by several functions. In summary, based on analysis of all definitions of component relevance in Table 2.1, two conditions of component relevance will be used for heterogeneous MSS with the structure function (1.1) in this work. We suppose these conditions are most useful in system reliability analysis, because they allow to declare strong relevance and weak relevance accordingly:

$$\phi(s_i,\boldsymbol{x}) > \phi((s-1)_i,\boldsymbol{x}), \tag{2.2}$$

$$\phi((m_i-1)_i,\boldsymbol{x}) > \phi(0_i,\boldsymbol{x}), \tag{2.3}$$

for all $s \in \{1, ..., m_i - 1\}$ and some $\phi(.{}_i,\boldsymbol{x})$.

Depending on the relevance types (2.2) or (2.3) three types of MSS were introduced for homogeneous system:

Table 2.1: The conceptions of relevance for coherent MSS

	Definition		
Relevance	Homogenous	Non-homogenous	Coherence type
EPS [34]	for $\forall i = 1,...,n$ and $\forall j = 0,...,m-1$, $\exists\phi(._i,\boldsymbol{x})$ such that $\phi(j_i,\boldsymbol{x}) = j$ and $\phi(l_i,\boldsymbol{x}) \neq j$, $j \neq l$	for $\forall i = 1,...,n$, $\exists\phi(._i,\boldsymbol{x})$ such that $\phi(s_i,\boldsymbol{x}) = j$ and $\phi(r_i,\boldsymbol{x}) \neq j$ for $s \neq r$	strong coherent
G1 [35]	for $\forall i = 1,...,n$ and $\forall j = 1,...,m-1$, $\exists\phi(._i,\boldsymbol{x})$ such that $\phi((j-1)_i,\boldsymbol{x}) < \phi(j_i,\boldsymbol{x})$, $\phi(\boldsymbol{j}) = j$, $j = 1,...,m$	for $\forall i = 1,...,n$ and for $\forall s = 1,...,m_i - 1$, $\exists\phi(._i,\boldsymbol{x})$ such that $\phi((s-1)_i,\boldsymbol{x}) < \phi(s_i,\boldsymbol{x})$	coherent
G2 [35]	for $\forall i = 1,...,n$ and $\forall j = 10,...,m-1$, $\exists\phi(._i,\boldsymbol{x})$ such that $\phi(0_i,\boldsymbol{x}) < \phi((m-1)_i,\boldsymbol{x})$	for $\forall i = 1,...,n$, $\exists\phi(._i,\boldsymbol{x})$ such that $\phi(0_i,\boldsymbol{x}) < \phi((m_i-1)_i,\boldsymbol{x})$	weak coherent
N1 [37]	for $\forall i = 1,...,n$ and $\forall j = 0,...,m$, $\exists\phi(._i,\boldsymbol{x})$ such that $\phi(j_i,\boldsymbol{x}) \geq j$ and $\phi((j-1)_i,\boldsymbol{x}) \leq j-1$	for $\forall i = 1,...,n$, $\forall s = 1,...,m_i - 1$ $\exists\phi(._i,\boldsymbol{x})$ such that $\phi(s_i,\boldsymbol{x}) \geq j$ and $\phi((s-1)_i,\boldsymbol{x}) \leq j$	coherent
BS [60]	for $\forall i = 1,...,n$ and $\forall j = 0,...,m-1$, $\exists\phi(._i,\boldsymbol{x})$ such that $\phi(\boldsymbol{0}) = 0$ and $\phi(\boldsymbol{m-1}) = m-1$	is not defined	weak coherent

Table 2.2: Examples of coherent and non-coherent MSS ($n = 2, m_1 = 2, m_2 = 3, m = 4$)

x_1	x_2	$\phi_1(\boldsymbol{x})$	$\phi_2(\boldsymbol{x})$	$\phi_3(\boldsymbol{x})$	$\phi_4(\boldsymbol{x})$	$\phi_5(\boldsymbol{x})$
0	0	0	0	0	0	0
0	1	1	1	1	0	2
0	2	2	1	1	0	1
1	0	1	2	1	1	1
1	1	2	3	1	1	3
1	2	3	3	1	1	2

1. strongly coherent - the system structure function is monotonically increasing and all its components are relevant according to (2.2);

2. coherent - the system structure function is monotonically increasing and all its components are relevant according to (2.2) or (2.3);

3. weakly coherent - the system structure function is monotonically increasing and all its components are weakly relevant according to (2.3).

For example, consider structure functions of MSSs with four performance levels ($m = 4$) consisting of two components ($n = 2$) and first of these components has two possible states ($m_1 = 2$) and second has three states ($m_2 = 3$) (Table 2.2). The structure function $\phi_1(\boldsymbol{x})$ is the structure function of strongly coherent MSS, because the structure function of this MSS is monotonically increasing and two system components are relevant according to (2.2). The MSS structure functions $\phi_2(\boldsymbol{x})$, $\phi_3(\boldsymbol{x})$ and $\phi_4(\boldsymbol{x})$ are monotonically increasing. The MSS with the structure function $\phi_2(\boldsymbol{x})$ is coherent because the relevance of the first component agrees with the condition of strong relevance (2.2) and the relevance of the second component is consistent with the condition of weak relevance (2.3). Two components of the MSS structure function $\phi_3(\boldsymbol{x})$ comply with the condition of weak relevance (2.3), therefore this MSS is weak coherent. The MSS with the structure function $\phi_4(\boldsymbol{x})$ is non-coherent because of the second component, for which none of the relevance conditions (2.2) or (2.3) apply. The fifth MSS $\phi_5(\boldsymbol{x})$ is non-coherent too, because its structure function is not monotonically increasing.

Let us consider the example of the non-coherent MSS with SF as described in Table 2.3. This system consists of 3 components ($n = 3$), where x_1 is binary-state ($m_1 = 2$) and x_2 and x_3, as well as system itself are multi-states (m_2,m_3,$m = 3$).

Table 2.3: Example of structure function for non-coherent system with 3 components

x_1	x_2	x_3	$\phi(\boldsymbol{x})$
0	0	0	0
0	0	1	1
0	0	2	2
0	1	0	2
0	1	1	0
0	1	2	1
0	2	0	2
0	2	1	1
0	2	2	0
1	0	0	2
1	0	1	1
1	0	2	0
1	1	0	2
1	1	1	0
1	1	2	1
1	2	0	0
1	2	1	1
1	2	2	2

The considered system is non-coherent because its structure function is not monotonically increasing according to (2.1) $\phi(0,0,2) = 2 \geq \phi(0,1,2) = 1$ for $x_2 = 0 < x_2 = 1$. Therefore this specific should be took into account in analysis of this system. We propose to consider the non-coherent MSS analysis using MVL based methods. Such method has been developed for coherent MSS and some of them are presented, for example, in [6, 41]. The importance analysis of MSS in these investigation was developed based on the use of logical differential calculus which is one of mathematical approaches of MVL.

2.2 Direct Partial Logical derivatives of non-coherent MSS

The logical differential calculus, in particular, DPLD allows analysis of MSS based on logical expression. The reliability analysis of MSS based on logical expression with application of MVL allows making the computation and encoding straightforward for MSS evaluation. But most of known MVL based methods for MSS reliability evaluation have been developed for coherent systems [56, 61, 6]. Such methods are effectively used in MSS critical component analysis [6] and importance analysis [41]. In our work we propose to consider the application of DPLD in analysis and evaluation of non-coherent MSS to define the system critical states.

The important specific of non-coherent system is the influence of component state change into the change of the system performance level: degradation (failure) of the system component does not always lead to system functioning degradation or its failure. Therefore the analysis of component criticality should take into account different changes of states of fixed component and their correlations with changes of system states. In case of non-coherent BSS Andrews and Beeson [11, 12] proposed to consider the probability that component is critical to system failure as sum of two probabilities of component-failure criticality and component-repair criticality. The analysis of the component criticality of non-coherent MSS based on DPLD is introduced below.

The IDPLDs (1.5) - (1.8) in section 1.2 are introduced to analyse system degradation depending on the fixed change of the investigated component from state s to state r. The non-zero values of the derivatives (1.5) - (1.8) agree with the critical system states for i-th component state change from s to r. The development and generalization of these derivatives for non-coherent MSS allow defining new type of IDPLD for fixed change of the i-th component from the state s to state r. The analysis of non-

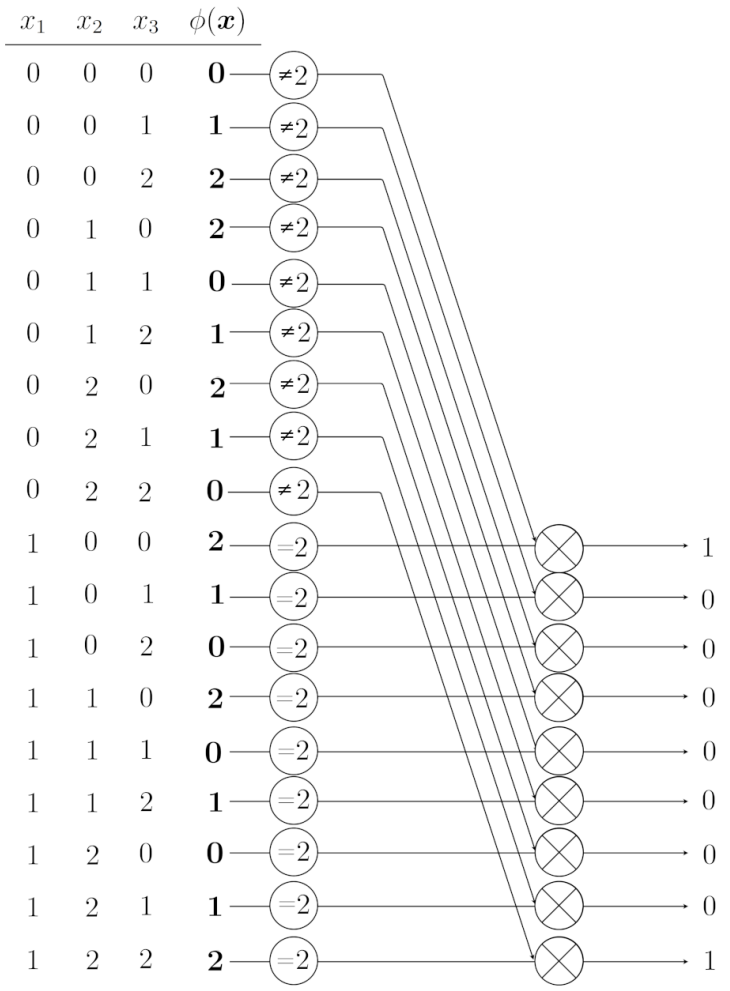

Figure 2.1: Calculation of IDPLD (2.4) for component x_1 and system performance level 2 of the structure function in Tab. 2.3

coherent MSS by IDPLD should take into account every system performance level j (for $j \in \{0, 1,..., m-1\}$) changes resulted by this component state change and is defined as:

$$\frac{\partial \phi(j \downarrow\uparrow)}{\partial x_i(s \to r)} = \begin{cases} 1, & \text{if} \quad \phi(s_i, \boldsymbol{x}) = j \quad \text{and} \quad \phi(r_i, \boldsymbol{x}) \neq j \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

The derivative (2.4) allows to indicate the critical states of the non-coherent MSS for the fixed system performance level j depending the i-th component state change from s to r.

For example, let us to consider the analysis of critical states by the derivative (2.4). The structure function of this system is in Table 2.3. The first component of this system has two states only. Therefore there are two possible changes of this component: from state 1 to zero and from zero to state 1. Three derivatives according to (2.4) can

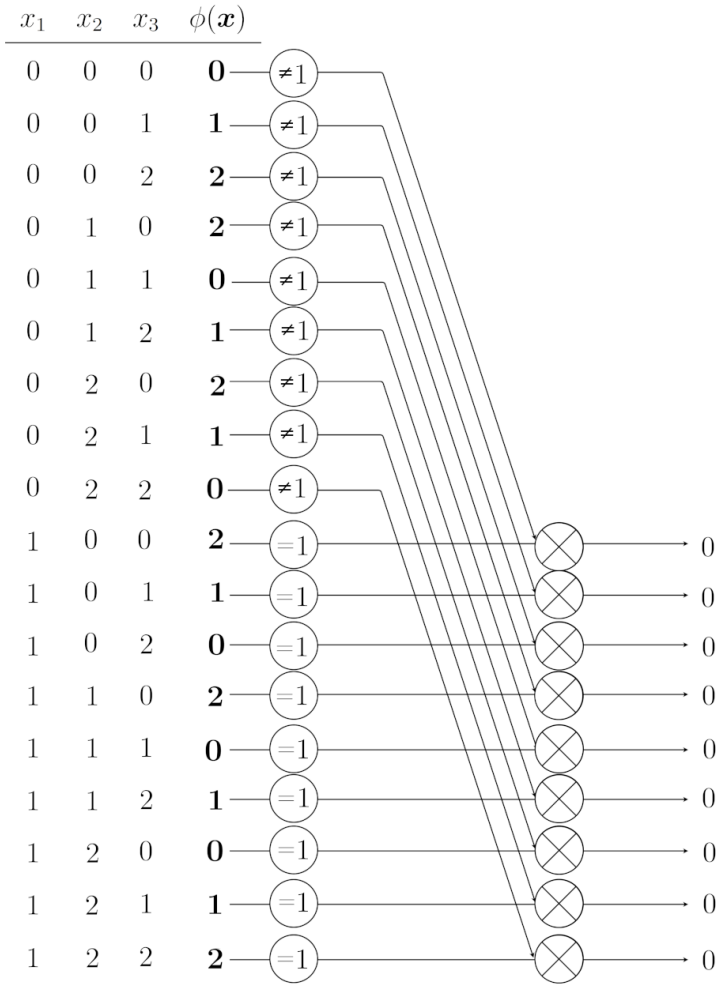

Figure 2.2: Calculation of IDPLD (2.4) for component x_1 and system performance level 1 of the structure function in Tab. 2.3

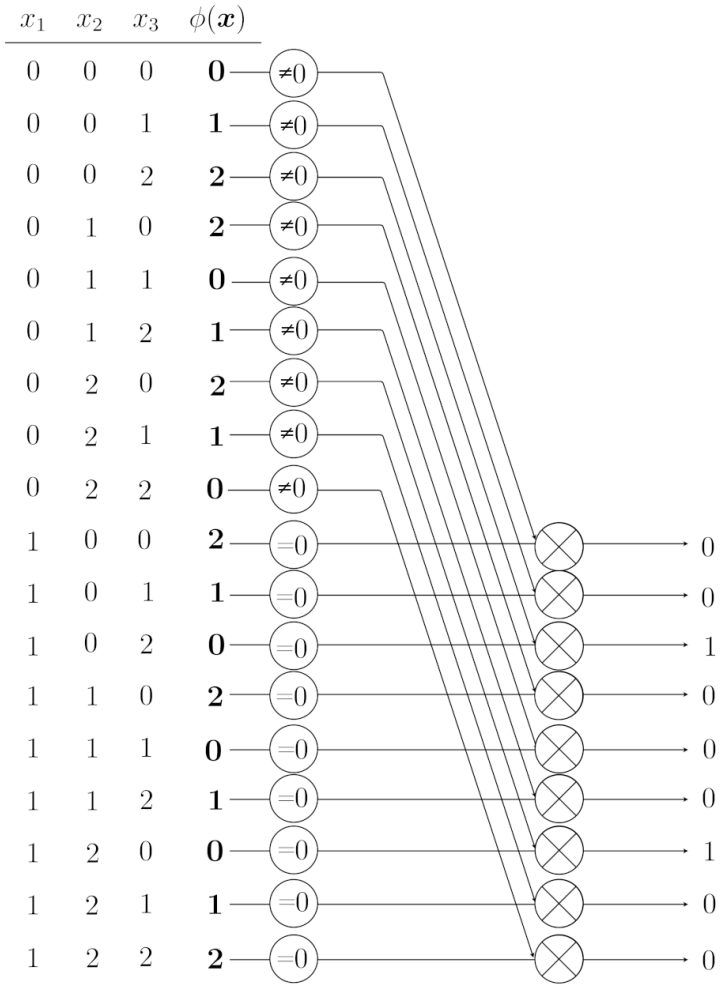

Figure 2.3: Calculation of IDPLD (2.4) for component x_1 and system performance level 0 of the structure function in Tab. 2.3

be computed for this component change from state 1 to zero: $\partial\phi(2\downarrow\uparrow)/\partial x_i(1\to 0)$, $\partial\phi(1\downarrow\uparrow)/\partial x_i(1\to 0)$ and $\partial\phi(0\downarrow\uparrow)/\partial x_i(1\to 0)$, which are consistent with the system states. The calculation of IDPLD $\partial\phi(2\downarrow\uparrow)/\partial x_i(1\to 0)$ by the flow diagram is shown in Fig.2.1. Non-zero values of the derivative indicates the system states which are critical for this system performance level 2 and the first component change from one to zero. This derivative has two non-zero values for states vectors $x_1x_2x_3 = ((1\to 0)\ \ 0\ \ 0)$ and $x_1x_2x_3 = ((1\to 0)\ \ 2\ \ 2)$. In the context of the considered system these critical states show that the change of this component performance causes the change of the whole system performance. In the similar way the influence of the first system component can be defined for other system performance levels (Fig.2.2 and Fig.2.3). The derivatives $\partial\phi(1\downarrow\uparrow)/\partial x_i(1\to 0)$ (Fig.2.2) and $\partial\phi(0\downarrow\uparrow)/\partial x_i(1\to 0)$ (Fig.2.3) show that the change of the performance of first component from 1 to 0 doesn't result in the change of the system performance level 1 and has influence for the level 0. Therefore for the first system component failure can be defined four critical state vectors: $x_1x_2x_3 = ((1\to 0)\ \ 0\ \ 0)$, $x_1x_2x_3 = ((1\to 0)\ \ 2\ \ 2)$, $x_1x_2x_3 = ((1\to 0)\ \ 0\ \ 2)$ and $x_1x_2x_3 = ((1\to 0)\ \ 2\ \ 0)$.

The DPLDs for non-coherent MSS (2.4) can be generalized for all possible system performance level as:

$$\frac{\partial\phi(\downarrow\uparrow)}{\partial x_i(s\to r)} = \bigvee_{j=0}^{m-1}\frac{\partial\phi(j\downarrow\uparrow)}{\partial x_i(s\to r)} = \begin{cases} 1, & \text{if} \quad \phi(s_i,\boldsymbol{x}) \neq \phi(r_i,\boldsymbol{x}) \\ 0, & \text{otherwise} \end{cases} \tag{2.5}$$

According to the derivative (2.5), all system critical states can be computed for the i-th system component which are caused by its state change from s to r. The calculations of the derivatives (2.5) for the first, second and third variables and their changes from state 1 to state 0 of the MSS structure function (introduced in Table 2.3) are illustrated by flow-diagrams in Fig. 2.4, Fig. 2.5 and Fig. 2.6 accordingly. These derivatives allow us to consider the influence of these components failure to the system functioning. The failure of the first component results the change for four states vectors considered above. The failure of the second component causes the system performance level change for five state vectors $x_1x_2x_3 = (0\ \ (1\to 0)\ \ 0)$, $x_1x_2x_3 = (0\ \ (1\to 0)\ \ 1)$, $x_1x_2x_3 = (0\ \ (1\to 0)\ \ 2)$, $x_1x_2x_3 = (1\ \ (1\to 0)\ \ 1)$ and $x_1x_2x_3 = (1\ \ (1\to 0)\ \ 2)$. And the failure of the third component always leads to a change of the system performance level.

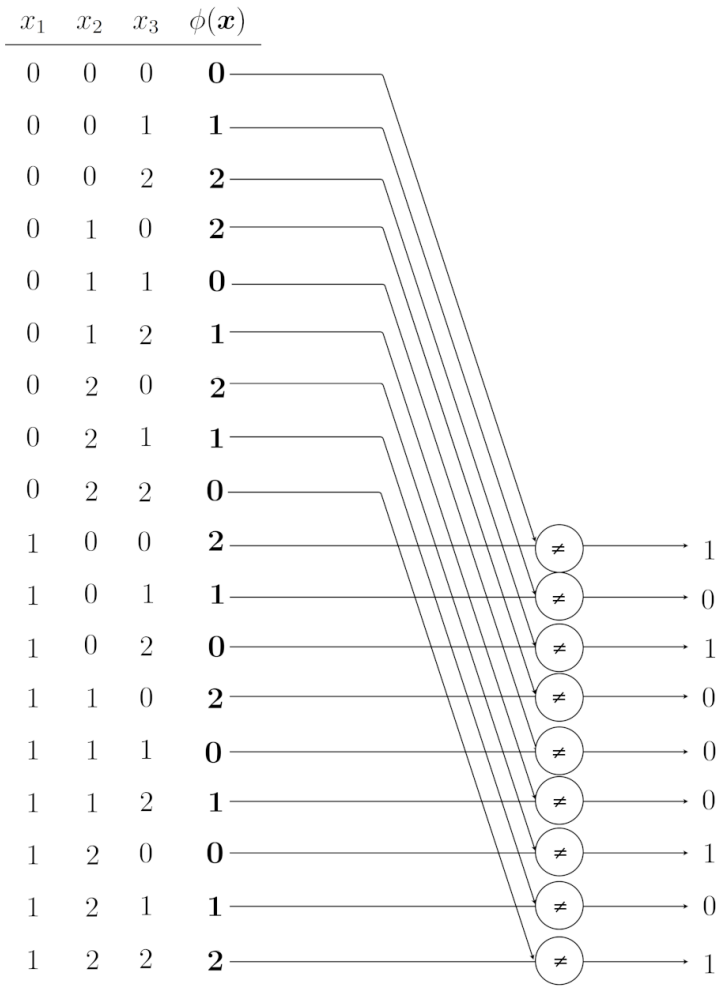

Figure 2.4: Calculation of IDPLD (2.5) for component x_1 of the structure function in Tab. 2.3

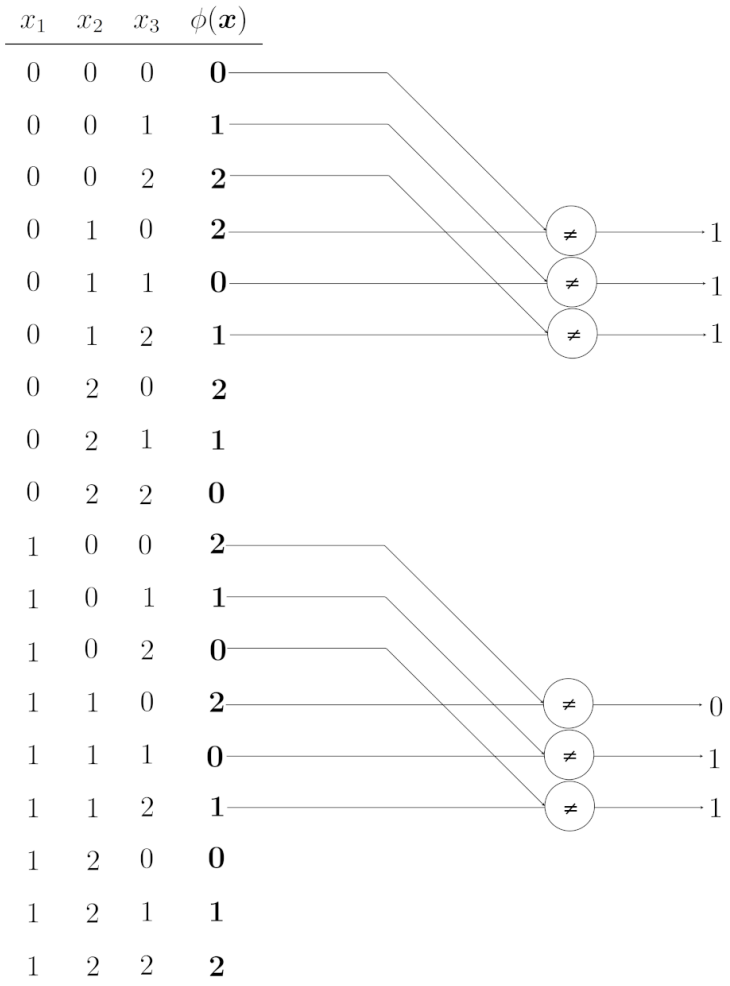

Figure 2.5: Calculation of IDPLD (2.5) for component x_2 of the structure function in Tab. 2.3

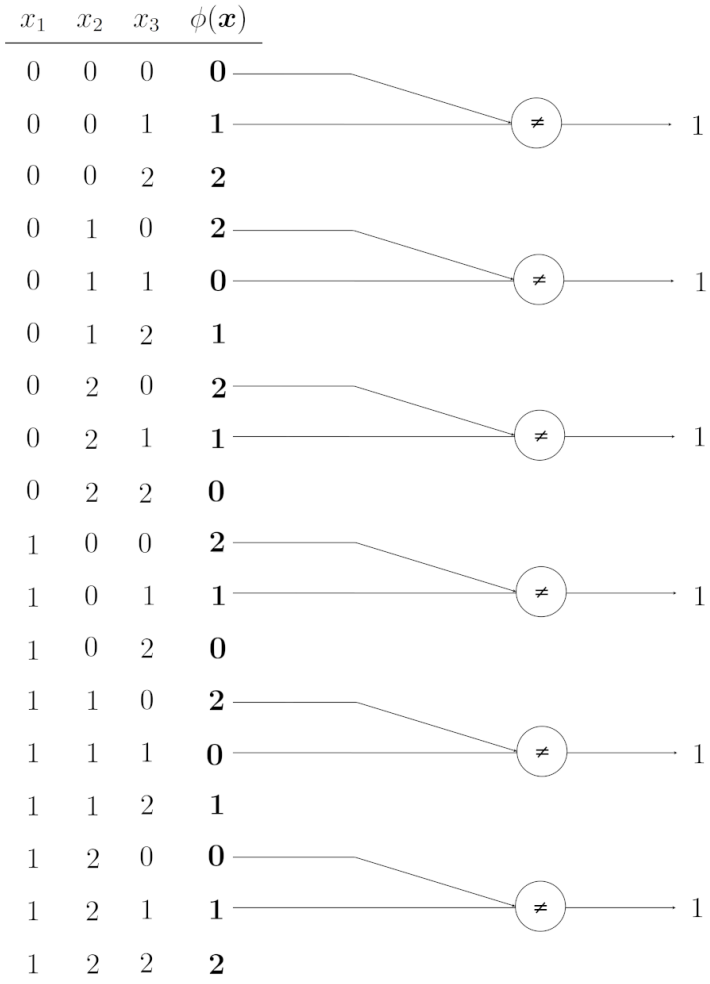

Figure 2.6: Calculation of IDPLD (2.5) for component x_3 of the structure function in Tab. 2.3

The information about system critical states obtained based on DPLD (2.4) and (2.5) can be not sufficient for some application, because the derivatives (2.4) - (2.5) allow indicating the critical system states for indicated component and fixed change of its state. By the other words, these critical states are defined according to the component state change and the system performance level change is secondary. Therefore the derivatives for analysis a certain change of the system performance level should be proposed too. The IDPLD with respect to the i-th variable of the structure function for this function value change from j to h is defined as:

$$\frac{\partial\phi(j\to h)}{\partial x_i}=\begin{cases}1, & \text{if} \quad \phi(s_i,\boldsymbol{x})=j \quad \text{and} \quad \phi(r_i,\boldsymbol{x})=h\\ 0, & \text{otherwise}\end{cases}, \tag{2.6}$$

for any $s,r\in 0,...,m_i-1$ if $s\neq r$.

The IDPLD (2.6) is computed for all possible changes of the i-th variable, which are defined by values s and r. The non zero values of this derivative indicate the state vectors of the structure function for which the change of the i-th variable value leads to the change of the structure function value from j to h. According to non-zero values of this derivative there is a possibility to indicate critical system states for the system performance level change from j to h depending on any change of the i-th component state. This derivative can be defined by other way taking into account the definition of DPLD (1.4):

$$\begin{aligned}\frac{\partial\phi(j\to h)}{\partial x_i}&=\frac{\partial\phi(j\to h)}{\partial x_i\downarrow}+\frac{\partial\phi(j\to h)}{\partial x_i\uparrow}\\ &=\bigvee_{s=1}^{m_i-1}\bigvee_{r=0}^{s-1}\frac{\partial\phi(j\to h)}{\partial x_i(s\to r)}+\bigvee_{s=0}^{m_i-2}\bigvee_{r=s+1}^{m_i-1}\frac{\partial\phi(j\to h)}{\partial x_i(s\to r)}\end{aligned} \tag{2.7}$$

The IDPLD (2.7) consists of two parts. One of them $\partial\phi(j\to h)/\partial x_i\downarrow$ allows indicating the system critical states for which the degradation of the i-th system component results in the system performance change from j to h. The other part of this derivative $\partial\phi(j\to h)/\partial x_i\uparrow$ allows obtaining the system critical states for the system performance level change from j to h caused by the improving of the i-th system component state. Need to note that the derivatives (2.6) and (2.7) allow calculating critical states for specified change of the system performance level, but these derivatives don't provide the information about component state change in this case. The more

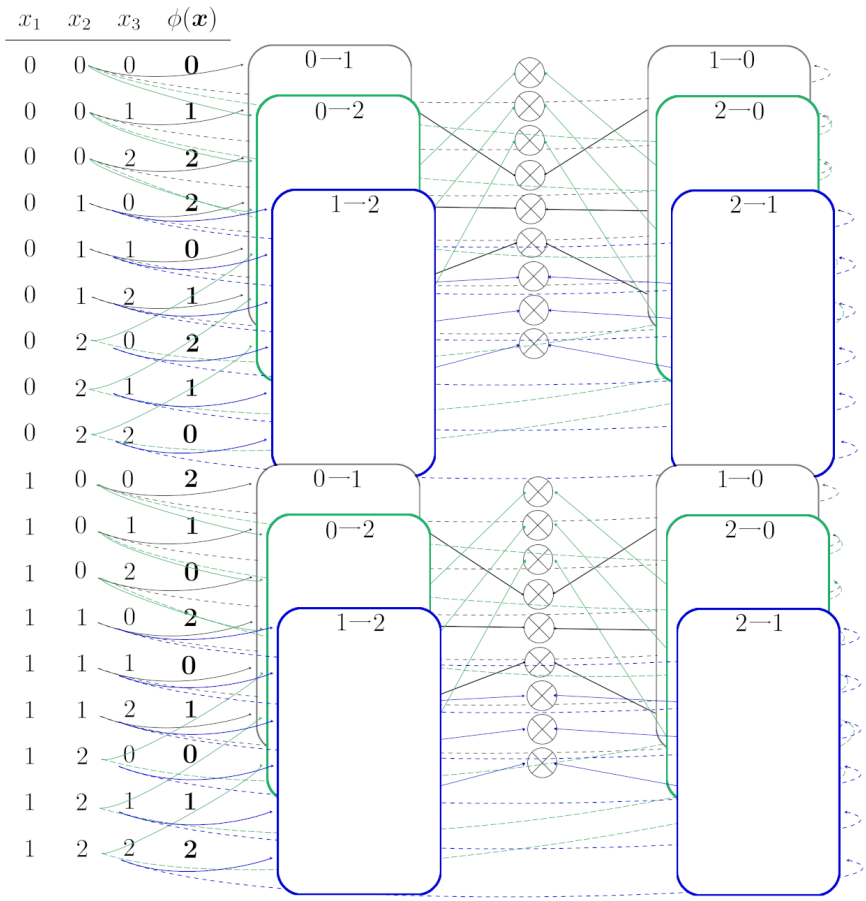

Figure 2.7: Calculation of IDPLD (2.7) for component x_2 of the structure function in Tab. 2.3

detail information in this case can be discovered based on DPLD (1.4) for specified change of the system performance level.

Let us to continue the example with the structure function described in Table 2.3 by the derivative (2.7) and define the critical states for the system degradation from system performance level 2 to 1. These critical states are computed based on the derivatives $\partial\phi(2 \to 1)/\partial x_i$ for $i = 1, 2, 3$. The first component of this system has only two states and analysed by the derivative $\partial\phi(2 \to 1)/\partial x_1$. This derivative doesn't have a non-zero value. The second and third system components have three states and their analysis is implemented based on the derivatives $\partial\phi(2 \to 1)/\partial x_2$ and $\partial\phi(2 \to 1)/\partial x_3$ accordingly. The flow-diagram in Fig. 2.7 illustrates the computation of the derivative for the analysis of the second component. The derivative with respect to the second variable has two non-zero values. It means that there are two critical states depending on the second component for the system degradation from the performance level 2 to performance level 1: $x_1x_2x_3 = (0\ \ (0 \to 1)\ \ 2)$ and $x_1x_2x_3 = (1\ \ (2 \to 1)\ \ 2)$. The derivative $\partial\phi(2 \to 1)/\partial x_3$ has six non-zero values. The critical states for the considered system degradation depend on the changes in the state of the third component are presented in Table 2.4. All possible critical states of this system degradation and failure are shown in Table 2.4. These critical states in this Table are defined based on the IDPLD (2.7). These states can be used in the development of scenarios for the system improving.

For example, according to the Table 2.4 the first system component can cause error if other components works perfectly (the system performance level change from 2 to 0). The change of the third component results in the largest number of the critical states.

2.3 Multi-Valued Decision diagrams

As we mentioned previously, there are many ways to represent SF. Structure function of real systems can have large dimension, it is necessary to represent it in effective way, for example in form of Multi-Valued Decision Diagram (MDD). This diagram is rooted acyclic graph that meets two conditions [62, 63]:

1. graph is canonical - the representation is unique for a particular variable ordering

2. graph is compact - any other graph representation contains more nodes.

Table 2.4: Critical states for structure function described in Tab. 2.3

System state change	Components and their states		
	x_1	x_2	x_3
$0 \to 1$		0 $(1 \to 0)$ 1 1 $(1 \to 0)$ 1 0 $(2 \to 1)$ 2	0 2 $(2 \to 1)$ 1 0 $(2 \to 1)$
$0 \to 2$	$(1 \to 0)$ 0 2 $(1 \to 0)$ 2 0	0 $(2 \to 0)$ 2 1 $(2 \to 0)$ 0 1 $(2 \to 1)$ 0	0 1 $(1 \to 0)$ 1 0 $(1 \to 0)$ 0 2 $(2 \to 0)$ 1 0 $(2 \to 0)$
$1 \to 2$		0 $(1 \to 0)$ 2	0 2 $(1 \to 0)$ 1 0 $(1 \to 0)$ 0 1 $(2 \to 0)$ 1 1 $(2 \to 0)$
$1 \to 0$		1 $(1 \to 0)$ 2 0 $(2 \to 1)$ 1 1 $(2 \to 1)$ 1	0 0 $(1 \to 0)$ 1 2 $(1 \to 0)$ 0 1 $(2 \to 1)$ 1 1 $(2 \to 1)$
$2 \to 0$	$(1 \to 0)$ 0 0 $(1 \to 0)$ 2 2	0 $(1 \to 0)$ 0 0 $(2 \to 0)$ 0 1 $(2 \to 0)$ 2	0 0 $(2 \to 0)$ 1 2 $(2 \to 0)$
$2 \to 1$		1 $(2 \to 1)$ 2	0 0 $(2 \to 1)$ 1 2 $(2 \to 1)$

MDD consists of sink and non-sink nodes. System described by Structure Function (1.1) has exactly m sink nodes labeled by numbers from 0 to $m-1$. These nodes are used to express corresponding system performance level. Non-sink nodes represents system components. Each of them has exactly m_i outgoing edges that are interpreted as a system component state, i.e. outgoing edge of node x_i labeled by value j means i-th system component is in a state j.

Path from root node to sink node expresses system performance level depending on a specific combination of system components performance level. For example highlighted path from Fig. 2.9 (d) is equal to SF $\phi(x_1 = 0, x_2 = 1, x_3 = 0) = 0$ or $\phi(0,1,0) = 0$. This can be expressed as case, when components x_1 and x_3 are not working and component x_2 is working, then system is not working.

MDD is an orthogonal form of SF [64]. Therefore it can be used for the system probabilistic analysis. In this case each edge has weight equal to probability $p_{i,s}$, that the system component i is in state s as can be seen in Fig. 2.9 (d).

2.3.1 MDD representation using vector of neighbours

Let us to consider the MDD representation and definition for the software development. One of these representations is by using vector of neighbours. This representation of MDD consists of 3 vectors as we can seen in Fig. 2.8, namely map vector, vector of indexes and vector of neighbours.

Map vector contains information about mapping nodes of MDD into indexes.

Vector of indexes contains indexes for individual nodes into the vector of neighbours. At index 0 in the vector of indexes there is written start position of neighbours in vector of neighbours for component with index 0. At index 1 there is start position in vector of neighbours for component with index 1 etc. In case node has no neighbours (e.g. sink nodes in MDD) value of next index is not incremented. This can be seen in Fig. 2.8 at index 5 and 6.

Vector of neighbours contains neighbours of individual nodes.

Let us consider this representation of MDD for system with SF defined in Table 2.5. Decision tree representation of this structure function is in Fig. 2.9 (a). As a first step, we create a map vector. At index 0 there is node x_1, etc. Please note, that sink nodes with the same value get the same number when mapping. Vector of indexes and vector

Table 2.5: Example of MSS

x_1	x_2	x_3	$\phi(\mathbf{x})$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

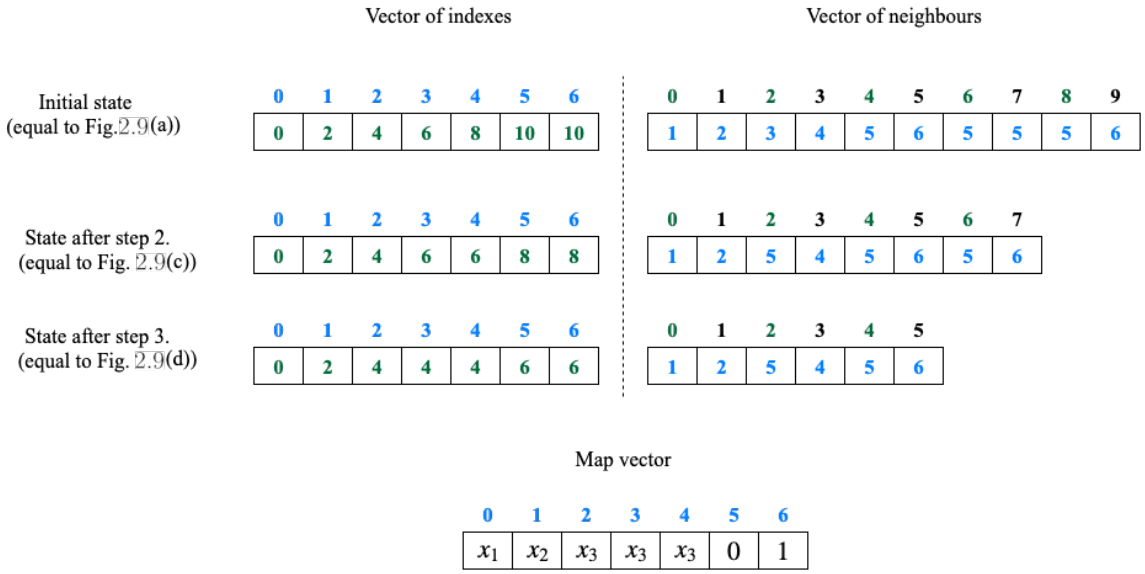

Figure 2.8: DT to MDD reduction using vector of neighbours

of mapping can be seen in Fig. 2.8.

In order to obtain neighbours of node with index 2 in vector of mapping, we retrieve value from the vector of indexes at index 2. This value is 4. That means neighbours of node 2 start at position 4 in the vector of neighbours. To obtain index of last neighbour for node 2 we take value from the next index in vector of indexes. This value is 6. That means, index of the last neighbour for node 2 is at index 5 in the vector of neighbours, i.e. neighbours of node 2 are at indexes 4 and 5 in the vector of neighbours. From map vector we can obtain information about original nodes. Index 2 represents node x_3 from original DT. Neighbours of this node are at indexes 5 and 6, which represents system states 0 and 1.

2.3.2 Algorithm for DT to MDD reduction

In order to transform DT to MDD, two rules has to be applied, specifically merging isomorphic subtrees and deletion of useless nodes [65, 62]. Our algorithm consists of three steps:

1. **Reduce number of sink nodes:**

   Let us take an example of a system consisting of 3 2-state variables. Structure function of this system can be seen in Table 2.5. DT consisting of 5 non-sink nodes and 6 sink nodes can be seen in Fig. 2.9 (a). Every branch of this DT ends in a node describing state of analyzed system. As a first step, we need to merge sink nodes describing common performance level. As a result of this step, we have exactly such number of sink nodes as is the number of system performance levels. Result of this step can be seen at Fig. 2.9 (b).

Algorithm 1 Step 1

```
tmp : array[m]
for all finalNode in finalNodes do
  if tmp[finalNode.state] is null then
    tmp[finalNode.state] = finalNode
  else
    tmp[finalNode.state].parents.add(finalNode.parents)
  end if
end for
```

   Using vector of neighbours representation, this step can be solved by correct mapping as we mentioned in previous section.

2. **Remove nodes, where all outgoing edges end in the same node:**

   In Fig. 2.9 (b), all outgoing edges of left bottom node x_3 end in the node 0. Therefore we can remove node x_3 and redirect its ingoing edges directly into node 0 as can be seen in Fig. 2.9 (c).

   Using vector of neighbours representation, this can be solved by looking for the same numbers in the vector of neighbours for some node. In our example presented in Fig. 2.8, there is the same value for node 3, Therefore we can remove that

Algorithm 2 Step 2

```
for all node in nodes do
  if all node.children are equals then
    for all child in node.children do
      child.parents = node.parents
    end for
    remove node
  end if
end for
```

node. In vector of neighbours, value 3 is replaced with value 5 and neighbours of node 3 are removed.

3. **Remove equal subtrees:**

   In this step we need to search the tree in order to find equal subtrees. As equal subtrees we count only that ones, that ends in the same nodes. In other words there has to be node/nodes in that both subtrees are merged. In our example in Fig. 2.9 (c) equal subtrees are both x_3 nodes. Nodes where both subtrees are merged are 0 and 1. Therefore we can remove one subtree and redirect its ingoing edges into subtree that was not removed as we can see in Fig. 2.9 (d).

   Using vector of neighbours representation, we are looking for the same sequences for node representing the same component of the system. In our example, nodes with numbers 2 and 4 can be merged together.

In the last row of Fig. 2.8, final MDD is written (equal to MDD as can be seen in Fig. 2.9 (d)).

2.4 Incompletely specified structure function of MSS

The problem with traditional methods in reliability analysis, such as methods based on structure function is the necessity of having full information about analyzed system. But in many real systems, we don't have such information, i.e. either we don't know all system components or we don't know system performance level based on system components states in some cases [46]. Therefore different approach has to be used. In

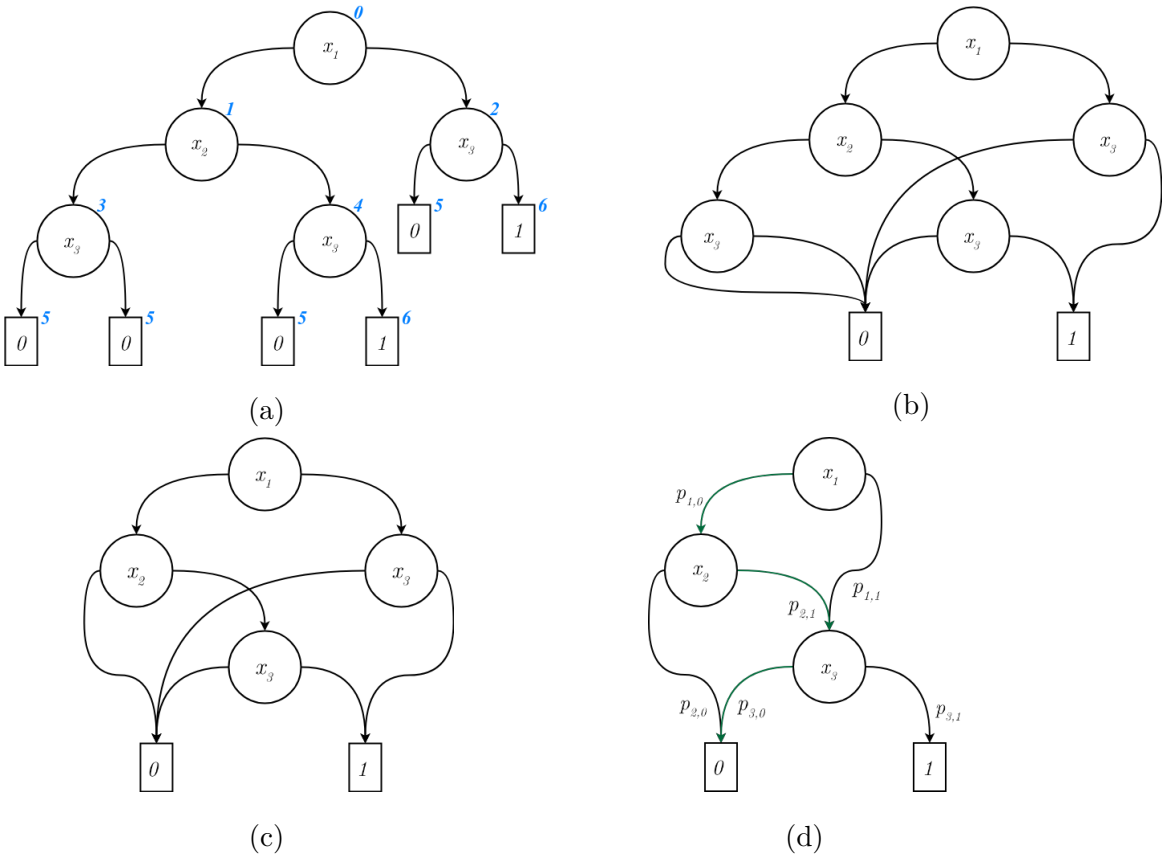

Figure 2.9: Example of DT to MDD reduction

Algorithm 3 Step 3

```
merged = 1
while merged = 1 do
  merged = 0
  for all node1 in nodes do
    for all node2 in nodes do
      if node1 != node 2 then
        if equalSubtrees(node1, node2) then
          node1.parents.add(node2.parents)
          remove node2
          merged = 1
        end if
      end if
    end for
  end for
end while
```

our work, we will use methods of datamining described in [50] and [66]. Using this methods we will construct decision tree (DT) from available data of analyzed system. This tree will be then reduced to multi-valued decision diagram (MDD) [67] that can be used for reliability analysis using classic methods.

3. Software Reliability

The development of methods for the quantification of reliability for software is relevant problem in reliability engineering [68, 69]. The conception of software reliability according to [70] is defined as the probability of a software operates failure-free in a specified environment for a specified exposure period. But probabilistic performance of software cannot be defined as the hardware performance due to the uncertainty in operation. According to this conception a software failure is discrepancy between intended and actual output, and software reliability engineering provides quantification of two things, expected use and desired major quality characteristics. A failure of software can occur at different stages of development and exploitation. But most often analysed stage is software development [70]. One of the possible methods for analyzing software reliability at the development stage is to consider a software testing process [71, 72]. During the testing defects and failures of software are detected, limited and removed. It allows growing of software reliability. The estimation of software reliability based on information from the software testing process, as a rule, is implemented by methods named as Software Reliability Growth Models (SRGMs) [71, 73].

The SRGMs can be considered as stochastic counting processes regarding the number of faults detected or failures experienced in software testing. Based on this data some of software reliability measures are computed as a probability that no failure occurs during a certain time interval. One of the first SRGM is known as Jelinski-Moranda model [74] which assumes that the elapsed time between failures is governed by the exponential probability distribution. There are some developments of this model with application of Markov based methods [71, 75, 72]. The main idea of these models is that software failure is random event (caused by errors in software) and this can be modelled using some well-known probability distribution. For example in Markov model, mainly exponential distribution is used. These models can be easily used to calculate software reliability and, for example predicts software failure using historical information about analysed software. The development of Jelinski-Moranda model named as Non-Homogeneous Poisson Processes (NHPPs) is often used in software reliability too [71, 73, 72]. All these methods of software reliability according to [76, 77] are time-dependent, but not time-data-dependent. In addition the structure or topology

of the software are not taken into account in the analysis based on SRGM [78, 79]. There are several models suitable also to calculate these characteristics of software and will be presented more deeply in the following section. One of them was proposed in paper [2] and can be used in case software is implemented with micro-service architecture. In this case dependence graph of individual services is created and this graph is used to create fault tree. Reliability of each service is calculated using historical data about its functioning and failures. The next model was proposed in paper [3] and is based on UML/DAM diagrams. In this case these diagrams are created as a representation of software, for example use case diagram, deployment diagram and activity diagrams. Using all of the mentioned diagrams fault tree is created. The software time-data-dependent analysis based on fault tree are considered in [79]. Risk analysis method for software evaluation is proposed in[80].

3.1 Software reliability models based on the structure function

Most of models used in software reliability are probabilistic. However there are several ones, that can be used also for analysis of structure of software systems. Although this approach can be currently used only in specific cases and needs more investigations.

3.1.1 Software reliability of microservice architecture

The first case this approach can be used was published in [2] and can be applied in case software is designed using microservice architecture. The difference between traditional and microservice architecture can be seen in Fig. 3.1. In this architecture software is not implemented as a whole, but as individual programs (services), each with its own functions and responsibilities. The whole software is then based on communication between these services. These services can depend on each other, for example to retrieve some data from database, service responsible for login has to approve user can obtain such data, so this service depends on result of login service etc.

In this case, dependency graph can be constructed. This graph describes dependencies between individual services. Example of such graph can be seen in Fig. 3.2 (a). Using this graph, together with information about system, fault tree can be created. Thanks to the fact fault tree is one of the representations of the structure function, it is

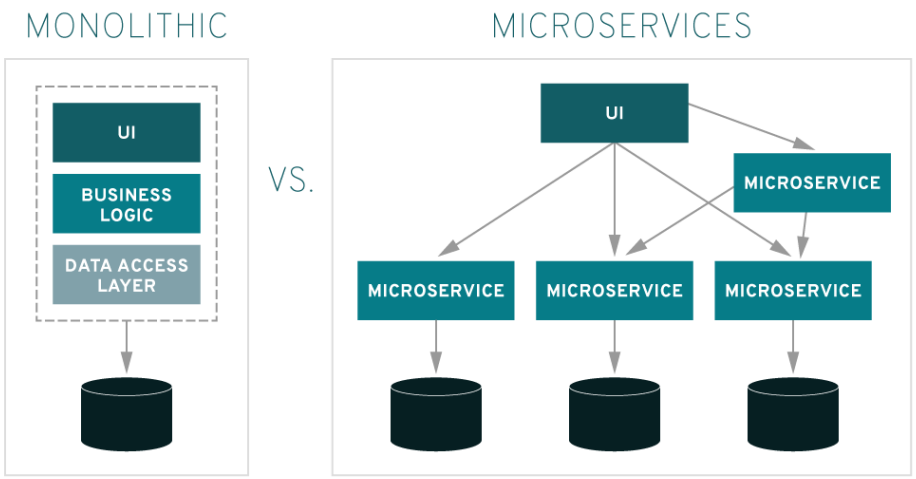

Figure 3.1: Monotilic vs microservice architecture [1]

possible to analyse software based on microservice architecture from reliability point of view using traditional methods. The only remaining problem is how to evaluate characteristics of individual service. This can be achieved using historical data of behavior of analysed service using tools of probability theory [2].

3.1.2 Model example - e-shop

Let us assume the following example. We need to analyse e-shop from reliability point of view. This e-shop is implemented using microservice architecture and consists of the following services: Web interface, Account management service, Stock management service and Orders management service. In order to increase reliability, some of services are duplicate, specifically Account management service and Orders management service. Let us assume, the probabilities of working/failure for each service match to information in Table 3.1. There is also map from service names to components in this table, in order to simplify work with them.

As we mentioned before, the first step is to create service dependency graph of analysed system. This can be seen in Fig. 3.3. Please note, that as we have some duplicity services, only one of them is required to work in order system to work.

This graph can be in the next step transformed directly to fault tree, as can be seen in Fig. 3.4. In case there is “or” in dependency graph, it is also present in fault tree. Otherwise connections in dependency graph results to “and” in fault tree.

Created fault tree can be used for both - calculation of system characteristics and

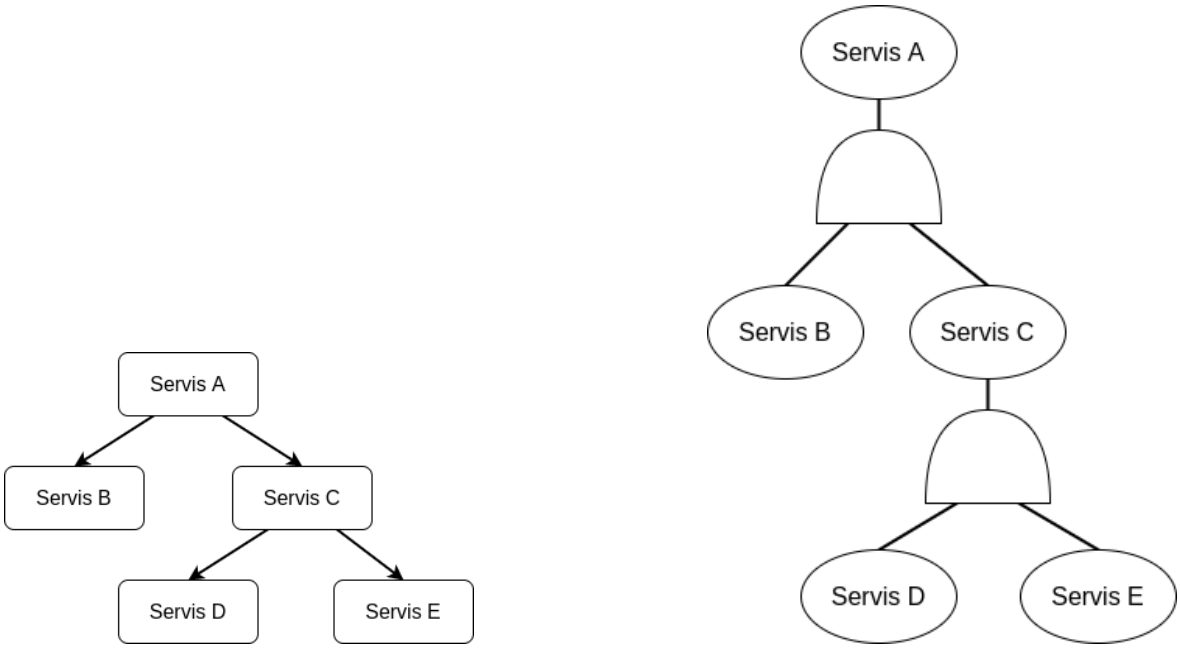

Figure 3.2: Service dependency graph and corresponding Fault Tree [2]

Table 3.1: Probabilities of states in e-shop example

Service name	Component	p	q
Web interface	x_1	0.85	0.15
Account management service 1	x_2	0.98	0.02
Account management service 2	x_3	0.9	0.1
Stock management service	x_4	0.9	0.1
Orders management service 1	x_5	0.0.7	0.3
Orders management service 2	x_6	0.8	0.2

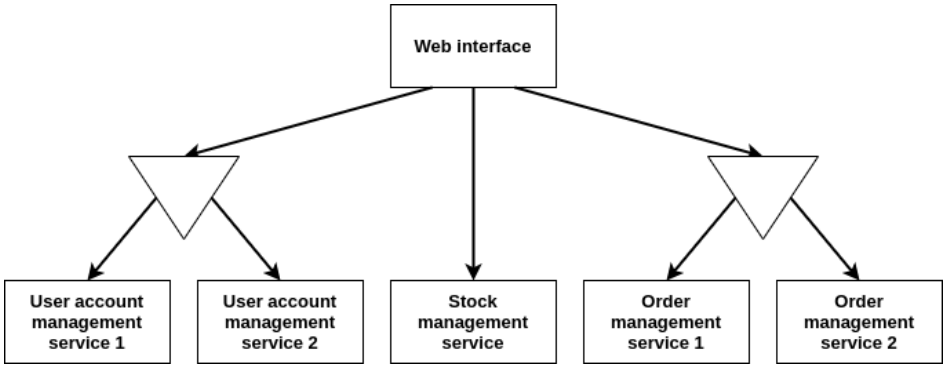

Figure 3.3: Dependency graph for e-shop example

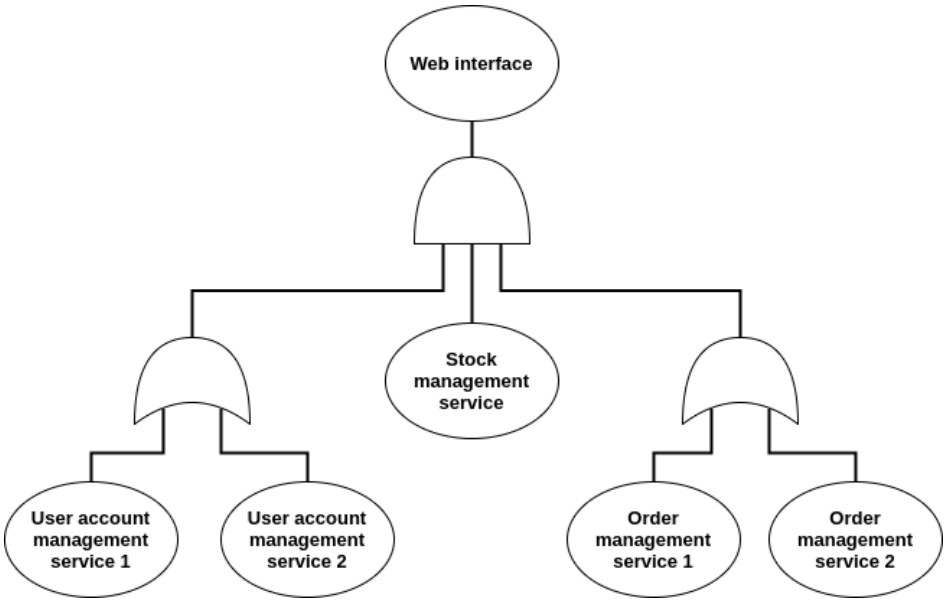

Figure 3.4: Fault tree for e-shop example

Table 3.2: Important measures for e-shop example

component	SI_i	BI_i
x_1	0.28	0.84
x_2	0.09	0.07
x_3	0.09	0.07
x_4	0.28	0.79
x_5	0.09	0.15
x_6	0.09	0.22

importance measures. Reliability function of this system can be seen in equation 3.1. Results of Structure importance and Birnbaum's importance can be seen in Table 3.2.

$$\begin{aligned} R &= p_1 \times ((p_2 + p_3 - p_2 \times p_3) \times p_4 \times (p_5 + p_6 - p_5 \times p_6) \\ R &\approx 0.717 \end{aligned} \tag{3.1}$$

3.1.3 Software reliability calculation based on UML/DAM diagram

The next case in that the structure function based model can be used in software reliability was published in [3]. This method uses software described using UML model anotated with DAM stereotypes to create fault tree. Example of such model can be

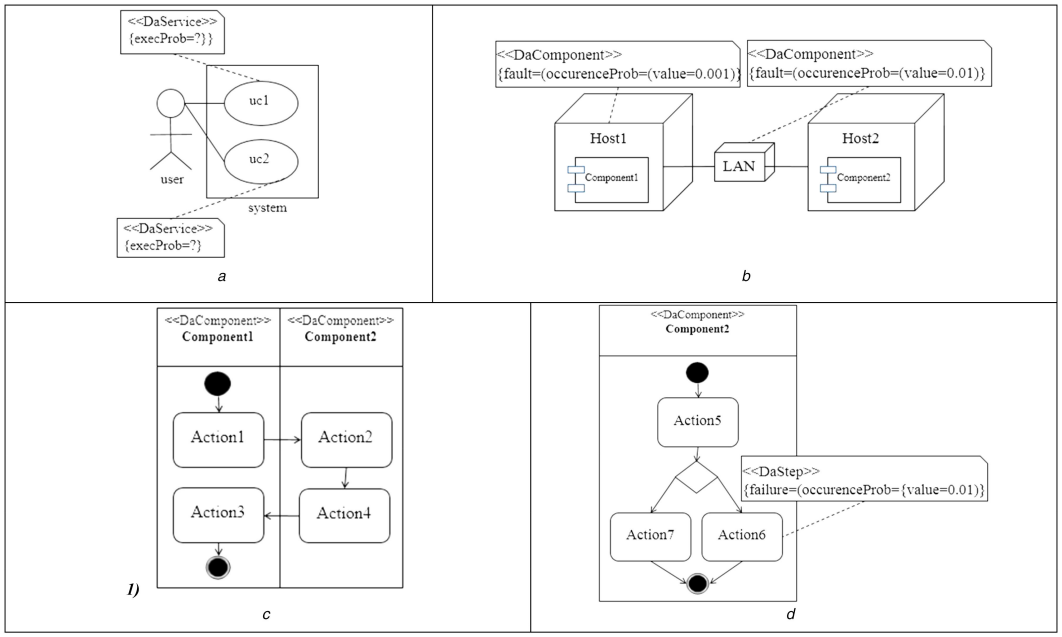

Figure 3.5: Example of UML/DAM diagrams [3]

seen in Fig. 3.5. This diagram describes software in details and contains information about use cases, realizations of these use cases via activity diagrams and how they are realized physically using deployment diagram. Each of these diagrams is annotated with information about probabilities of success/failure of individual parts.

These diagrams are used in the next step to create reliability model, specifically fault tree. Example of fault tree corresponding to UML/DAM diagram described in Fig. 3.5 can be seen in Fig. 3.6. Each use case is a branch in this tree. The use case has following nodes that describes its realization. The use case itself fails in case any part of its realization fails.

The created fault tree can be then used to analyse software from reliability point of view. The main problem of this method is correct parameter estimation, which is in this case probabilities of success/failure of individual components. These can be estimated based on opinions of experts, using evidence theory [3], etc.

3.1.4 Model example - calculator

Let us assume the following example. We need to analyse simple calculator from reliability point of view. This calculator has the basic functionality: add, subtract, multiply and divide two numbers. This is described in Use Case diagram as can be seen in Fig. 3.7 together with DAM information of probabilities of execution. This can for

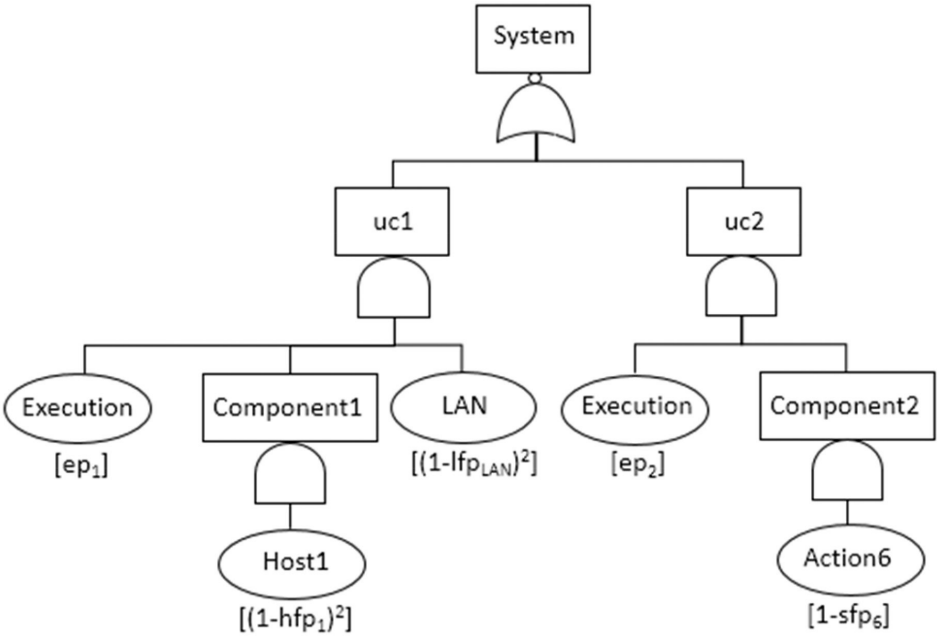

Figure 3.6: Example of Fault tree corresponding to diagram in Fig. 3.5 [3]

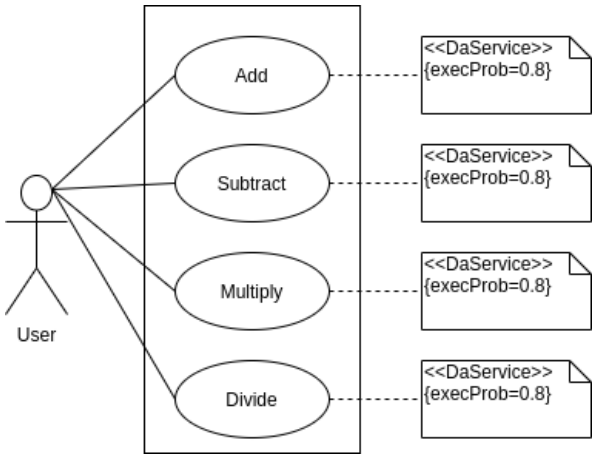

Figure 3.7: Calculator - use case diagram

example mean the human failure when entering inputs. Each of these use cases has its own activity diagram as can be seen in Fig. 3.8 together with DAM information of probabilities of failure. In this example no deployment diagram is necessary.

There are several options in creation of fault tree from these diagrams. In case we specify failure of this calculator as case when any of these use cases fails, fault tree can look as can be seen in Fig. 3.9. In this case operation is successful, when both execution of the use case and its realization in activity diagram success and the calculator is successful when all of the operations are operates successfully. Therefore the whole fault tree contains only and operations between nodes. Reliability function of this case

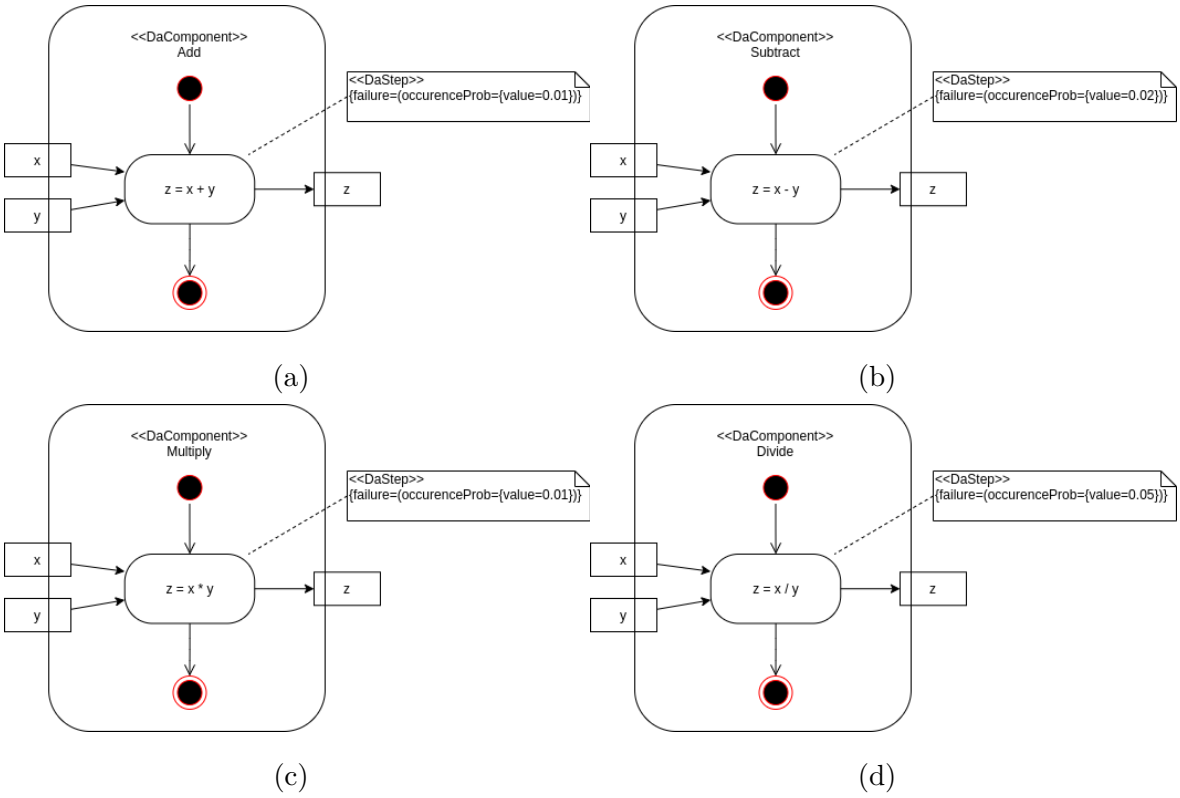

Figure 3.8: Calculator - activity diagrams

is following:

$$R = p_1 \times p_2 \times p_3 \times p_4 \times p_5 \times p_6 \times p_7 \times p_8$$
$$R \approx 0.3737 \tag{3.2}$$

The other option is to specify failure of this calculator as case, when none of the functionality is performed correctly. Fault tree of this case can be seen in Fig. 3.10. The only difference between these two trees is the top operation, which is in this case

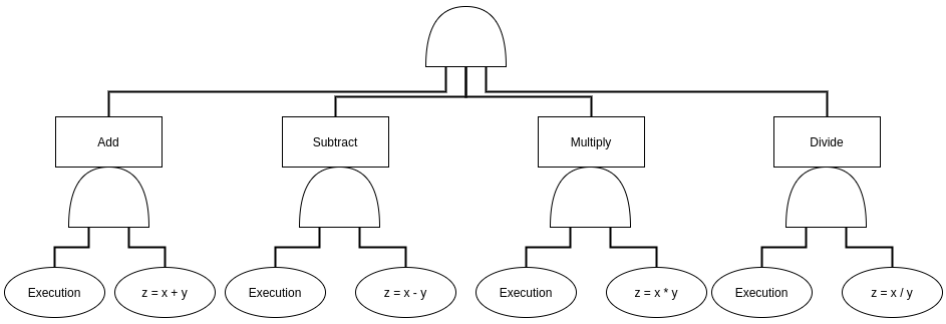

Figure 3.9: Calculator - fault tree case 1

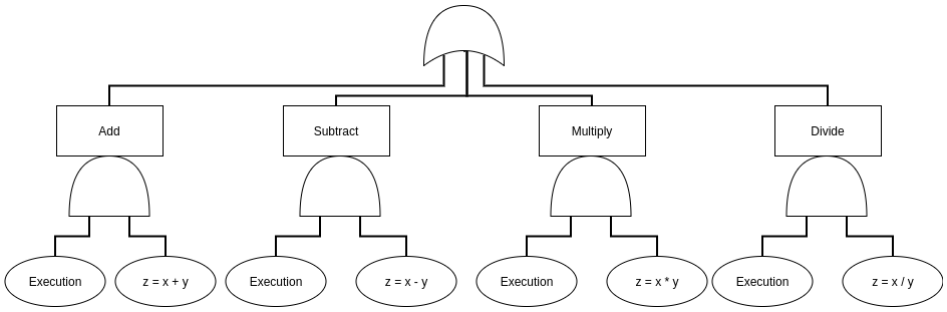

Figure 3.10: Calculator - fault tree case 2

"or" instead of "and". Reliability function in this case is following:

$$\begin{aligned} R = {} & p_1p_2 + p_3p_4 + p_5p_6 + p_7p_8 - p_1p_2p_3p_4 - p_1p_2p_5p_6 - p_1p_2p_7p_8 - p_3p_4p_5p_6 \\ & - p_3p_4p_7p_8 - p_5p_6p_7p_8 + p_1p_2p_3p_4p_5p_6 + p_1p_2p_3p_4p_7p_8 + p_1p_2p_5p_6p_7p_8 \\ & + p_3p_4p_5p_6p_7p_8 - p_1p_2p_3p_4p_5p_6p_7p_8 \\ R \approx {} & 0.9977 \end{aligned} \tag{3.3}$$

3.2 Software reliability model based on the source code

According to studies in[76, 80, 79] topological properties of software can be take into account mainly based on data-depend and structure-depend model of software. This approach is developed in this section. The main idea of this method is to use a source code and syntax tree to create software reliability model. The principle of this method can be seen in Fig. 3.11. This method consists of 2 essential steps. In the first step, source code is used to construct abstract syntax tree and the second is to construct a reliability model using created AST.

Abstract Syntax Tree (AST) is an abstract graph representation of the source code. This representation has many usages in different fields of computer science, for example plagiarism detection [81, 82], code evolution [83], summarization [84], etc. [85]. AST consists of nodes, where each of them represents one element of the source code. Elements in the same block of code are placed on the same level of the syntax tree. In case element of the code can be divided into parts, these parts will be placed as nodes in the next level and all of them will be connected to element this elements consists of. Therefore AST can be easily used to present a structure of the source code and to remove unnecessary information from it, such as gaps, variable names, etc. [82]. That

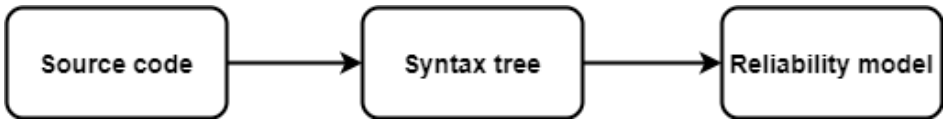

Figure 3.11: Principle of proposed method

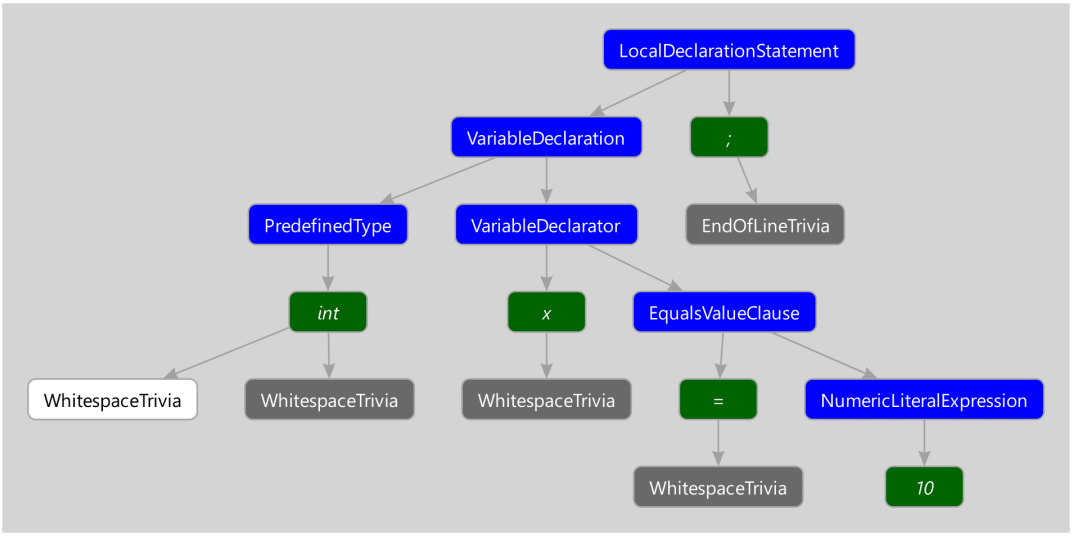

Figure 3.12: Syntax tree of local declaration statement

will be unnecessary for the purposes of reliability analysis. Using this representation of source code instead of using source code itself has also other advantages. One of them is, language independence. As AST is an abstract representation, using this form to create reliability model allows us to analyse software implemented in any programming language (as long as we are able to create AST from it). Example of AST can be seen in Fig. 3.12.

The next step of this method is to construct a reliability model using created AST. For this purpose we decide to use fault tree.

Fault tree is an acyclic graph that consists of 2 types of nodes: events and gates. An event is an occurrence within the system, that typically describes a failure or a degradation of subsystem or a degradation of component of the system. These events are connected via logic gates, typically AND or OR gates. These events together describes situation, when system fails [86, 87]. Fault tree can be easily transformed into structure function, that allows us to represent system as a Boolean function and use methods for its analysis such as logic differential calculus to perform reliability analysis [88].

4. Case Studies

The usage of all previously mentioned methods are demonstrated in this chapter. Section 4.1 presents example of anesthesia examination. This system is used to present calculus of critical states using proposed IDPLD of non-coherent MSS. The next two case studies, i.e. patients with hepatitis survival chance and bike crashes, described in sections 4.2 and 4.3 respectively, demonstrate reliability evaluation together with quantitative analysis for incompletely specified structure function of MSS and the usage of the MDD in this evaluation. The last case study, described in section 4.4 presents all steps in reliability analysis of software, i.e. creation of mathematical model from source code, usage of AST and quantitative analysis of this model.

4.1 Anesthesia examination

Evaluation of the medical error during a surgery depends on the anesthetic examination. As a rule in medical practice, several examinations has to be performed before surgery and one of them is needed to determine an anesthesia type which will be used. Anesthesia type is caused by patient state, his allergy etc. This examination is performed using several medical tests, forms etc. The anesthetic examination of a patient determines what type of anesthesia can be applied during surgery. It is common that the preoperative examination and anesthesiology during surgery is performed by two different doctors: the examination is done by one doctor and another one is responsible for anesthesia during surgery. The result of examination is written by the first doctor and the second doctor has to correctly interpret these results.

The simplified version of this system can be interpreted as a system of three components ($n = 3$): the examination result, the work of the first doctor before surgery and the work of the second doctor (anesthesiologist). The first component, x_1, is an examination about the anesthesia type. This will be introduced in the considered model as a component with two states $m_1 = 2$. State 1 means examination is correct. That means results of the examination corresponds with the expected results and a patient can take the chosen type of the anesthesia without harm. State 0 will mean examination is not correct (due to mistake in forms or other reasons), i.e. if chosen type of the anesthesia is harmful for patient, but the examination results denote this

Table 4.1: Structure function of medical error during a surgery depending on the anesthetic examination

x_1	x_2	x_3	$\phi(\boldsymbol{x})$
0	0	0	0
0	0	1	0
0	0	2	2
0	1	0	0
0	1	1	1
0	1	2	1
0	2	0	2
0	2	1	1
0	2	2	0
1	0	0	2
1	0	1	0
1	0	2	0
1	1	0	0
1	1	1	1
1	1	2	1
1	2	0	0
1	2	1	1
1	2	2	2

type of anesthesia as the correct one for patient.

The second system component x_2 represents the first doctor's work performing the preoperative examination. This doctor has to take the results of this examination, interpret them and summarise the results in order the last doctor doesn't have to study all results of examination to select a correct type of anesthesia. This will be modeled using three states $m_2 = 3$. State 2 will mean doctor interpret results of examination correctly. State 1 will mean doctor make small mistakes in interpretation but these mistakes are not deadly for patient. The state 0 will mean the doctor interprets the results incorrectly and if this type of anesthesia is taken, it will be deadly for patient.

The last component x_3 permits to describe the second doctor's work responsible for the anesthesia during surgery. This doctor has to take results written by the first doctor and depending on them give to the patient a correct type of anesthesia. This will be modeled also using 3 states. State 2 will mean doctor interprets results correctly and gives patient that type of anesthesia that is shown by the results. State 1 will mean doctor makes small mistake and pick anesthesia that is harmful for patient but not deadly. State 0 will mean the doctor picks a wrong type of the anesthesia.

This system with three states ($m = 3$) is whether the patient receives anesthesia, that is not harmful at all for him (state 2) or that type of anesthesia that is for him harmful but not deadly (state 1) or anesthesia that is deadly for him (state 0). The structure function of this system can be defined based on the expert evaluation and in the form of truth table it can be seen in Table 4.1.

Critical states of this system can be calculated using the IDPLDs (2.5) and (2.7). The first of these derivatives allows indicating of critical states depending of specified change of the component state (Table 4.2). According to the Table 4.2 there are five critical states for the first component failure and five critical state for the component recover. It means that there are five situation when the mistake in the examination about the anesthesia type results the change of a surgery depending on the anesthetic examination. These critical states show that the incorrect type of the anesthesia has influence on the result of surgery if one of the doctors is wrong or both of them are right. The mistake of the first doctor (the second system component) results in surgery's problem if the type anaesthesia is not correct and the second doctor makes mistake or works correctly. The mistake of the second doctor (anesthetist) has strong influence on

Table 4.2: Critical states for anesthesia examination according to Eq. 2.5

System state change	Components and their states		
	x_1	x_2	x_3
$0 \to 1$	((0 → 1) 0 0) ((0 → 1) 0 1) ((0 → 1) 0 2) ((0 → 1) 2 0) ((0 → 1) 2 2)	(0 (0 → 1) 0) (0 (0 → 1) 2) (1 (0 → 1) 1) (1 (0 → 1) 2)	(0 1 (0 → 1)) (0 2 (0 → 1)) (1 0 (0 → 1)) (1 1 (0 → 1)) (1 2 (0 → 1))
$0 \to 2$		(0 (0 → 2) 0) (0 (0 → 2) 1) (0 (0 → 2) 2) (1 (0 → 2) 0) (1 (0 → 2) 2)	(0 0 (0 → 2)) (0 1 (0 → 2)) (0 2 (0 → 2)) (1 0 (0 → 2)) (1 1 (0 → 2)) (1 2 (0 → 2))
$1 \to 2$		(0 (1 → 2) 1) (0 (1 → 2) 2) (1 (1 → 2) 0) (1 (1 → 2) 1) (1 (1 → 2) 2)	(0 0 (1 → 2)) (0 2 (1 → 2)) (1 0 (1 → 2)) (1 1 (1 → 2)) (1 2 (1 → 2))
$1 \to 0$	((1 → 0) 0 0) ((1 → 0) 0 1) ((1 → 0) 0 2) ((1 → 0) 2 0) ((1 → 0) 2 2)	(0 (1 → 0) 0) (0 (1 → 0) 2) (1 (1 → 0) 1) (1 (1 → 0) 2)	(0 0 (1 → 2)) (0 1 (1 → 2)) (0 2 (1 → 2)) (1 0 (1 → 2)) (1 1 (1 → 2))
$2 \to 0$		(0 (2 → 0) 0) (0 (2 → 0) 1) (0 (2 → 0) 2) (1 (2 → 0) 0) (1 (2 → 0) 2)	(0 0 (2 → 0)) (0 1 (2 → 0)) (0 2 (2 → 0)) (1 0 (2 → 0)) (1 1 (2 → 0)) (1 2 (2 → 0))
$2 \to 1$		(0 (2 → 1) 1) (0 (2 → 1) 2) (1 (2 → 1) 0) (1 (2 → 1) 1) (1 (2 → 1) 2)	(0 0 (2 → 1)) (0 2 (2 → 1)) (1 0 (2 → 1)) (1 1 (2 → 1)) (1 2 (2 → 1))

Table 4.3: Critical states for anesthesia examination according to Eq. 2.7

System state change	Components and their states		
	x_1	x_2	x_3
$0 \to 1$		(0 **0** 1) (0 **1** 1) (0 **2** 2) (1 **0** 2) (1 **1** 1)	(1 0 **2**) (1 1 **1**) (1 1 **2**) (1 2 **0**)
$0 \to 2$	(**0** 0 0) (**0** 2 2) (**1** 0 2) (**1** 2 0)	(0 **0** 0) (0 **2** 2) (1 **0** 2) (1 **2** 0)	(0 0 **0**) (0 0 **1**) (0 1 **1**) (0 2 **2**) (1 0 **2**) (1 1 **1**) (1 2 **0**)
$1 \to 2$		(0 **1** 2) (1 **1** 2)	(0 1 **2**) (0 2 **1**) (1 0 **1**) (1 1 **2**) (1 2 **1**)
$1 \to 0$		(0 **1** 2) (0 **2** 1) (1 **0** 1) (1 **1** 2) (1 **2** 1)	(0 1 **2**) (0 2 **1**) (1 0 **1**) (1 1 **2**) (1 2 **1**)
$2 \to 0$	(**0** 0 2) (**0** 2 0) (**1** 0 0) (**1** 2 2)	(0 **0** 2) (0 **1** 0) (0 **2** 0) (1 **0** 0) (1 **1** 0) (1 **2** 2)	(0 0 **2**) (0 1 **0**) (0 2 **0**) (1 0 **0**) (1 1 **0**) (1 2 **2**)
$2 \to 1$		(0 **0** 2) (1 **2** 2)	(0 1 **0**) (0 2 **0**) (1 0 **0**) (1 1 **0**) (1 2 **2**)

the surgery too. All critical states are shown in Table 4.2 for this system component.

The analysis of the critical states depending on the system performance level change is implemented based on derivatives (2.7) and results are shown in Table 4.3. In terms of our example failure of the anesthesia examination tests leads to patient's death in case it fails and both of the doctors perform perfectly or none of them performs correctly. Failure of the anesthesia examination tests leads to the correct type of anesthesia in case exactly one of the doctors fails. For the component x_1 there are no critical state vectors for other changes of system state, namely for changes $0 \to 1$, $1 \to 2$, $1 \to 0$ and $2 \to 1$. The first component has two states only therefore to define the detail of the component changes the additional investigation are not needed. But in case of component with more number of states it can be needed. For example, for the critical state (0**1**1 for system performance level change $0 \to 1$ the direction of the second component should be investigated by two derivatives DPLDs $\partial\phi(0 \to 1)/\partial x_2(1 \to 0)$ and $\partial\phi(0 \to 1)/\partial x_2(1 \to 2)$, which calculated according to (1.4).

According to the Table 4.2 and Table 4.3 we can see that the most number of critical states have the second and the third system components which describe the work of two doctors. Therefore the doctors' mistakes have more influence on the surgery result.

4.2 Hepatitis dataset

In this section, we will present reliability analysis calculus of patient survival chance based on dataset obtained from repository [89]. This dataset has been used in paper [90] to present method for minimization of variables needed to create Multi-Valued Decision Function. This function can be then used for reliability analysis. We are using different approach to analyse this system - using methods of datamining a decision tree is created [50, 66] that is in the next step reduced using proposed algorithm to MDD. Final MDD is then used for importance measures calculation.

Dataset we use for presenting this method consists of 19 attributes. The final state of system is represented by binary variable. Value 1 represents case, when patient will survive illness and vice versa. Therefore, also our modeled system is two-stated although not all attributes are binary.

6 of 19 attributes are multi-valued, specifically:

- Age: 10, 20, 30, 40, 50, 60, 70, 80

- Bilirubin: 0.39, 0.80, 1.20, 2.00, 3.00, 4.00
- Alk phosphate: 33, 80, 120, 160, 200, 250
- SGOT: 13, 100, 200, 300, 400, 500
- Albium: 2.1, 3.0, 3.8, 4.5, 5.0, 6.0
- Protime: 10, 20, 30, 40, 50, 60, 70, 80, 90

and the remaining 13 variables are two-valued, specifically:

- Sex
- Steriod
- Ativirals
- Fatigue
- Malaise
- Anorexia
- Liver big
- Liver firm
- Spleen palpable
- Spiders
- Ascites
- Varices
- Histology

The whole dataset contains 155 entries as vector of attributes values and a final state value. Only 80 of these entries are fully specified, i.e. we know value of every attribute.

We will present 2 approaches in calculus of this system. The first one will be by using only 80 fully specified entries, similarly as was presented in the paper [90]. The another one will be by using all 155 entries.

4.2.1 Calculus using only fully specified entries

As we mentioned in previous section, in order to create MDD we firstly need to build DT. Then we reduce DT to MDD using algorithm described in section 2.3.2. It turned out, not all variables are relevant, i.e. not all system components have an impact on system state, but only 7 of them, specifically: Protime, Spiders, Ascites, Age, Alk phosphate, Sex and Antivirals. The result MDD has 9 non-sink nodes, as we can see on Fig 4.1.

This diagram was used to calculate Structure Importance for each component. Calculated results can be seen in Fig. 4.2. On the x axis, there are system component

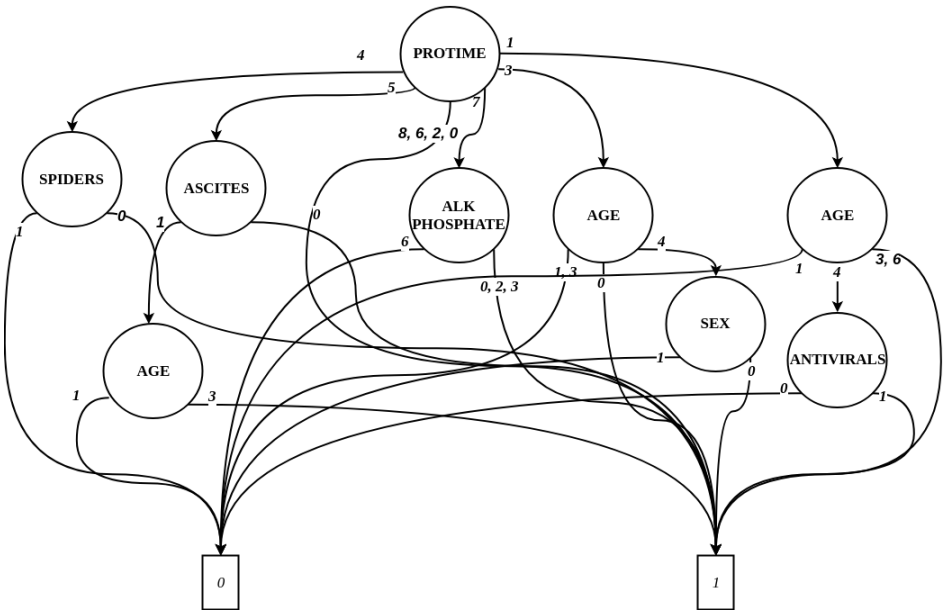

Figure 4.1: MDD from hepatitis dataset using only fully specified records

changes and system performance level changes. The y axis contains values of SI for individual components. The highest number of SI means, the given component is most important for whole system performance and vice versa, value 0 of SI for some component means that given component has no impact on system performance. The highest values from Fig. 4.2 along with system component changes and system changes are following:

- Protime
  - $40 \rightarrow 10$, patient state changes from $Dead \rightarrow Live$
  - $40 \rightarrow 100$, patient state changes from $Dead \rightarrow Live$
  - $60 \rightarrow 40$, patient state changes from $Live \rightarrow Dead$
  - $80 \rightarrow 40$, patient state changes from $Live \rightarrow Dead$

  SI = 0.0902778
- Ascites
  - $Yes \rightarrow No$, patient state changes from $Dead \rightarrow Live$

  SI = 0.0486111
- Spiders

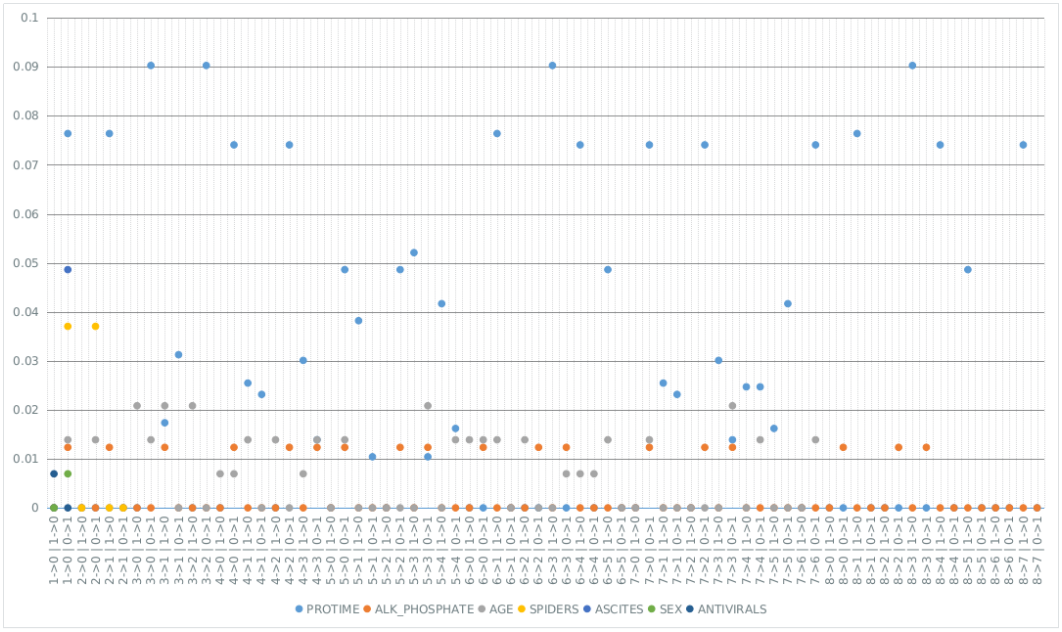

Figure 4.2: SI results for hepatitis dataset using only fully specified records

– $Yes \rightarrow No$, patient state changes from $Dead \rightarrow Live$

SI = 0.037037

4.2.2 Calculus using all entries

Differently we can take all entries into account instead of using only fully specified ones. In this case, after applying presented methods, resulting diagram can be seen in Fig. 4.3. In this case 9 attributes are relevant for system performance, specifically: Albumin, Protime, Spiders, Age, Alk phosphate, Ascites, Bilirubin, Live firm and Anorexia. The result MDD has 17 non-sink nodes. Also note, that a new system state has to be added. This state can be interpreted as a case we cannot decide whether patient will survive or not and probably another tests has to be done.

Results calculated using diagram from Fig. 4.3 are shown in Fig. 4.4 with similar meaning as it was in previous figure. The biggest values of SI are following:

- Albumin
  - $5 \rightarrow 2.1$, patient state changes from $Live \rightarrow Dead$
  - $6 \rightarrow 2.1$, patient state changes from $Live \rightarrow Dead$
  - $6.4 \rightarrow 2.1$, patient state changes from $Live \rightarrow Dead$

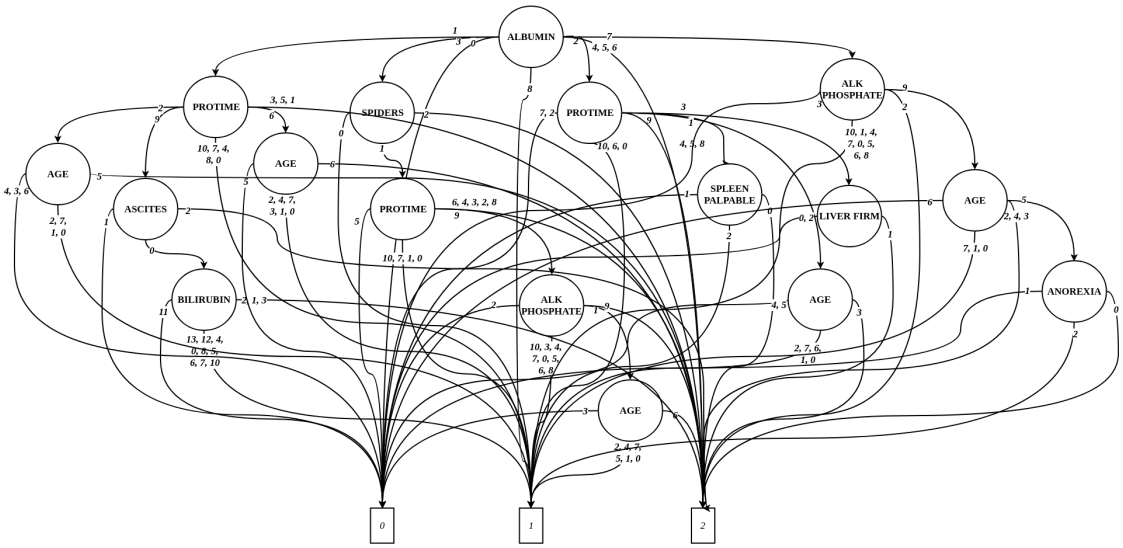

Figure 4.3: MDD from hepatitis dataset using all records

SI = 0.125

- Spiders
  - $Yes \rightarrow No$, patient state changes from $Live \rightarrow Unknown$

  SI = 0.0417

- Protime
  - $70 \rightarrow 10$, patient state changes from $Live \rightarrow Unknown$

  SI = 0.0227

4.3 Bike crashes dataset

In this section we will demonstrate application of described methods in dataset obtained from [91]. This dataset contains information about bicycle crashes such as driver age, whether ambulance was called or not, what was the weather like etc., together with information if bicycle driver was killed, injured or without injure. The whole dataset consists of 11 attributes and 162 records. In order to simplify manipulation with data, each attribute value was mapped into numeric value. Complete list of attributes together with possible values and their mapping can be seen in Table 4.4.

Table 4.4: Details of Bike Crashes Dataset

Attribute Name	Number of values	List of Values (mapped into state)
Ambulance	2	Yes (1), No (0)
Age	7	$0-10$ (0), $11-19$ (1), $20-29$ (2),... 70+ (7)
Bike Direction	4	With Traffic (0), Facing Traffic (1), Not applicable (2), Unknown (3)
Bike Position	7	Travel Line (0), Sidewalk / Crosswalk / Driveway Crossing (1), Non-Roadway (5), Bike Lane / Paved Shoulder (6), Driveway / Alley (3), Multi-use Path (4), Unknown (2)
Sex	2	Male (0), Female (1)
Biker Alcohol	2	Yes (1), No (0)
Driver Alcohol	3	Yes (2), No (0), Missing (1)
Driver speed	8	$0-10$ (0), $11-20$ (1), $21-30$ (2),...,$61-65$ (6), Unknown (7)
Light Condition	5	DayLight (3), Dark - No Light (1), Dark - Lighted (2), Dusk (0), Unknown (4)
Road Surface	5	Croashed Asphalt (4), Smooth Asphalt (3), Concrete (0), Gravel (1), Other (2)
Weather	3	Clear (2), Cloudy (1), Rain (0)

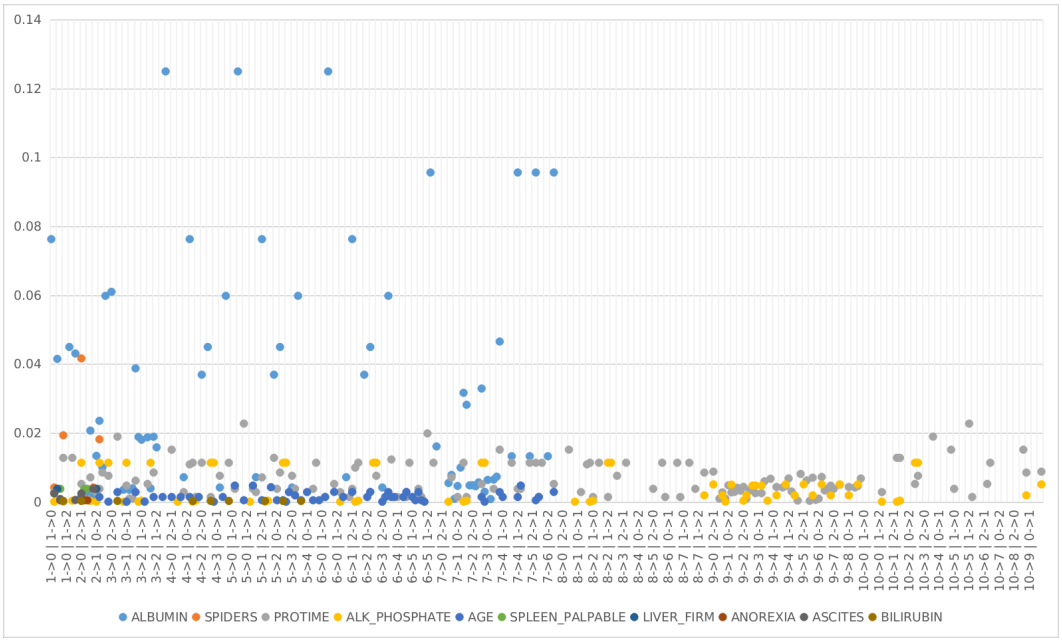

Figure 4.4: SI results for hepatitis dataset using all records

In order to handle incompleteness of these data, methods of datamining presented in [50, 66] was used. Using these methods, decision tree was created. This tree was in the next step reduced into MDD using methods described in paper [61, 65, 62] and in the section 2.3.2 of this thesis. Resulting MDD can be seen in Fig. 4.5. Created mathematical model is in the next step used to perform quantitative analysis.

4.3.1 Topological analysis

In the first step, created mathematical model was used to perform topological analysis, specifically SI for each component was calculated. According to eq. 1.11. From definition of this equation, different DPLDs can be used to calculate SI. Firstly we take into account only cases, when decrease of system component state leads to decrease of system performance. This corresponds to DPLD $\frac{\partial\phi(j\to h)}{\partial x_i(s\to r)}$, where $h < j \wedge s < r$. Results of this calculation can be seen in Fig. 4.6. The x axis contains individual component and system changes. On the y axis there are values of calculated SIi for every component. So for example values in column "$2 \to 0|1 \to 0$" is equal to value of SI, when system component state changes from 2 to 0 and system changes from 1 to 0.

From these results we can identify most crucial components according to system structure. These can be seen in Tab. 4.5.

Similarly, second SI was calculated using DPLD $\frac{\partial\phi(j\to h)}{\partial x_i(s\to r)}$, where $h > j \wedge s < r$. This

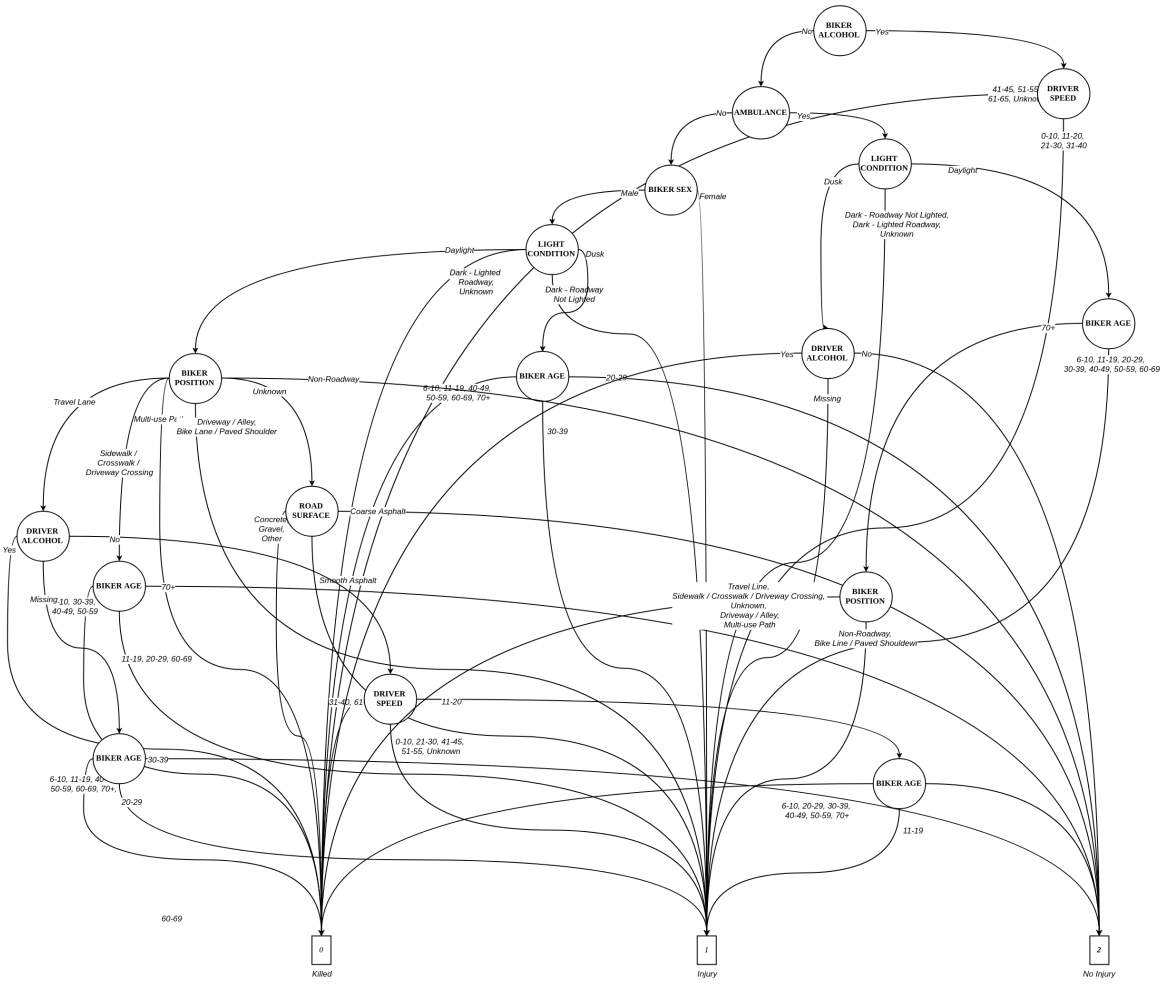

Figure 4.5: MDD of Bike Crashes Dataset

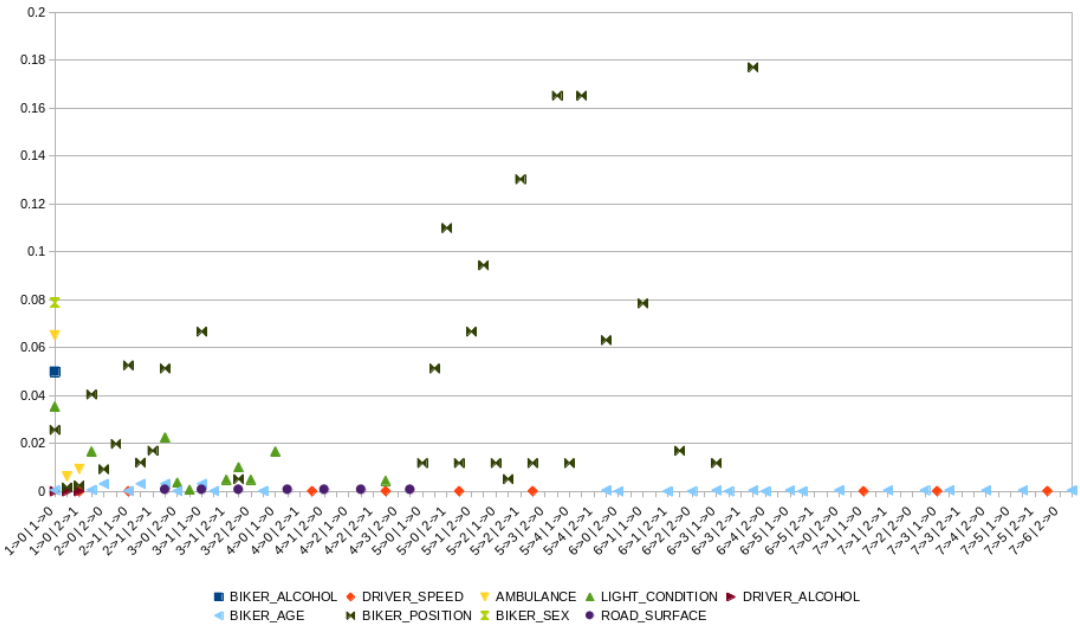

Figure 4.6: Results of $SI^{\downarrow}_{i\downarrow}$ for Bike Crashes Dataset

Table 4.5: Attributes with highest values of $SI_{i\downarrow}^{\downarrow}$ for Bike Crashes Dataset

Component	Component change|System change	Value of $SI_{i\downarrow}^{\downarrow}$
Biker Sex	$1 \to 0 \vert 1 \to 0$	0.0787426
Ambulance	$1 \to 0 \vert 1 \to 0$	0.0651749
Biker Alcohol	$1 \to 0 \vert 1 \to 0$	0.0499926
Light condition	$1 \to 0 \vert 1 \to 0$	0.0354167

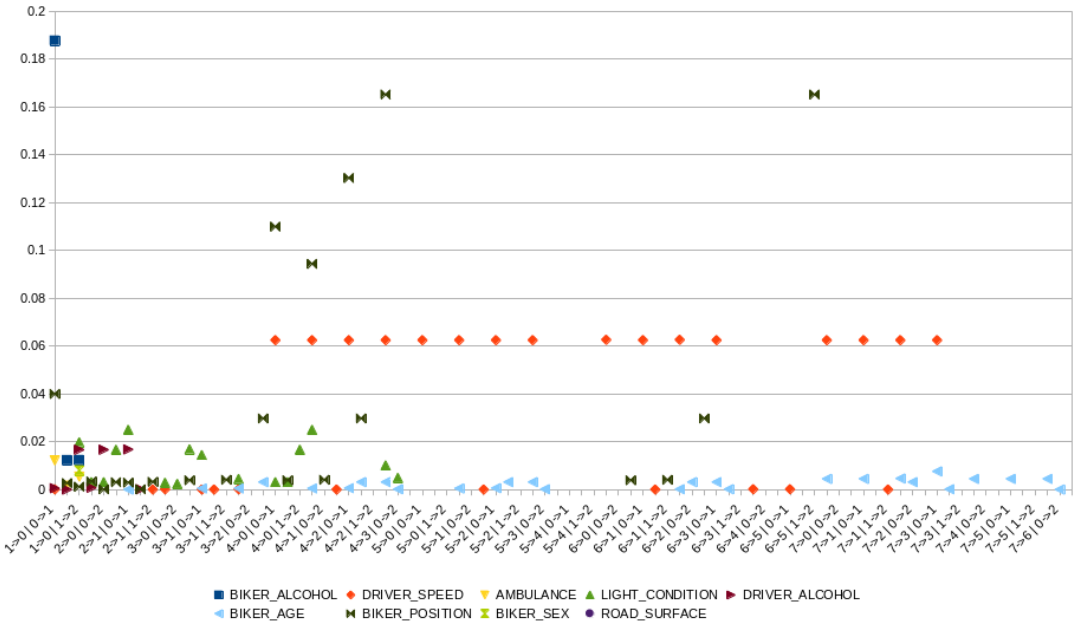

Figure 4.7: Results of SI_i for Bike Crashes Dataset

contains only cases, when system component performance decrease leads to the case system increase its state. Results of this calculation can be seen in Fig. 4.7. The most crucial components together with value of its SI can be seen in Table 4.6.

In the next step, these two SIs was summed for each system component state change and specific system change. This can be expressed using the following form:

$$SI_i^{j,k} = \sum_{r=0}^{m_i} \sum_{s=0}^{m_i} \mathrm{TD}\left(\frac{\partial \phi(j \to k)}{x_i(r \to s)}\right); r \neq s \tag{4.1}$$

Table 4.6: Attributes with highest values of SI_i for Bike Crashes Dataset

Component	Component change|System change	Value of SI_i
Biker Alcohol	$1 \to 0 \vert 0 \to 1$	0.187783
Driver Speed	$6 \to 0 \vert 0 \to 1$,$6 \to 1 \vert 0 \to 1$, $6 \to 2 \vert 0 \to 1$	0.0626488
Light Condition	$2 \to 1 \vert 0 \to 1$, $4 \to 1 \vert 0 \to 1$	0.025

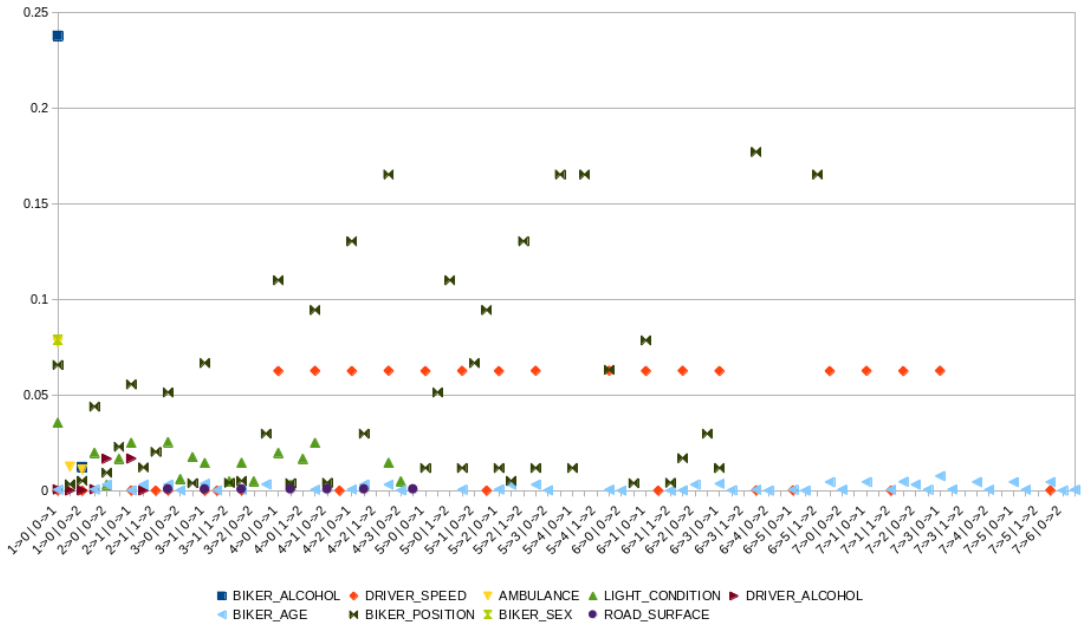

Figure 4.8: Results of SI_i for Bike Crashes Dataset

Table 4.7: Attributes with highest values of $SI_i^{\downarrow}$ for Bike Crashes Dataset

Component	Component change|System change	Value of $SI_{i\downarrow}^{\downarrow}$
Biker Alcohol	$1 \rightarrow 0 \vert 0 \leftrightarrow 1$	0.2377756
Biker Sex	$1 \rightarrow 0 \vert 0 \leftrightarrow 1$	0.0787426
Ambulance	$1 \rightarrow 0 \vert 0 \leftrightarrow 1$	0.0774108
Driver Speed	$4 \rightarrow 3 \vert 0 \leftrightarrow 1$,$5 \rightarrow 3 \vert 0 \leftrightarrow 1$, $7 \rightarrow 3 \vert 0 \leftrightarrow 1$	0.06264881
Light Condition	$1 \rightarrow 0 \vert 0 \leftrightarrow 1$	0.0354167

This includes both - cases when decrease of system component leads to decrease of system state and cases when decrease of system component leads to increase of system state. Results of this case can be seen in Fig. 4.8. The components with highest SI can be seen in Tab. 4.7.

The last mentioned SI is sum of $m_i \times (m_i - 1)$ values for component x_i. In case of heterogeneous system it is necessary to normalize these values as they are results of different number of summed values. This can be performed using the following equation:

$$\mathrm{SIn}_i^{j,k} = \frac{\mathrm{SI}_i^{j,k}}{m_i * (m_i - 1)} \tag{4.2}$$

Results of this normalization can be seen in Tab. 4.9.

According to Tab. 4.9, the most crucial attribute of this system is biker alcohol. This can be interpreted as the fact, that whether biker drank alcohol or not is the

Table 4.8: SI for specific system change for Bike crashes dataset

	System state change from j to k					
Attribute	$2 \to 0$	$2 \to 1$	$1 \to 0$	$1 \to 2$	$0 \to 2$	$0 \to 1$
Biker Alcohol	0	0	0.04999	0.01235	0.01234	0.18778
Driver Speed	0	0.00002	0.00104	0.00007	0.00004	1.00104
Ambulance	0.00625	0.00938	0.06517	0.00541	0.00158	0.01224
Light Condition	0.00858	0.00558	0.10586	0.06997	0.01356	0.08813
Driver Alcohol	0.00006	0.00009	0.00006	0.01667	0.01685	0.01815
Biker Age	0.00843	0.00151	0.00924	0.00313	0.01311	0.04712
Biker Position	0.31134	0.53930	0.64874	0.24834	0.04074	0.54636
Biker Sex	0	0	0.07874	0.00802	0	0
Road Surface	0.00214	0.00071	0.00214	0	0	0

Table 4.9: Normalized SI for specific system change for Bike crashes dataset

	System state change from j to k					
Attribute	$2 \to 0$	$2 \to 1$	$1 \to 0$	$1 \to 2$	$0 \to 2$	$0 \to 1$
Biker Alcohol	0	0	0.025	0.0062	0.0062	0.0939
Driver Speed	0	0	0	0	0	0.0179
Ambulance	0.0031	0.0047	0.0326	0.0027	0.0008	0.0061
Light Condition	0.0004	0.0003	0.0053	0.0035	0.0007	0.0044
Driver Alcohol	0	0	0	0.0028	0.0028	0.003
Biker Age	0.0002	0	0.0002	0.0001	0.0003	0.0011
Biker Position	0.0074	0.0128	0.0154	0.0059	0.001	0.013
Biker Sex	0	0	0.0394	0.004	0	0
Road Surface	0.0001	0	0.0001	0	0	0

```
static void Main(string[] args)
{
    int x = 10;
    int y = 20;
    for (int i = x; i < y; i++)
    {
        if (i % 3 == 0)
            Console.WriteLine("1");
        else
            Console.WriteLine("0");
    }
}
```

Figure 4.9: Example of source code

most significant factor that makes difference between his death or injury. The next crucial attribute is ambulance. The fact ambulance was called to accident has significant impact on result of the accident. The next one is biker sex. The dataset implies, that women has less deaths than men. Significant role in biker survival is caused also by driver speed, where change from $51 - 55$ mps to $31 - 40$ or from $41 - 50$ to $31 - 40$ leads to the fact biker will be alive.

On the other hand, factors as weather or bike direction (whether it is with traffic or facing traffic) has no impact on result of the accident. However it should be noted, that accuracy of these results depends on amount of available data.

4.4 Software reliability evaluation

In this section we will demonstrate whole process of reliability analysis of software system from source code, through syntax tree and reliability model creation to calculation of software characteristics using typical method of reliability analysis - differential calculus.

Let us take the source code from Fig. 4.9. It is a simple program written in C# language in which 2 local variables are declared. Then there is a for loop statement in which it is iterated between these two local variables. In case current value is divisible by 3, value "1" is written into standard output. Otherwise value "0" is written.

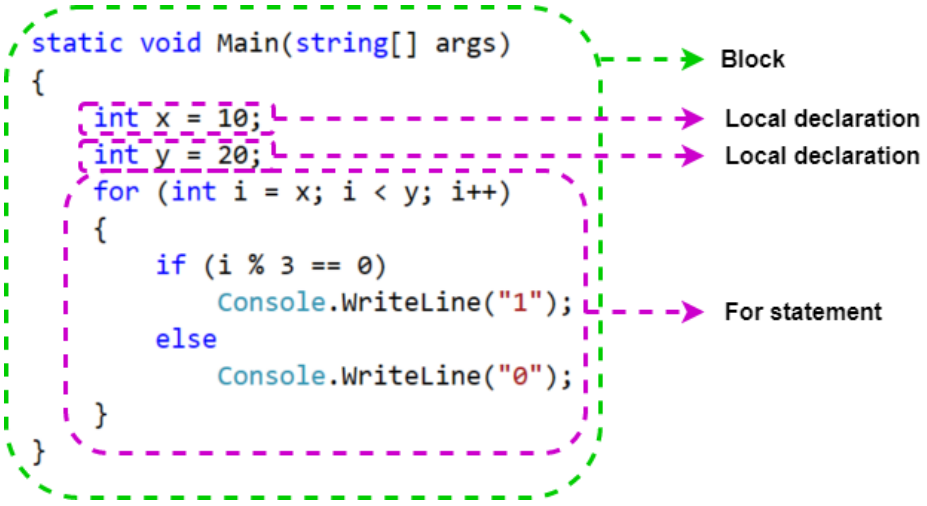

Figure 4.10: Top level of source code description - main function

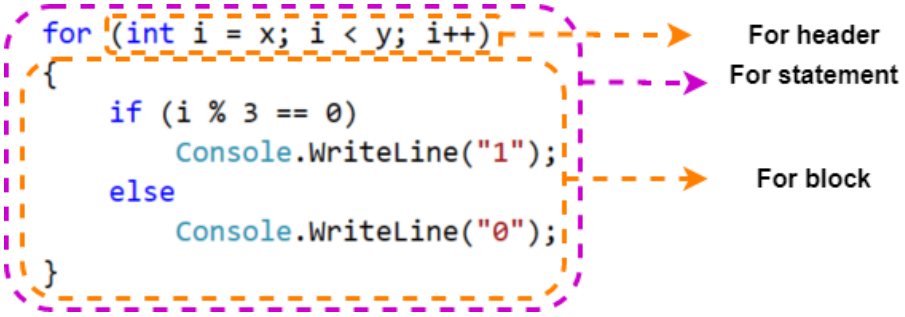

Figure 4.11: Second level of source code description - for loop

4.4.1 Syntax tree creation

This source code can be interpreted in form of syntax tree in the following way. The main block of code consists of three elements, as can be seen in Fig. 4.10. There are two local declaration statements and one for loop statement. In the syntax tree, local declaration can be taken as is and nodes for them are created. The for loop statement can be analysed deeper, as can be seen in Fig. 4.11. For loop consists of the header and the body. The header contains declaration, iteration step and condition to stop the loop. Fig. 4.12 describes the body of the for loop. It contains only one element - if statement. And finally, if statement consists of 3 elements: condition, true block and false block. This can be seen in Fig. 4.13.

All of the mentioned elements are taken into the final syntax tree as can be seen in Fig. 4.14. The elements of the source code from one block are on the same level of the syntax tree. Please note, that it is possible to continue and split some nodes into parts.

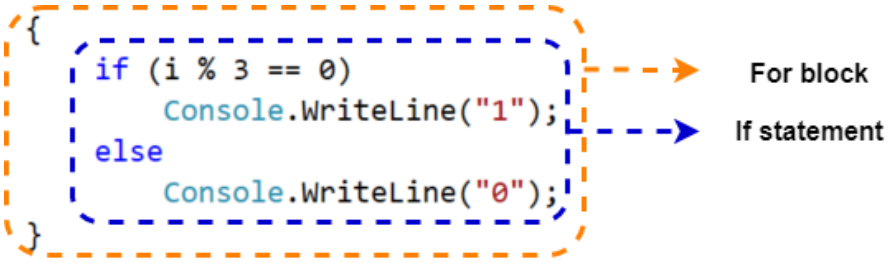

Figure 4.12: Third level of source code description - body of for loop

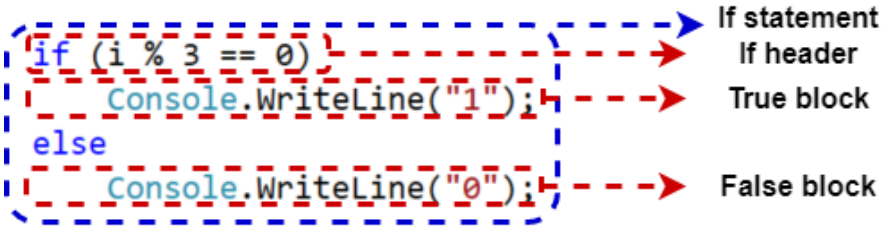

Figure 4.13: The final level of source code description - if statement

For example there is a node for the header element in the for loop statement. In our case, we take it only as one node - variable declaration, but actually this consists of three elements: variable declaration, iteration step and condition to stop. Similarly this split can be applied also to the other elements.

In practise there are several tools, that constructs syntax tree from the source code, for example .NET compiler platform Roslyn for C# language [92] or AST parser for Java language [93]. But a syntax tree created using these tools has too large dimension and is not suitable for the creation of the reliability model unless unnecessary nodes are removed or merged together. Example of the syntax tree generated using Roslyn can be seen in Fig. 3.12. This is representation of the local declaration statement. As we can see, there are 16 nodes, although many of them are not important for our purposes. Therefore this tree has to be processed before using to create a reliability model.

Either way we process our source code into syntax tree, i.e. either we split blocks of code into nodes or we merge unnecessary nodes, the main decision that has to be taken is what depth should be taken in order to create reliability model from syntax tree. There are several options. We can for example use fixed depth and process syntax tree only on the first level. In this case our resulting syntax tree will consist only of 4 nodes - block, 2 local declaration statement and for loop. Other option, we also have used in our work is not to use fixed depth, but to use specific depth for specific elements, i.e.

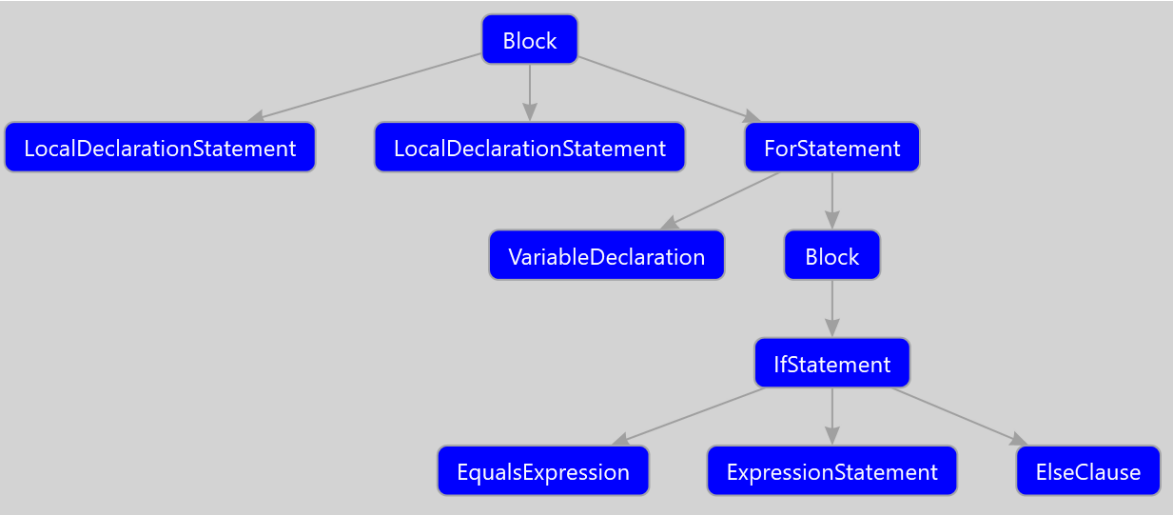

Figure 4.14: Resulting syntax tree corresponding to source code in Fig. 4.9

Table 4.10: Individual source code elements and how to split them

Source code element	**Split**
block	split
declaration statement	no split
for loop	declaration statement & block
if	equal expression & true clause & else clause
equal expression	no split
true clause	no split
else clause	no split

depending on the type of the element, we decide to split or not. Actual decisions for each element can be seen in Table 4.10. In this table, value "no split" means, we stop on the given element and don't continue in current branch. This is used for the true and the else clause of the if statement as they contains only one element of code. In case there will be more than one element, we would decide to continue splitting. Value "split" means we continue in actual branch. In our case we are using it in the block element. That means we take the block as is and all containing elements will be placed in the next level. For other elements of our example, table contains information what element will it be divided into. Please note that this table does not cover all elements of source code, but only the ones used in example. There can be also other elements like other types of loops, function calls, more complex conditions etc.

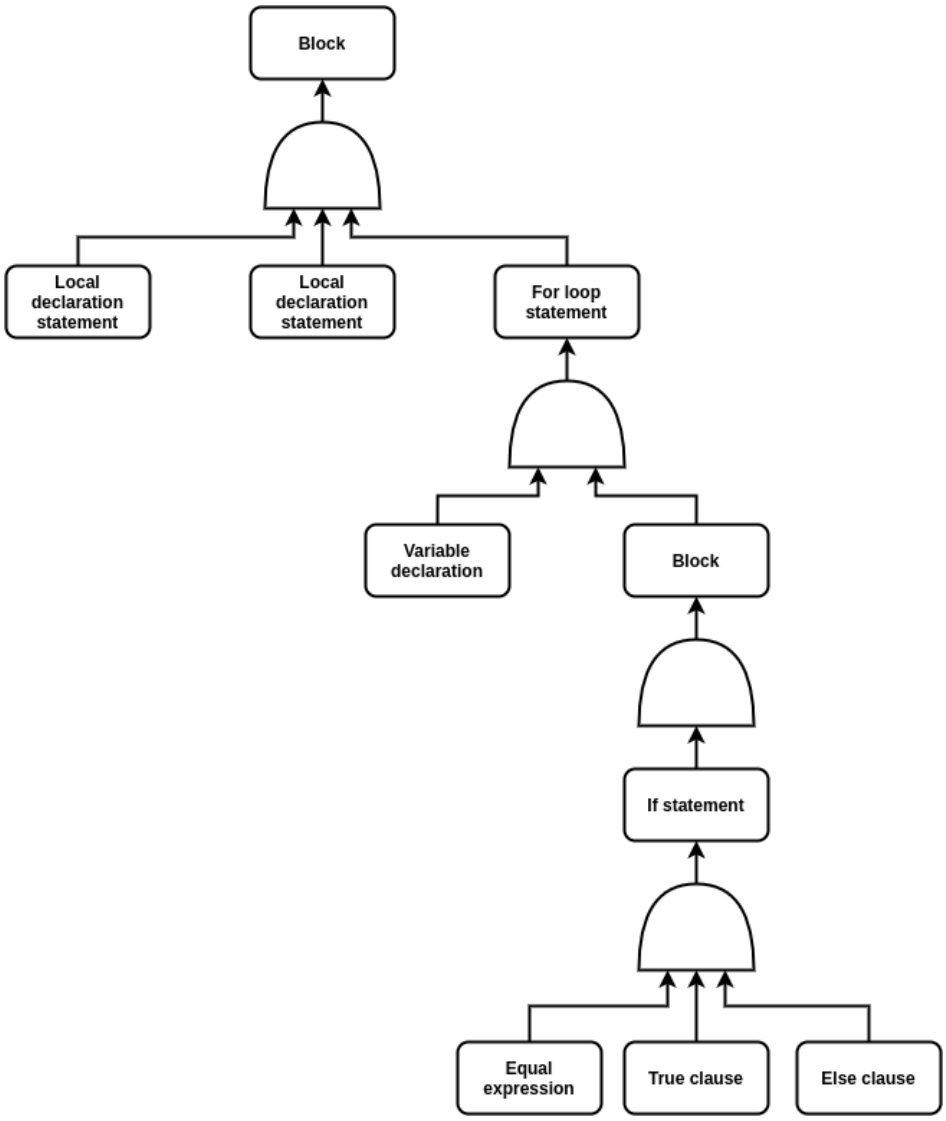

Figure 4.15: Example of fault tree created using syntax tree from Fig. 4.14

4.4.2 The creation of the reliability model from syntax tree

Created syntax tree will be in the next step used to create a reliability model. We have decided to use fault tree. Each node of syntax tree will match with one event in fault tree. This event describes situation where the given elements fail. For example a failure of the local declaration statement means there is an unexpected value assigned to this variable or the local declaration statement is defined with unexpected type as it was expected etc. A failure of the for loop means, that at least one of its child elements fails, e.g. wrong iteration step will be set or it will iterate between incorrect boundaries, etc.

Another decision that has to be taken is how to connect individual events together.

Table 4.11: Probabilities of states for each node in Fig. 4.15

Node name	**Component label**	p_i	q_i
Local declaration statement	x_1	0.98	0.02
Local declaration statement	x_2	0.98	0.02
Variable declaration	x_3	0.90	0.10
Equal expression	x_4	0.95	0.05
True clause	x_5	0.96	0.04
Else clause	x_6	0.96	0.04

The main problem is that failure of these events is not independent. In our example failure of the first local declaration statement will lead to failure of the whole for loop as the for loop iterates between these two local variables. And furthermore a variable declared in the for loop statement is then used in the if statement. So the mentioned failure of the first local declaration statement can lead to the failure of the whole software. Therefore we decide to connect these events using AND gates. This will ensure that this model will not work incorrectly. But using AND gates between all events can lead to an inaccuracy of this model as not every failure lead to the failure of the whole software. Let us consider situation, where a value assigned to the local declared variable "x" will be 1 instead of 10 and a value assigned to the variable "y" will be 11 instead of 20. The number of iterations in the for loop statement will be the same (10 iterations) and even the output of the software will match with the expected one. In this case a mutual failure of both local declaration statements lead to the correct behavior of the software. Mentioned inaccuracy is in this model neglected.

The resulting reliability model created using syntax tree from Fig. 4.14 can be seen in Fig. 4.15. Structure of this model corresponds to the syntax tree this model was constructed from and individual events are connected via AND logic gates together.

Created fault tree can be easily transformed into structure function in the form of the truth table. This allows us to use the logic differential calculus and using this tool perform topological analysis and calculate also other characteristics of the system such as system reliability. Calculation of some of these characteristics is possible only with information about probabilities of individual system component states. These probabilities can be seen in Table 4.11. Probabilities listed in the mentioned table are only example values. Real values can be obtained using statistical information or

Table 4.12: Structure function in form of Truth Table

x_1	x_2	x_3	x_4	x_5	x_6	$\phi(\boldsymbol{x})$	x_1	x_2	x_3	x_4	x_5	x_6	$\phi(\boldsymbol{x})$
0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	1	0	1	0	0	0	0	1	0
0	0	0	0	1	0	0	1	0	0	0	1	0	0
0	0	0	0	1	1	0	1	0	0	0	1	1	0
0	0	0	1	0	0	0	1	0	0	1	0	0	0
0	0	0	1	0	1	0	1	0	0	1	0	1	0
0	0	0	1	1	0	0	1	0	0	1	1	0	0
0	0	0	1	1	1	0	1	0	0	1	1	1	0
0	0	1	0	0	0	0	1	0	1	0	0	0	0
0	0	1	0	0	1	0	1	0	1	0	0	1	0
0	0	1	0	1	0	0	1	0	1	0	1	0	0
0	0	1	0	1	1	0	1	0	1	0	1	1	0
0	0	1	1	0	0	0	1	0	1	1	0	0	0
0	0	1	1	0	1	0	1	0	1	1	0	1	0
0	0	1	1	1	0	0	1	0	1	1	1	0	0
0	0	1	1	1	1	0	1	0	1	1	1	1	0
0	1	0	0	0	0	0	1	1	0	0	0	0	0
0	1	0	0	0	1	0	1	1	0	0	0	1	0
0	1	0	0	1	0	0	1	1	0	0	1	0	0
0	1	0	0	1	1	0	1	1	0	0	1	1	0
0	1	0	1	0	0	0	1	1	0	1	0	0	0
0	1	0	1	0	1	0	1	1	0	1	0	1	0
0	1	0	1	1	0	0	1	1	0	1	1	0	0
0	1	0	1	1	1	0	1	1	0	1	1	1	0
0	1	1	0	0	0	0	1	1	1	0	0	0	0
0	1	1	0	0	1	0	1	1	1	0	0	1	0
0	1	1	0	1	0	0	1	1	1	0	1	0	0
0	1	1	0	1	1	0	1	1	1	0	1	1	0
0	1	1	1	0	0	0	1	1	1	1	0	0	0
0	1	1	1	0	1	0	1	1	1	1	0	1	0
0	1	1	1	1	0	0	1	1	1	1	1	0	0
0	1	1	1	1	1	0	1	1	1	1	1	1	1

estimations of experts. Please note, that only leaf nodes of the fault tree is required as probabilities of non-leaf nodes depend only on these leaf nodes. This table also contains information about mapping node names into variable names. This makes work with these nodes simpler. Structure function in form of the truth table can be seen in Table 4.12.

4.4.3 Quantitative analysis

These information allow us to perform quantitative analysis according to section 1.4. As we mentioned, reliability is a probability system works without failure. From structure function represented in the form of truth table as can be seen in Table 4.12, there is only one row, where system is working and that is the case, when all of its components works. Therefore the reliability of this system can be calculated as following:

$$\begin{aligned} R &= p_1 \times p_2 \times p_3 \times p_4 \times p_5 \times p_6 \\ R &= 0.98 \times 0.98 \times 0.90 \times 0.95 \times 0.96 \times 0.96 \\ R &= 0.7567 \end{aligned} \tag{4.3}$$

Similarly unreliability U is defined as a probability system will fails. Using the truth table we can see, there are 63 cases, when system will fails. To calculate the unreliability we can either sum the probabilities of these cases or to use calculated reliability:

$$\begin{aligned} U &= 1 - R \\ U &= 1 - 0.7567 \\ U &= 0.2433 \end{aligned} \tag{4.4}$$

Structure function can be also used to calculate importance measures for individual components. Firstly we will demonstrate the calculation of the structure importance for component x_1. Using direct partial Boolean derivative and information from Table 4.12, we can see, that there is only one case, when the failure of this component results in the failure of the whole system and it is in the case, when all other components are working. The number of all possible DPBDs for the component x_1 are 32. The resulting SI for component x_1 can be therefore calculated as following:

$$\mathrm{SI}_1 = \mathrm{TD}\left(\frac{\partial\phi(1\to 0)}{\partial x_1(1\to 0)}\right) = \frac{1}{32} \approx 0.03125 \tag{4.5}$$

Structure importance can be similarly calculated also for other components. The results of this calculation can be seen in Table 4.13. As we can see, the resulting SI is equal for all components. That means, that from topological point of view every component, or in our case every element of the source code, has the same impact on system reliability, or in our case the same impact for the software to work correctly.

Table 4.13: Results of SI_i calculation

Component	SI_i
x_1	0.03125
x_2	0.03125
x_3	0.03125
x_4	0.03125
x_5	0.03125
x_6	0.03125

The usage of the information about probabilities of the individual component states presented in Table 4.11 allow us to calculate also different importance measure, Birnbaum's importance. A calculation of BI for the component x_1 is similar to the calculation of SI. Using DPBD we find cases, where failure of the component x_1 results in the failure of the system. Then we will calculate the probability of the occurrence of this cases. There is only one case for the component x_1 where DPBD takes value 1 - when all components are working. The Birnbaum's importance for this component can be then calculated as following:

$$\begin{aligned} \mathrm{BI}_1 &= p_2 \times p_3 \times p_4 \times p_5 \times p_6 \\ \mathrm{BI}_1 &= 0.98 \times 0.90 \times 0.95 \times 0.96 \times 0.96 \\ \mathrm{BI}_1 &\approx 0.7722 \end{aligned} \tag{4.6}$$

The Birnbaum's importance calculus for the other components will be similar to the calculus of the first one. The results of this calculation can be seen in Table 4.14. The results show, that component x_3 is slightly more important for our system that the other components. Component x_3 represents a variable declaration statement in

the for loop statement and it is caused by the fact, there is a higher probability of its failure as it is for the other elements of the source code. The second most important component according to results is the component x_4, what represents condition in the if statement. Then there is components x_5 and x_6 and the least important according to Birnbaum's importance are components x_1 and x_2. The results show us, that we need to put a bigger effort and be more careful when writing declarations of the for loop statement and conditions compared to declaring local variables.

Table 4.14: Results of BI_i calculation

Component	BI_i
x_1	0.7722
x_2	0.7722
x_3	0.8408
x_4	0.7966
x_5	0.7883
x_6	0.7883

Conclusion

The main focus of our work was on reliability analysis of non-coherent systems. Within this process, we set several goals, specifically: 1. The investigation of methods for reliability analysis of multi-state non-coherent systems. 2. The investigation of methods for the reduction of computational complexity of structure function. 3. The investigation of methods for construction of structure function based on incompletely specified data.

The first goal is described in sections 2.1 and 2.2. This goal is achieved by proposing new approach for non-coherent MSS analysis based on mathematical methods of MVL. Similarly to the studies [6, 41], the investigated system is represented by structure function. According to [6], the structure function can be interpreted as an MVL function, and this allows us to use MVL mathematical methods for analysis and evaluation of MSS. In particular in this thesis the analysis of critical system component states are considered. The critical component states are defined for every system component. For these states, the specified change of component state results in the change of the system performance level. According to previous studies in [6], the critical system state can be computed by methods of Logical Differential Calculus, in particular, Direct Partial Logical Derivatives (DPLD). In study [41] the derivatives named Integrated Direct Partial Logical Derivatives (IDPLD) have been developed for calculation of Importance Measures of coherent MSS. Other modification of DPLD in [42] allows definition of minimal cut/path sets of coherent MSS. In this thesis, we develop DPLD-based approach for the computation of critical component states of non-coherent system. The development of this approach is based on analysis of definition of non-coherent MSS proposed in investigation [33, 34, 35, 36, 37].

The application of this method is presented in section 4.1 where example of anesthesia examination is considered. On this system, critical state vectors are calculated using proposed method. This allows us to investigate the system behaviour from reliability point of view more deeply.

The second goal is described in section 2.3 and is achieved using MDD representation of structure function. Thanks to the fact that this representation is compact it is possible to describe also systems with large dimension of structure function. This

representation can be also easily used to perform quantitative analysis, both - topological and probabilistic. This is due to the fact that MDD is an orthogonal representation of SF [64]. Within this goal, we also decide to represent MDD using vector of neighbours as is described in section 2.3.1.

The application of this method is present in section 4.2. In this section, case study for patient with hepatitis survival chance is described. Using data obtained from [89], MDD is constructed and this diagram is used to calculate structure and Birnbaum's importance.

The last specified goal was to investigate methods for construction of incompletely specified structure function. This problem is described in section 2.4 and partially in 2.3 and is achieved using methods of data mining to analyse system and construct decision tree. Decision tree can be reduced into MDD. Algorithm of this reduction is described in section 2.3.2. As MDD is a form of structure function, it can be then easily used to perform reliability analysis using DPLD and other methods for SF analysis.

The application of this method is present in sections 4.2 and 4.3. Both of these case studies describe systems with incompletely specified data and, using the mentioned method, MDD are constructed. These case studies also demonstrate the quantitative analysis.

The last part of this thesis described in chapter 3 is focused on software reliability. The main idea of our proposed method is to use a source code and syntax tree to create software reliability model. This method consists of 2 essential steps. In the first step, source code is used to construct abstract syntax tree (AST) and, in the second step, its reliability model is constructed using the created AST. Specifically, fault tree is created. This can be easily transformed into structure function. This allows us to analyse software using typical methods for reliability analysis. The main disadvantage of this method is the large dimension of structure function even for simple software. However natural breakdown of source code into functions and procedures allows us to easily use methods for reduction of this complexity.

The application of this method is present in section 4.4 where example of the whole process of analysis of source code and the creation of structure function is demonstrated. This structure function is used to calculate basic characteristics, such as reliability.

Our future work will be focused on investigation of non-coherent systems more

deeply. We will consider the usage of proposed DPLD and critical state vectors to perform also quantitative analysis based on importance measures. In the next part of our future research, we will investigate a possibility to further reduce computational complexity of performing reliability analysis for systems with large structure function, for example, to use methods as modular decomposition [94, 44, 95] in MDD. Finally, we plan to deal with improvement of our proposed method for software analysis, as there are mainly 2 problems with current model. The first one is its large dimension. As can be seen in section 4.4, even quite simple code results in structure function with large dimension. There are several ways to solve this problem. The first one is by reducing the size of the syntax tree, necessary to create this model, by merging or removing nodes until resulting syntax tree has more suitable size. This way, created reliability model will also have reduced size, therefore, the whole analysis will be simpler. The other solution is to use method of modular decomposition [94, 44, 95]. Using this method, reliability model can be divided into several modules, and each module can be analysed separately. This will reduce time required to perform reliability calculation of the whole system with usage of methods like parallelism. The next problem is the usage of the AND gates what can leads to inaccuracy in the created model. In order to solve this problem, the future research is required. This will help us to determine how each event should be connected to other events in order to increase accuracy of the created model.

Resume

Spoľahlivosť je v súčasnosti dôležitá charakteritstika akéhokoľvek systému. Analýza spoľahlivosti je komplexný proces, ktorého prvým krokom je tvorba matematickej reprezentácie skúmaného systému [4, 5, 6]. Matematický model systému je tvorený s ohľadom na špecifiká analýzy systému a vlastností daného systému. V závislosti od počtu úrovní výkonností môžu byť matematické modely rozdelené na dve skupiny [4]: dvojstavové systémy a viacstavové systémy.

Oba tieto typy môžu byť v závislosti od vplyvu degradácie komponentu na funkčnosť systému koherentné a nekoherentné [7, 8, 9]. Degradácia alebo zlyhanie komponentu koherentného systému nemôže viesť k zvýšeniu úrovne funkčnosti systému a všetky komponenty koherentného systému sú relevantné pre fungovanie systému [10, 8]. V spoľahlivostnom inžinierstve sú koherentné systémy v porovnaní s nekoherentnými intenzívne študované.

Analýza spoľahlivosti nekoherentných systémov vyžaduje špeciálne metódy. Je to spôsobené nemonotónnym vplyvom zlyhania komponentu na funkčnosť systému. Najčastejší prístup k nekoherentným dvojstavovým systémom je založený na analýze prostých implikantov. Tieto boli použité pri vývoji metód na ohodnotenie nekoherentných systémov v publikáciách [27, 12, 28, 29]. Primárne implikanty boli navrhnuté v [29] ako analógia k množinám minimálnych rezov na definovanie minimálnych kombinácií zlyhaní komponentov, ktoré spôsobia zlyhanie systému. Stromy poruchových stavov a množiny minimálnych rezov sú efektívnym nástrojom na analýzu koherentných systémov, ale nie sú ekvivalentné s primárnymi implikantami a neberú do úvahy nemonotónny vplyv zlyhania komponentu nekoherentného systému.

Na rozdiel od dvojstavových systémov, nekoherentné viacstavové systémy nie sú intenzívne študované. Existujú štúdie takýchto systémov, kde sa zvažujú niektoré teoretické aspekty [33, 34, 35, 36], ale metódy analýzy neboli ešte rozvinuté. Je to spôsobené ťažkosťami a nepresnosťami v teoretickej interpretácii nekoherentných viacstavových systémov a taktiež výpočtovej zložitosti analýzy takýchto systémov. Koncept koherencie a nekoherencie dvojstavových systémov bol zovšeobecnený pre viacstavové systémy v publikáciách [34, 35, 36]. Výpočet charakteristík a ukazovateľov nekoherentných viacstavových systémov bol prezentovaný v publikáciách [38, 39, 40]. Bossche

v publikáciách [38, 39] navrhol frekvenčné ohodnotenie spoľahlivosti nekoherentných viacstavových systémov založené na stromoch poruchových stavov s aplikáciou prostých implikantov. Autor zovšeobecnil koncept prostých implikantov z nekoherentných dvojstavových systémov a upravil prístup dvojhodnotovej logiky pre analýzu viacstavových systémov. Metódy viachodnotovej logiky sa používajú na definovanie ukazovateľov dôležitosti nekoherentných viacstavových systémov v publikácií [40]. Avšak autori navrhli výpočet ukazovateľov dôležitosti intuitívne a neanalyzovali teoretický základ a definície nekoherentných viacstavových systémov.

V rámci práce sme sa preto rozhodli stanoviť si nasledujúce ciele:

1. Preskúmať metódy analýzy spoľahlivosti viacstavových nekoherentných systémov.
2. Preskúmať metódy redukcie výpočtovej zložitosti analýzy štruktúrnej funkcie.
3. Preskúmať metódy tvorby štruktúrnej funkcie založené na neúplne definovaných dátach.

Okrem týchto cieľov sme sa rozhodli rozšíriť náš výskum taktiež na spoľahlivosť softvéru a preskúmať možnosti reprezentácie softvéru vo forme štruktúrnej funkcie.

V rámci práce navrhujeme nový prístup pre analýzu viacstavových nekoherentných systémov založených na matematických metódach viachodnotovej logiky. Tento prístup vychádza zo štúdií koherentných viacstavových systémov, ktoré boli prezentované v publikáciách [6, 41]. Podobne ako v daných štúdiách, skúmaný systém je reprezentovaný pomocou štruktúrnej funkcie, ktorá mapuje všetky možné stavy komponentov na úroveň výkonnosti systému. Podľa [6] štruktúrna funkcia môže byť interpretovaná ako viachodnotová logická funkcia a tento fakt nám umožňuje použitie metód viachodnotovej logiky pri analýze a ohodnotení viacstavového systému. V našej práci navrhujeme prístup pomocou orientovaných parciálnych logických derivácií na výpočet kritických stavov komponentov systému pre nekoherentné systémy. Vývoj tohto prístupu je založený na analýze definícií nekoherentných viacstavových systémov navrhnutých a preskúmaných v publikáciách [33, 34, 35, 36, 37].

Základné definície a princípy v spoľahlivostnom inžinierstve

Ohodnotenie spoľahlivosti systému je komplexný proces, ktorého výsledkom sú informácie o systéme a jeho charakteristiky z pohľadu spoľahlivosti ako spoľahlivosť,

ukazovetele dôležitosti, ktirické stavy atď. Celá analýza je prispôsobená v závislosti od charakteristík, ktoré chceme získať. Toto prispôsobenie pozostáva z výberu matematického modelu analyzovaného systému. Existujú viaceré matematické modely používané pri analýze spoľahlivosti a pri jeho výbere sa berú do úvahy 2 kritériá:

- počet stavov systému;
- matematický prístup, ktorý určuje algoritmy a metódy použité na ohodnotenie systému.

Počet stavov systému a jeho komponentov závisí od požiadaviek na analýzu. Podľa počtu stavov systému delíme systémy na dvojstavové a viacstavové.

Dvojstavový systém je matematická reprezentácia systému s dvomi úrovňami výkonnosti, t.j. systém buď funguje alebo zlyhal. Táto matematická reprezentácia sa používa, ak sú systémy prirodzene binárne [21, 43], alebo ak analyzujeme následky zlyhaní [5].

Viacstavový systém nám umožňuje definovať viac ako dve úrovne výkonnosti systému a popísať postupnú degradáciu výkonnosti systému z úplne funkčného na úplne nefunkčný [44, 4, 45]. Viacstavový systém nám umožňuje analyzovať systém detailnejšie, ale výpočtová zložitosť analýzy sa zvyšuje a sú potrebné špecifické metódy pre kvantitatívnu analýzu systému reprezentovanú takýmto modelom.

Štruktúrna funkcia

V závislosti od matematického základu použitého v analýze systémov existuje viacero modelov. Jedným z nich je štruktúrna funkcia, ktorá vyjadruje závislosť zmeny úrovne výkonnosti systému od zmeny úrovne výkonnosti jeho komponentov. Majme viacstavový systém s n komponentami. Systém má m úrovní výkonností. i-ty komponent takéhoto systému má m_i stavov. Úrovne výkonnosti systému a jeho komponentov nadobúdajú hodnoty od 0 na reprezentáciu zlyhania systému po $m-1$ a m_i-1 respektíve na reprezentáciu perfektnej funkčnosti. Závislosť medzi úrovňou výkonnosti systému a jeho komponentov môže byť vyjadrená použitím štruktúrnej funkcie vo forme [47, 6]:

$$\phi(x_1, x_2, ..., x_n) = \phi(\boldsymbol{x}) : \{0, 1, ..., m_1 - 1\} \times \{0, 1, ..., m_2 - 1\} \times ... \\ \times \{0, 1, ..., m_n - 1\} \rightarrow \{0, 1, ..., m - 1\},$$

kde x_i je stav i-teho komponentu systému, $i \in \{1, 2, ..., n\}$, $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ je stavový vektor komponentov systému.

Pravdepodobnosť, že i-ty komponent je v stave j môže byť definovaná nasledovne:

$$p_{i,j} = \Pr\{x_i = j\}$$
$$q_i = p_{i,0} = \Pr\{x_i = 0\}.$$

Reprezentácie štruktúrnej funkcie

Štruktúrna funkcia môže byť reprezentovaná viacerými spôsobmi:

- tabuľka pravdivostných hodnôt,
- blokový diagram spoľahlivosti,
- dvoj a viachodnotový rozhodovací diagram,
- množina minimálnych rezov a vektor minimálnych rezov.

Mnoho reálnych systémov pozostáva z veľkého množstva komponentov, čo vedie k veľkému rozmeru štruktúrnej funkcie. Preto je potrebné používať efektívne reprezentácie štruktúrnej funkcie, napríklad viachodnotový rozhodovací diagram.

Logický diferenciálny počet

Interpretácia štruktúrnej funkcie ako viachodnotovej logickej funkcie nám umožňije použitie matematických metód viachodnotovej logiky na analýzu a ohodnotenie viacstavových systémov. Autori publikácií [50, 51] navrhli použitie orientovaných parciálnych logických derivácií na ohodnotenie dôležitosti komponentov systému. V publikácii [41] boli predstavené uazovatele dôležitosti definované pomocou týchto derivácií. Avšak tento výskum sa zameriaval na koherentné viacstavové systémy. V tejto práci je navrhnuté ich použitie pre ohodnotenie nekoherentných systémov.

Orientované parciálne logické derivácie boli definované pre binárne funkcie a neskôr zovšeobecnené pre viachodnotové logické funkcie v publikácii [52]. Tieto derivácie vo viachodnotovej logike s ohľadom na premennú x_i indikujú zmenu funkcie z j na h v závislosti od zmeny premennej z s na r. V oblasti spoľahlivosti, orientovaná parciálna

logická derivácia štruktúrnej funkcie umožňuje definovať zmenu úrovne výkonností systému z j na h v závislosti od zmeny stavu i-teho komponentu z s na r [50]:

$$\frac{\partial\phi(j\to h)}{\partial x_i(s\to r)}=\begin{cases}1, & \text{if} \quad \phi(s,\boldsymbol{x})=j \quad \text{and} \quad \phi(r,\boldsymbol{x})=h\\ 0, & \text{otherwise}\end{cases}$$

pre $s,r\in\{0,1,...,m_i-1\}, s\neq r, j,h\in\{0,1,...,m-1\}, j\neq h$.

Všetky možné zmeny úrovne výkonnosti systému môžu byť definované vzľadom na orientovanú parciálnu logickú deriváciu (1.4) pre špecifickú zmenu stavu komponentu z s na r, ak je táto derivácia vypočítaná pre všetky možné zmeny úrovne výkonnosti systému definované parametrami j a h. Toto vyžaduje výpočet veľkej množiny derivácií. V publikácii [41] boli definované nové typy derivácií pre analýzu množiny úrovní výkonnosti systému v závislosti od špecifickej zmeny komponentu systému. Tieto derivácie sa nazývajú integrované orientované parciálne logické derivácie. V publikácii [41] boli predstavené 3 typy týchto derivácií definované pre zmenu i-teho komponentu z s na $s-1$ pre analýzu degradácie systému. Integrované orientované logické derivácie pre zmenu stavu komponentu z s na $s+1$ nám umožňujú analyzovať zvýšenie úrovne funkčnosti systému [41] a derivácie pre zmenu i-teho komponentu z s na r môžu byť definované podobným spôsobom.

Integrované orientované parciálne logické derivácie I. typu nám umožňujú identifikovať stavové vektory, kedy degradácia i-teho komponentu zo stavu s na r spôsobí degradáciu úrovne výkonnosti systému z j na akýkoľvek stav $h<j$:

$$\frac{\partial\phi(j\downarrow)}{\partial x_i(s\to r)}=\bigvee_{h=0}^{j-1}\frac{\partial\phi(j\to h)}{\partial x_i(s\to r)}=\begin{cases}1, & \text{if} \quad \phi(s,\boldsymbol{x})=j \quad \text{and} \quad \phi(r,\boldsymbol{x})<j\\ 0, & \text{otherwise}\end{cases}$$

pre $j\in\{1,2,...,m-1\}$.

Ďalšia verzia I. typu tejto derivácie pre koherentné systémy nám umožňuje identifikovať stavové vektory, kde degradácia i-teho komponentu zo stavu s na r vedie k degradácii úrovne výkonnosti systému z akéhokoľvek stavu $h>j$ na j:

$$\frac{\partial\phi(\downarrow j)}{\partial x_i(s\to r)}=\bigvee_{h=j+1}^{m-1}\frac{\partial\phi(h\to j)}{\partial x_i(s\to r)}=\begin{cases}1, & \text{if} \quad \phi(s,\boldsymbol{x})>j \quad \text{and} \quad \phi(r,\boldsymbol{x})=j\\ 0, & \text{otherwise}\end{cases} \tag{4.7}$$

pre $j\in\{1,2,...,m-1\}$.

Analýza všetký možných vplyvov zmeny stavu i-teho komponentu z s na r sa dá vypočítať pomocou derivácie II. typu. Táto derivácia je definovaná ako spojenie derivácií I. typu:

$$\frac{\partial\phi(\downarrow)}{\partial x_i(s\to r)} = \bigvee_{j=1}^{m-1}\frac{\partial\phi(j\downarrow)}{\partial x_i(s\to r)} = \bigvee_{j=0}^{m-2}\frac{\partial\phi(\downarrow j)}{\partial x_i(s\to r)} = \begin{cases} 1, & \text{if} \quad \phi(s_i,\boldsymbol{x}) > \phi(r_i,\boldsymbol{x}) \\ 0, & \text{otherwise} \end{cases} \tag{4.8}$$

III. typ integrovaných orientovaných parciálnych logických derivácií pre zmenu úrovne výkonnosti systému j identifikuje všetky stavové vektory, pre ktoré zmena stavu komponentu i z s na r spôsobí zmenu úrovne výkonnosti systému z hodnoty väčšej ako alebo rovnej j na úroveň menšiu ako j:

$$\frac{\partial\phi(h_{\geq j}\to h_{<j})}{\partial x_i(s\to r)} = \bigvee_{h_u=j}^{m-1}\bigvee_{h_d=0}^{j-1}\frac{\partial\phi(h_u\to h_d)}{\partial x_i(s\to r)} = \begin{cases} 1, & \text{if} \quad \phi(s_i,\boldsymbol{x}) \geq j \\ & \text{and} \quad \phi(r_i,\boldsymbol{x}) < j \\ 0, & \text{otherwise} \end{cases} \tag{4.9}$$

kde $j \in \{1,2,...,m-1\}$ a notácia $h_{\geq j}(h_{<j})$ znamená, že všetky stavy systému väčšie ako alebo rovné (menšie ako) j sa berú do úvahy.

Spomenuté integrované orientované logické parciálne derivácie môžu byť použité na určenie kritických stavov systému.

Pri zovšeobecnení týchto derivácií pre analýzu nekoherentných systémov musíme vziať do úvahy možnosť degradácie a zvýšenia špecifikovanej úrovne výkonnosti systému j v závislosti od zmeny stavu i-teho komponentu systému z s na r [12, 53].

Koherentné a nekoherentné systémy

V závislosti od správania sa môžeme systémy rozdeliť na dve skupiny - **koherentné** and **nekoherentné** [11, 34, 7]. Dominantnou skupinou systémov sú koherentné, pre ktoré zlyhanie akéhokoľvek komponentu môže spôsobiť zlyhanie systému. Nekoherentné systémy sú preskúmané najmä pre dvojstavové systémy. Nalsedujúce kritéria musia platiť, aby bol dvojstavový systém koherentný [34, 7, 12]:

1. štruktúrna funkcia je monotónne neklesajúca: $\phi(1_i,\boldsymbol{x}) \geq \phi(0_i,\boldsymbol{x})$ pre akýkoľvek komponent i, kde $\phi(1_i,\boldsymbol{x}) = \phi(x_1,...,x_{i-1},1,x_{i+1},...,x_n)$ a $\phi(0_i,\boldsymbol{x}) = \phi(x_1,...,x_{i-1},0,x_{i+1},...,x_n)$;

2. Každý komponent systému je pre systém relevantný: $\phi(1_i,\boldsymbol{x}) \neq \phi(0_i,\boldsymbol{x})$ pre niektoré $\boldsymbol{x}$.

Väčšina technických systémov je koherentná. Ak aspoň jedna z podmienok neplatí, potom je systém nekoherentný [7, 12]. Príklad nekoherentných systémov môže byť systém dodávky plynu [12, 7], logické obvody a siete [21], systémy s ľudským faktorom [54] alebo softvérové komponenty [30]. Jeden z relevantných problémov pri nekoherentných dvojstavových systémov je výpočet analýzy dôležitosti [7, 12, 19]. Autori publikácie [51] navrhujú použitie logického diferenciálneho počtu pre výpočet ukazovateľov dôležiosti dvojstavových systémov. Ako bolo ukázané v publikáciách [53, 21], tento prístup môže byť použitý aj pre nekoherentné dvojstavové systémy. V rovnakom čase bol zovšeobecnený prístup logického diferenciálneho počtu pre analýzu viachodnotových logických funkcií a stal sa efektívnym nástrojom na analýzu viacstavových systémov. Metódy analýzy koherentných viacstavových systémov založené na logickom diferenciálnom počte boli prezentované napríklad v publikáciách [6, 41].

Kvantitatívna analýza

Štruktúrna funkcia môže byť použitá na analýzu systému z hľadiska spoľahlivosti. Toto zahŕňa výpočet charakteristík systému ako sú spoľahlivosť a dostupnosť. Okrem týchto charakteristí je možné vypočítať charakteristiky jednotlivých komponentov ako sú ukazovatele dôležitosti.

Spoľahlivosť je základná charakteristika systému a môže byť definovaná ako pravdepodobnosť, že systém funguje.

$$R = \Pr\{\phi(\boldsymbol{x}) = 1\}.$$

V prípade viacstavových systémov je možné vyjadriť taktiež pravdepodobnosť ostatných úrovní funkčnosti systému. V tom prípade hovoríme o dostupnosti.

Dostupnosť a nedostupnosť môžu byť definované ako pravdepodobnosť, že systém je aspoň v stave j. Podobne nedostupnosť je pravdepodobnosť, že systém funguje v stave nižšom ako j.

$$A^{\geq j}(\boldsymbol{p}) = Pr\{\phi(\boldsymbol{x}) \geq j\},$$
$$U^{\geq j}(\boldsymbol{p}) = Pr\{\phi(\boldsymbol{x}) < j\}$$

kde $\boldsymbol{p}$ je vektor pravdepodobností stavov komponentov [55, 45, 44].

V mnohých prípadoch je potrebné poznať nielen charakteristiky systému, ale taktiež ako je úroveň funkčnosti systému ovplyvnená jednotlivými komponentami. Na to nám slúžia ukazovatele dôležitosti.

Štruktúrna dôležitosť $\mathbf{SI}_i$

Štruktúrna dôležitosť vyjadruje vplyv komponentu systému na úroveň funkčnosti systému z topologického hľadiska. Môže byť vypočítaný nasledovne:

$$\mathrm{SI}_i = \mathrm{TD}\left(\frac{\partial\phi(j \to k)}{\partial x_i(r \to s)}\right)$$

kde TD je funkcia definovaná ako relatívny počet prípadov, kedy funkcia $\frac{\partial\phi(j\to k)}{\partial x_i(r\to s)}$ nadobúda hodnotu 1 ku všetkým možným prípadom [56].

Birnbaumova dôležitosť $\mathbf{BI}_i$

BI_i je podobná ako SI_i s jedným zásadným rozdielom a to, že BI_i berie do úvahy pravdepodobnosti stavov jednotlivých komponentov. BI_i môže byť vypočítaná nasledovne [56, 57, 58]:

$$\mathrm{BI}_i = \Pr\left\{\frac{\partial\phi(j \to k)}{\partial x_i(r \to s)} = 1\right\}.$$

Nekoherentné systémy

V predchádzajúcej časti boli predstavené základné termíny zo spoľahlivostného inžinierstva. Táto časť je zameraná na nekoherentné systémy, ktoré sú preskúmané viac do hĺbky, čo je taktiež hlavné zameranie našej práce.

Štruktúrna funkcia nekoherentných viacstavových systémov

Podobne ako pri dvojstavových systémoch, aj viacstavové systémy je možné rozdeliť v závislosti od správania sa systému na dve skupiny: koherentné a nekoherentné. Koncept koherentných a nekoherentných systémov bol naskôr preskúmaný pre analýzu viacstavových systémov v publikáciách [33, 34, 35, 36]. Tento koncept je zovšeobecnený z konceptu pre dvojstavový systém [10].

Podmienky relevancie majú niekoľko interpretácií pre viacstavové systémy. Publikácie od autorov Barlow & Wu [8], El-Neweihi, Proschan, Sethuraman [9] and Natvig [37] sa venujú skúmaniu základných princípov koherencie viacstavových systémov a navrhli niektoré definície relevancie komponentov. Tabuľka 4.15 sumarizuje definície relevancie komponentov. Predpokladáme, že tieto podmienky sú najužitočnejšie pre analýzu spoľahlivosti systémov, pretože nám umožňujú deklarovať silnú a slabú relevanciu v nasledujúcom poradí:

$$\phi(s_i,\boldsymbol{x}) > \phi((s-1)_i,\boldsymbol{x}),$$

$$\phi((m_i-1)_i,\boldsymbol{x}) > \phi(0_i,\boldsymbol{x}),$$

pre všetky $s \in \{1,...,m_i-1\}$ a niektoré $\phi(.{}_i,\boldsymbol{x})$.

V závislosti od typu relevancie boli definované tri typy viacstavových homogénnych systémov:

1. silno koherentné,
2. koherentné,
3. slabo koherentné.

Orientovaná parciálna logická derivácia nekoherentných viacstavových systémov

Logický diferenciálny počet nám umožňuje analýzu viacstavových systémov založenú na logických výrazoch. Analýza spoľahlivosti viacstavových systémov s využitím viachodnotovej logiky umožňuje priamy výpočet ohodnotenia viacstavových systémov. Avšak väčšina známych metód analýzy viacstavových systémov založených na viachodnotovej logike bola vyvinutá pre koherentné systémy [56, 61, 6]. Takéto metódy sa dajú efektívne použiť pri analýze kritických stavov viacstavových systémov [6] a analýze dôležitosti [41]. V rámci našej práce sme navrhli aplikovanie orientovaných parciálnych logických derivácií pri analýze a ohodnotení nekoherentných viacstavových systémov na definovanie kritických stavov systému.

Analýza nekoherentných viacstavových systémov by mala brať do úvahy rozdielny vplyv zmeny funkčnosti komponentu na úroveň funkčnosti systému. Integrované orientované parciálne logické derivácie predstavené v predchádzajúcej sekcii je možné

Table 4.15: The conceptions of relevance for coherent MSS

Relevance	Definition: Homogenous	Definition: Non-homogenous	Coherence type
EPS [34]	for $\forall i = 1,...,n$ and $\forall j = 0,...,m-1$, $\exists\phi(._i,\boldsymbol{x})$ such that $\phi(j_i,\boldsymbol{x}) = j$ and $\phi(l_i,\boldsymbol{x}) \neq j$, $j \neq l$	for $\forall i = 1,...,n$, $\exists\phi(._i,\boldsymbol{x})$ such that $\phi(s_i,\boldsymbol{x}) = j$ and $\phi(r_i,\boldsymbol{x}) \neq j$ for $s \neq r$	strong coherent
G1 [35]	for $\forall i = 1,...,n$ and $\forall j = 1,...,m-1$, $\exists\phi(._i,\boldsymbol{x})$ such that $\phi((j-1)_i,\boldsymbol{x}) < \phi(j_i,\boldsymbol{x})$, $\phi(\boldsymbol{j}) = j$, $j = 1,...,m$	for $\forall i = 1,...,n$ and for $\forall s = 1,...,m_i - 1$, $\exists\phi(._i,\boldsymbol{x})$ such that $\phi((s-1)_i,\boldsymbol{x}) < \phi(s_i,\boldsymbol{x})$	coherent
G2 [35]	for $\forall i = 1,...,n$ and $\forall j = 10,...,m-1$, $\exists\phi(._i,\boldsymbol{x})$ such that $\phi(0_i,\boldsymbol{x}) < \phi((m-1)_i,\boldsymbol{x})$	for $\forall i = 1,...,n$, $\exists\phi(._i,\boldsymbol{x})$ such that $\phi(0_i,\boldsymbol{x}) < \phi((m_i-1)_i,\boldsymbol{x})$	weak coherent
N1 [37]	for $\forall i = 1,...,n$ and $\forall j = 0,...,m$, $\exists\phi(._i,\boldsymbol{x})$ such that $\phi(j_i,\boldsymbol{x}) \geq j$ and $\phi((j-1)_i,\boldsymbol{x}) \leq j-1$	for $\forall i = 1,...,n$, $\forall s = 1,...,m_i - 1$ $\exists\phi(._i,\boldsymbol{x})$ such that $\phi(s_i,\boldsymbol{x}) \geq j$ and $\phi((s-1)_i,\boldsymbol{x}) \leq j$	coherent
BS [60]	for $\forall i = 1,...,n$ and $\forall j = 0,...,m-1$, $\exists\phi(._i,\boldsymbol{x})$ such that $\phi(\boldsymbol{0}) = 0$ and $\phi(\boldsymbol{m-1}) = m-1$	is not defined	weak coherent

použiť na analýzu degradácie úrovní funkčnosti systému v závislosti od fixnej zmeny sledovaného komponentu zo stavu s na stav r. Nenulová hodnota týchto derivácií zodpovedá kritickým stavom i-teho komponentu. Vývoj a zovšeobecnenie týchto derivácií pre nekoherentné viacstavové systémy nám umožňuje definovať nové typy integrovaných orientovaných parciálnych logických derivácií pre fixnú zmenu i-teho komponentu zo stavu s na stav r. Analýza nekoherentných viacstavových systémov pomocou týchto derivácií by mala brať do úvahy každú zmenu úrovne výkonnosti systému j (pre $j \in \{0, 1, ..., m-1\}$) spôsobenú zmenou stavu komponentu. To môže byť definované nasledovne:

$$\frac{\partial \phi(j \downarrow\uparrow)}{\partial x_i(s \to r)} = \begin{cases} 1, & \text{if} \quad \phi(s_i, \boldsymbol{x}) = j \quad \text{and} \quad \phi(r_i, \boldsymbol{x}) \neq j \\ 0, & \text{otherwise} \end{cases}$$

Spomenutá derivácia nám umožňuje identifikovať kritické stavy nekoherentného viacstavového systému pre fixnú úroveň funkčnosti systému j v závislosti od zmeny stavu i-teho komponetu z s na r.

Orientovaná parciálna logická derivácia pre nekoherentný viacstavový systém môže byť zovšeobecnená pre všetky možné úrovne funkčnosti systému nasledovne:

$$\frac{\partial \phi(\downarrow\uparrow)}{\partial x_i(s \to r)} = \bigvee_{j=0}^{m-1} \frac{\partial \phi(j \downarrow\uparrow)}{\partial x_i(s \to r)} = \begin{cases} 1, & \text{if} \quad \phi(s_i, \boldsymbol{x}) \neq \phi(r_i, \boldsymbol{x}) \\ 0, & \text{otherwise} \end{cases} \tag{4.10}$$

Informácie o kritických stavoch získané spomínanými deriváciami nemusia byť postačujúce v niektorých prípadoch, pretože nám umožňujú určite kritické stavy systému vzhľadom na komponent a jeho špecifickú zmenu a zmena úrovne výkonnosti systému je druhoradá. Preto je vhodné navrhnúť deriváciu, ktorá by brala do úvahy špecifickú zmenu úrovne funkčnosti systému. Túto je možné definovať vzhľadom na komponent i a zmenu úrovne funkčnosti systému z j na h nasledovne:

$$\frac{\partial \phi(j \to h)}{\partial x_i} = \begin{cases} 1, & \text{if} \quad \phi(s_i, \boldsymbol{x}) = j \quad \text{and} \quad \phi(r_i, \boldsymbol{x}) = h \\ 0, & \text{otherwise} \end{cases}, \tag{4.11}$$

pre akékoľvek $s, r \in 0, ..., m_i - 1$, kde $s \neq r$.

Táto derivácia je počítaná pre všetky možné zmeny i-tej premennej, ktoré boli definované hodnotami s a r. Táto derivácia môže byť definovaná aj iným spôsobom s

ohľadom na definíciu orientovanej parciálnej logickej derivácie:

$$\frac{\partial\phi(j \to h)}{\partial x_i} = \frac{\partial\phi(j \to h)}{\partial x_i \downarrow} + \frac{\partial\phi(j \to h)}{\partial x_i \uparrow}$$
$$= \bigvee_{s=1}^{m_i-1} \bigvee_{r=0}^{s-1} \frac{\partial\phi(j \to h)}{\partial x_i(s \to r)} + \bigvee_{s=0}^{m_i-2} \bigvee_{r=s+1}^{m_i-1} \frac{\partial\phi(j \to h)}{\partial x_i(s \to r)}$$

Táto derivácia pozostáva z dvoch častí. Prvá z nich nám umožňuje identifikovať kritické stavy systému pre ktoré degradácia i-teho komponentu systému vedie k zmene úrovne funkčnosti systému z hodnoty j na hodnotu h. Druhá časť tejto derivácie nám umožňuje získať kritické stavy systému, kde je zmena úrovne funkčnosti systému spôsobená zvýšením funkčnosti i-teho komponentu systému.

Viachodnotový rozhodovací diagram

Keďže štruktúrna funkcia reálnych systémov môže mať veľký rozmer, je potrebné reprezentovať ju efektívnym spôsobom, napríklad vo forme viachodnotového rozhodovacieho diagramu. Tento diagram je acyklický graf, ktorý spĺňa dve podmienky [62, 63]:

1. graf je kanonický,
2. graf je kompaktný.

Viachodnotový rozhodovací diagram pozostáva z listových vrcholov, ktoré predstavujú úroveň výkonnosti systému a nelistových vrcholov, ktoré predstavujú jednotlivé komponenty systému. Každý z nich má práve m_i výstupných hrán, ktoré je možné interpretovať ako stav komponentu. Cesta z koreňa do listu predstavuje špecifickú kombináciu úrovní funkčnosti komponentov systému a k nim prislúchajúcu úroveň výkonnosti systému.

Keďže tento diagram je ortogonálna forma štruktúrnej funkcie [64], je možné ju použiť aj na pravdepodobnostnú analýzu. V tomto prípade sú hrany vrcholov ohodnotené pravdepodobnosťou jednotlivých stavov.

Reprezentácia viachodnotového rozhodovacieho diagramu pomocou poľa susedov

Grafové štruktúry je možné v počítači reprezentovať rôznymi spôsobmi. Jedným z nich je pole susedov. Táto reprezentácia pozostáva z troch polí, ako je možné vidieť

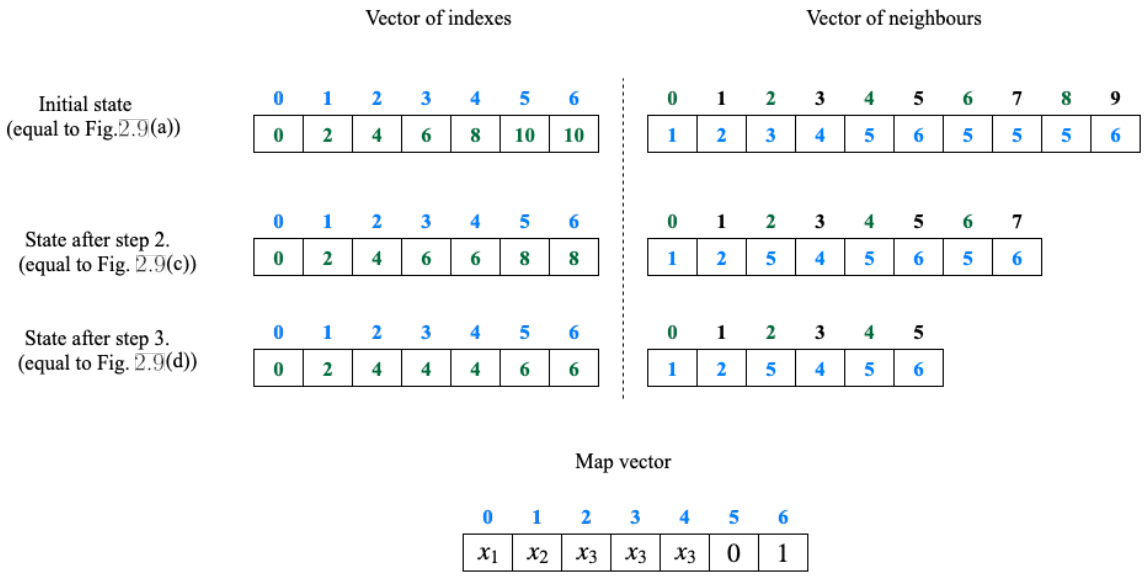

Figure 4.16: DT to MDD reduction using vector of neighbours

na obrázku 4.16, menovite mapovacie pole, pole indexov a pole susedov. Mapovacie pole obsahuje informácie o mapovaní vrcholov rozhodovacieho diagramu na index. Pole indexov obsahuje indexy jednotlivých vrcholov do poľa susedov. Na indexe 0 v poli indexov je zapísaná počiatočná pozícia susedov v poli susedov pre komponent s indexom 0. Na indexe 1 je počiatočná pozícia susedov pre komponent s indexom 1 atď. Pole susedov obsahuje susedné vrcholy jednotlivých vrcholov diagramu.

Reprezentácia viachodnotového rozhodovacieho diagramu pomocou poľa susedov má niekoľko výhod. Jednou z nich je jednoduchá redukcia rozhodovacieho stromu na viachodnotový rozhodovací diagram. Algoritmus tejto redukcie pozostáva z 3 krokov:

1. Redukcia počtu listových vrcholov - toto je možné vyriešiť správnym namapovaním vrcholov do mapovacieho poľa.

2. Odstránenie vrcholov, kde všetky výstupné hrany končia v rovnakom vrchole - toto je možné vyriešiť hľadaním rovnakých čísel v poli susedov pre niektorý vrchol.

3. Odstránenie izomorfných podgrafov - toto je možné vyriešiť hľadaním rovnakých sekvencií pre vrcholy reprezentujúce rovnaký komponent systému.

Neúplne definovaná štruktúrna funkcia viacstavového systému

Hlavný problém tradičných metód v analýze spoľahlivosti je nutnosť úplnej informácie o analyzovanom systéme. Túto však v mnohých reálnych systémoch nemáme. Preto je potrebné použiť iný prístup. V našej práci sme nadviazali na práce našich kolegov popísané v publikáciách [50] a [66]. Pomocou metód dolovania dát sa vytvorí rozhodovací strom, ktorý sa následne redukuje na viachodnotový rozhodovací diagram, čo je forma štruktúrnej funkcie a je možné použiť klasické metódy analýzy spoľahlivosti.

Model spoľahlivosti softvéru založený na syntaktickom strome

Vývoj metód na ohodnotenie spoľahlivosti softvéru je v súčasnosti relevantný problém spoľahlivostného inžinierstva [68, 69]. Väčšina v súčastnosti používaných modelov na analýzu spoľahlivosti softvéru sú pravdepodobnostné modely. Tieto sa dajú použiť na výpočet charakteristík systému ako sú spoľahlivosť, avšak nie je možné ich použiť na analýzu jednotlivých komponentov softvérových systémov a ich vplyvu na funkčnosť systému. Existuje však niekoľko takých, ktoré berú do úvahy aj štruktúru softvéru, avšak je možné ich použiť len v špecifických prípadoch, napríklad keď je softvérový systém implementovaný pomocou architektúry mikroservisov. Preto sme sa rozhodli navrhnúť vlastný model. Hlavný princíp navrhovanej metódy je použiť zdrojový kód na tvorbu spoľahlivostného modelu. Táto metóda pozostáva z 2 krokov. Prvým z nich je tvorba abstraktného syntaktického stromu zo zdrojového kódu. Syntaktický strom je jazykovo nezávislý, je možné ho jednoducho využiť na vyjadrenie štruktúry softvéru a je možné pomocou neho jednoduchým spôsobom odfiltrovať nepodstatné informácie zo zdrojového kódu. Druhým krokom je tvorba spoľahlivostného modelu zo syntaktického stromu. Syntaktický strom je možné transformovať na strom poruchových stavov.

Prípadové štúdie

Použitie všetký spomenutých metód je demonštrované v kapitole 4. Táto kapitola obsahuje 4 prípadové štúdie. Prvá z nich je anesteziologické vyšetrenie. Tento systém je použitý na prezentovanie výpočtu kritických stavov použitím navrhovaných integrovaných logických derivácií nekoherentných viacstavových systémov. Ďalšie dve príapdové štúdie, t.j. prežitie pacientov s hepatitídou a zrážky cyklistov sú použité na

demonštrovanie výpočtu analýzy spoľahlivosti spolu s kvantitatívnou analýzou systémov s neúplne definovanými dátami a taktiež použitie štruktúrnej funkcie vo forme viachodnotového rozhodovacieho diagramu na túto analýzu. Posledná prípadová štúdia predstavuje možnosť výpočtu analýzy spoľahlivosti softvérového systému spolu so všetkými krokmi, t.j. tvorba syntaktického stromu zo zdrojového kódu, tvorba matematického modelu z tohoto stromu a následná kvantitatívna analýza tohto systému.

Záver

Hlavné zameranie našej práce je na analýzu spoľahlivosti nekoherentných viacstavových systémov. Vrámci toho sme si stanovili viacero cieľov, špecificky: 1. Preskúmať metódy analýzy spoľahlivosti viacstavových nekoherentných systémov. 2. Preskúmať metódy redukcie výpočtovej zložitosti štruktúrnej funkcie. 3. Preskúmať metódy tvorby štruktúrnej funkcie založené na neúplne špecifikovaných dátach.

Prvý cieľ sme dosiahli návrhom nového prístupu pre analýzu nekoherentných viacstavových sysémov pomocou matematických metód viachodnotovej logiky. Podibne ako v štúdiách [6, 41] sme interpretovali štruktúrnu funkciu ako viachodnotovú logickú funkciu, čo nám umožnilo použiť metódy viachodnotovej logiky na analýzu a ohodnotenie viacstavového systému. Špecificky sme v rámci našej práce navrhli možnosť analýzy kritických stavov takýchto systémov. Aplikáciu tejto metódy sme demonštrovali na prípadovej štúdii ansesteziologického vyšetrenia.

Druhý cieľ bol dosiahnutý použitím viachodnotového rozhodovacieho diagramu ako reprezentácie štruktúrnej funkcie. Vďaka tomu, že táto reprezentácia je kompaktná, je možné pomocou nej popísať taktiež systémy veľkých rozmerov. Táto reprezentácia je taktiež vhodná na výpočet kvantitatívnej analýzy - topologickej aj pravdepodobnostnej. V rámci tohto cieľa sme sa taktiež rozhodli reprezentovať tento diagram pomocou poľa susedov. Aplikácia tejto metódy bola prezentovaná na príkladoch prežitia pacientov s hepatitídou a zrážok cyklistov.

Posledný špecifikovaný cieľ sme splnili použitím metód dolovania dát a tvorby rozhodovacieho stromu. V rámci našej práce sme popísali algoritmus redukcie rozhodovacieho stromu na viachodnotový rozhodovací diagram pri použití poľa susedov ako reprezentácie tohto diagramu. Aplikácia tejto metódy je demonštrovaná na príkladoch prežitia pacientov s hepatitídou a zrážok cyklistov.

Posledná časť našej práce bola zameraná na spoľahlivosť softvéru. Hlavná myšlienka nami navrhovanej metódy je použitie zdrojového kódu a syntaktického stromu na tvorbu spoľahlivostného modelu. Táto metóda bola prezentovaná na príklade jednoduchého softvéru.

V rámci našej budúcej práce sa chceme zamerať na preskúmanie nekoherentných viacstavových systémov viac do hĺbky. Preskúmame použitie navrhovaných orientovaných parciálnych logických derivácií a kritických stavov na kvantitatívnu analýzu a výpočet ukazovateľov dôležitosti. V ďalšej časti nášho budúceho výskumu preskúmame možnsoť ďalšej redukcie výpočtovej zložitosti analýzy spoľahlivosti systémov veľkých rozmerov, napríklad použitím metód modulárnej dekompozície [94, 44, 95] pri viachodnotových rozhodovacích diagramoch. Nakoniec plánujeme vylepšiť navrhovaný model spoľahlivosti softvéru, keďže v súčasnosti obsahuje najmä 2 problémy. Prvý z nich je jeho veľký rozmer, čo môžme vidieť aj na prezentovanom príklade, kedy aj pomerne jednoduchý zdrojový kód vyústil do štruktúrnej funkcie veľkého rozmeru. Existuje viacero spôsobov ako vyriešť tento problém. Prvým z nich je redukovať veľkosť syntaktického stromu potrebného na tvorbu tohto modelu spájaním a odstraňovaním vrcholov, pokým výsledný syntaktický strom nebude vhodnejšej veľkosti. Takýmto spôsobom bude taktiež vytvorený matematický model menší a celá analýza bude jednoduchšia. Iné riešenie je použiť modulárnu dekompozíciu, ako už bolo spomínané. Použitím tejto metódy môžme rozdeliť spoľahlivostný model na viacero modulov a každý z nich analyzovať zvlášť. Toto zníži čas potrebný na analýzu spoľahlovosti celého systému s použitím metód ako paralelizácia. Ďalší problém tohto modelu je použitie AND logických operácií, čo vedie k nepresnostiam vo vytvorenom modeli. Na vyriešenie tohto problému bude potrebný ďalší výskum. Toto nám pomôže určiť akým spôsobom majú byť prepojené jednotlivé udalosti v strome poruchových stavov, aby sme zvýšili presnosť vytvoreného modelu.

Bibliography

[1] What are microservices? `https://www.redhat.com/en/topics/microservices/what-are-microservices`. [Accessed: 15-Apr-2020].

[2] Zhigang Zang, Qiaoyan Wen, and Kangming Xu. A fault tree based microservice reliability evaluation model. *IOP Conference Series: Materials Science and Engineering*, 569:032069, August 2019.

[3] Ali Sedaghatbaf and Mohammad Abdollahi Azgomi. Reliability evaluation of UML/DAM software architectures under parameter uncertainty. *IET Software*, 12(3):236–244, June 2018.

[4] Anatoly Lisnianski and Gregory Levitin. *Multi-State System Reliability*. WORLD SCIENTIFIC, March 2003.

[5] Enrico Zio. *An Introduction to the Basics of Reliability and Risk Analysis*. World Scientific Publishing Company, February 2007.

[6] Elena Zaitseva and Vitaly Levashenko. Reliability analysis of multi-state system with application of multiple-valued logic. *International Journal of Quality & Reliability Management*, 34(6):862–878, June 2017.

[7] Hananeh Aliee, Emanuele Borgonovo, Michael Glaßa, and Jurgen Teicha. On the boolean extension of the birnbaum importance to non-coherent systems. *Reliability Engineering & System Safety*, 160:191–200, April 2017.

[8] Richard E. Barlow and Alexander S. Wu. Coherent systems with multi-state components. *Mathematics of Operations Research*, 3(4):275–281, November 1978.

[9] Emad El-Neweihi, Frank Proschan, and Jayaram Sethuraman. Multistate coherent systems. *Journal of Applied Probability*, 15(4):675–688, December 1978.

[10] Richard E. Barlow. *Statistical theory of reliability and life testing: probability models*. Holt, Rinehart and Winston, New York, 1974.

[11] John D. Andrews and Sally Beeson. Birnbaum's measure of component importance for noncoherent systems. *IEEE Transactions on Reliability*, 52(2):213–219, June 2003.

[12] Sally Beeson and John D. Andrews. Importance measures for non-coherent-system analysis. *IEEE Transactions on Reliability*, 52(3):301–310, September 2003.

[13] Gregory Levitin, Liudong Xing, and Yuanshun Dai. Reliability of non-coherent warm standby systems with reworking. *IEEE Transactions on Reliability*, 64(1):444–453, March 2015.

[14] Hoang Pham. Optimal design of a class of noncoherent systems. *IEEE Transactions on Reliability*, 40(3):361–363, August 2015.

[15] Klaus D. Heidtmann. A class of noncoherent systems and their reliability analysis. In *Proceedings of 11th Annual Symposium on Fault Tolerant Computing*, pages 96–98. IEEE Comput. Soc, 1981.

[16] Beena Nailwal and Suraj B. Singh. Reliability and sensitivity analysis of an operating system with inspection in different weather conditions. *International Journal of Reliability, Quality and Safety Engineering*, 19(2), April 2012.

[17] Yuchang Mo, Liudong Xing, and Joanne B. Dugan. Performability analysis of k-to-l-out-of-n computing systems using binary decision diagrams. *IEEE Transactions on Dependable and Secure Computing*, 15(1):126–137, 2018.

[18] Francesco Di Maio, Samuele Baronchelli, Matteo Vagnoli, and Enrico Zio. Determination of prime implicants by differential evolution for the dynamic reliability analysis of non-coherent nuclear systems. *Annals of Nuclear Energy*, 102:91 – 105, 2017.

[19] Jussi Vaurio. Importances of components and events in non-coherent systems and risk models. *Reliability Engineering & System Safety*, 147:117–122, March 2016.

[20] Shambhu J. Upadhyaya and Hoang Pham. Analysis of noncoherent systems and an architecture for the computation of the system reliability. *IEEE Transactions on Computers*, 42(4):484–493, 1993.

[21] Miroslav Kvassay, Elena Zaitseva, Vitaly Levashenko, and Jozef Kostolny. Reliability analysis of multiple-outputs logic circuits based on structure function approach. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2016.

[22] Suprasad V. Amari. Performance computing failure frequency of noncoherent systems. *International Journal of Performability Engineering*, 2(2):123, 2006.

[23] Qin Zhang and Qizhi Mei. Reliability analysis for a real non-coherent system. *IEEE Transactions on Reliability*, R-36(4):436–439, October 1987.

[24] Peter S. Jackson. On the s-importance of elements and prime implicants of non-coherent systems. *IEEE Transactions on Reliability*, R-32(1):21–25, 1983.

[25] Winfrid G. Schneeweiss. A short boolean derivation of mean failure frequency for any (also non-coherent) system. *Reliability Engineering & System Safety*, 94(8):1363 – 1367, 2009.

[26] Miroslav Kvassay, Elena Zaitseva, Vitaly Levashenko, and Jozef Kostolny. Binary decision diagrams in reliability analysis of standard system structures. In *2016 International Conference on Information and Digital Technologies (IDT)*, pages 164–172, 2016.

[27] Emanuele Borgonovo. The reliability importance of components and prime implicants in coherent and non-coherent systems including total-order interactions. *European Journal of Operational Research*, 204(3):485 – 495, 2010.

[28] Guenter Becker and Leonidas Camarinopoulos. Failure frequencies of non-coherent structures. *Reliability Engineering & System Safety*, 41(3):209 – 215, 1993.

[29] Chris Garrett and George Apostolakis. Context in the risk assessment of digital systems. *Risk Analysis*, 19(1):23–32, 1999.

[30] Vaurio Jussi. Ideas and developments in importance measures and fault-tree techniques for reliability and risk analysis. *Reliability Engineering & System Safety*, 95:99–107, February 2010.

[31] Antoine Rauzy and Yves Dutuit. Exact and truncated computations of prime implicants of coherent and non-coherent fault trees within aralia. *Reliability Engineering & System Safety*, 58(2):127 – 144, 1997. ESREL '95.

[32] Ayyoub Juba Imakhlaf, Yunhui Hou, and Mohamed Sallak. Evaluation of the reliability of non-coherent systems using binary decision diagrams. *IFAC-PapersOnLine*, 50(1):12243–12248, July 2017.

[33] Abdulrahman M. Abouammoh and M. A. Al-Kadi. Component relevancy in multistate reliability models. *IEEE Transactions on Reliability*, 40(3):370–374, 1991.

[34] Emad El-Neweihi and Frank Proschan. Degradable systems:a survey of multistate system theory. *Communications in Statistics - Theory and Methods*, 13(4):405–432, January 1984.

[35] William S. Griffith. Multistate reliability models. *Journal of Applied Probability*, 17(3):735–744, September 1980.

[36] Praveen P. Gupta and S. C. Agarwal. A boolean algebra method for reliability calculations. *Microelectronics Reliability*, 23(5):863–865, January 1983.

[37] Bent Natvig. Two suggestions of how to define a multistate coherent systems. *J. Applied Probability*, 15:675–688, 1978.

[38] Andre Bossche. The top-event's failure frequency for non-coherent multi-state fault trees. *Microelectronics Reliability*, 24(4):707 – 715, 1984.

[39] Andre Bossche. Calculation of critical importance for multi-state components. *IEEE Transactions on Reliability*, R-36(2):247–249, June 1987.

[40] Elena Zaitseva, Miroslav Kvassay, Vitaly Levashenko, and Jozef Kostolny. Introduction to knowledge discovery in medical databases and use of reliability analysis in data mining. In *Proceedings of the 2015 Federated Conference on Computer Science and Information Systems*. IEEE, October 2015.

[41] Miroslav Kvassay, Elena Zaitseva, and Vitaly Levashenko. Importance analysis of multi-state systems based on tools of logical differential calculus. *Reliability Engineering and System Safety*, 165:302–316, 2017.

[42] Miroslav Kvassay, Vitaly Levashenko, and Elena Zaitseva. Analysis of minimal cut and path sets based on direct partial boolean derivatives. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 230(2):147–161, 2016.

[43] Mihir R. Choudhury and Kartik Mohanram. Reliability analysis of logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(3):392–405, March 2009.

[44] Bent Natvig. *Multistate Systems Reliability Theory with Applications*. John Wiley & Sons, Ltd, January 2011.

[45] Anatoly Lisnianski, Ilia Frenkel, and Yi Ding. *Multi-state System Reliability Analysis and Optimization for Engineers and Industrial Managers*. Springer London, 2010.

[46] Terje Aven, Piero Baraldi, Roger Flage, and Enrico Zio. *Uncertainty in Risk Assessment: The Representation and Treatment of Uncertainties by Probabilistic and Non-Probabilistic Methods*. Wiley, 2013.

[47] Way Kuo and Xiaoyan Zhu. *Importance Measures in Reliability, Risk, and Optimization*. John Wiley & Sons, Ltd, May 2012.

[48] Miroslav Kvassay and Elena Zaitseva. Topological analysis of multi-state systems based on direct partial logic derivatives. In *Springer Series in Reliability Engineering*, pages 265–281. Springer International Publishing, August 2017.

[49] Miroslav Kvassay, Elena Zaitseva, and Vitaly Levashenko. Minimal cut and minimal path vectors in reliability analysis of binary- and multi-state systems. In *ICTERI*, 2017.

[50] Elena Zaitseva and Vitaly Levashenko. Construction of a reliability structure function based on uncertain data. *IEEE Transactions on Reliability*, 65(4):1710–1723, December 2016.

[51] Elena Zaitseva and Vitaly Levashenko. Multiple-valued logic mathematical approaches for multi-state system reliability analysis. *Journal of Applied Logic*, 11(3):350–362, September 2013.

[52] Moiez A. Tapia, Tayeb A. Guima, and Abdollah Katbab. Calculus for a multivalued-logic algebraic system. *Applied Mathematics and Computation*, 42(3):255–285, April 1991.

[53] Miroslav Kvassay, Elena Zaitseva, Jozef Kostolny, and Vitaly Levashenko. Reliability analysis of noncoherent systems based on logical differential calculus. In *Risk, Reliability and Safety: Innovating Theory and Practice*, pages 1367–1374. CRC Press, September 2016.

[54] Elena Zaitseva, Vitaly Levashenko, Jan Rabcan, and Emil Krsak. Application of the structure function in the evaluation of the human factor in healthcare. 2020, 12, 93. *Symmetry*, 12:93, 2020.

[55] Anatoly Lisnianski and Gregory Levitin. Multi-state system reliability: Assessment, optimization and application. *Proc. Eng.*, 6, 01 2003.

[56] Miroslav Kvassay, Elena Zaitseva, Vitaly Levashenko, and Jozef Kostolny. Minimal cut vectors and logical differential calculus. In *2014 IEEE 44th International Symposium on Multiple-Valued Logic*. IEEE, May 2014.

[57] Miroslav Kvassay, Elena Zaitseva, Jozef Kostolny, and Vitaly Levashenko. Importance analysis of multi-state systems based on integrated direct partial logic derivatives. In *2015 International Conference on Information and Digital Technologies*. IEEE, July 2015.

[58] Miroslav Kvassay, Elena Zaitseva, and Vitaly Levashenko. Importance analysis of multi-state systems based on tools of logical differential calculus. *Reliability Engineering & System Safety*, 165:302–316, September 2017.

[59] Nader Ebrahimi. Multistate reliability models. *Naval Research Logistics Quarterly*, 31(4):671–680, December 1984.

[60] Thomas H. Savits Henry W. Block. A decomposition for multistate monotone systems. *J.Applied Probability*, 19:391–402, 1982.

[61] Peter Sedlacek, Jan Rabcan, and Jozef Kostolny. Importance analysis of multi-state system based on incompletely specified data by multi-valued decision diagrams.

In *2019 International Conference on Information and Digital Technologies (IDT)*. IEEE, June 2019.

[62] Michael D. Miller and Rolf Drechsler. Implementing a multiple-valued decision diagram package. In *Proceedings. 1998 28th IEEE International Symposium on Multiple- Valued Logic (Cat. No.98CB36138)*. IEEE Comput. Soc.

[63] Yuchang Mo, Liudong Xing, and Suprasad V. Amari. A multiple-valued decision diagram based method for efficient reliability analysis of non-repairable phased-mission systems. *IEEE Transactions on Reliability*, 63(1):320–330, March 2014.

[64] Michael D. Miller and Rolf Drechsler. On the construction of multiple-valued decision diagrams. In *Proceedings 32nd IEEE International Symposium on Multiple-Valued Logic*. IEEE Comput. Soc.

[65] Yuchang Mo, Liudong Xing, Lirong Cui, and Shubin Si. MDD-based performability analysis of multi-state linear consecutive- k -out-of- n : F systems. *Reliability Engineering & System Safety*, 166:124–131, October 2017.

[66] Elena Zaitseva, Vitaly Levashenko, Miroslav Kvassay, and Jan Rabcan. Application of ordered fuzzy decision trees in construction of structure function of multi-state system. In *Information and Communication Technologies in Education, Research, and Industrial Applications*, pages 56–75. Springer International Publishing, 2017.

[67] Dragan Jankovic, Radomir S. Stankovic, and Rolf Drechsler. Reduction of sizes of multi-valued decision diagrams by copy properties. In *Proceedings. 34th International Symposium on Multiple-Valued Logic*. IEEE Comput. Soc.

[68] R. Dillibabu P. Govindasamy. Development of software reliability models using a hybrid approach and validation of the proposed models using big data. *The Journal of Supercomputing*, 76:2252–2265, April 2020.

[69] Stefano Russoa Roberto Pietrantuono, Peter Popov. Reliability assessment of service-based software under operational profile uncertainty. *Reliability Engineering & System Safety*, 204:1–13, December 2020.

[70] Kazuhira Okumoto John D. Musa, Anthony Iannino. *Software reliability-measurement, prediction, application*. McGraw-Hill, 1987.

[71] Shigeru Yamada. *Software Reliability Modeling*. Springer, 2014.

[72] Min Xie. *Software Reliability Modeling*. Springer, 1991.

[73] Pratik Roy, G.S. Mahapatra, Pooja Rani, S.K.Pandey, and Kashi N.Dey. Robust feedforward and recurrent neural network based dynamic weighted combination models for software reliability prediction. *Applied Soft Computing*, 22:629–637, September 2014.

[74] Z. Jelinski and P. Moranda. SOFTWARE RELIABILITY RESEARCH. In *Statistical Computer Performance Evaluation*, pages 465–484. Elsevier, 1972.

[75] Yuan-Shun Dai Min Xie, Kim-Leng Poh. *Computing System Reliability*. Kluwer Academic Publishers, 2004.

[76] Maxim S. Finkelstein. A point-process stochastic model for software reliability. *Reliability Engineering & System Safety*, 63:67–71, January 1999.

[77] Giuseppe Di Marco Alberto Pasquini, Elio De Agostino. An input-domain based method to estimate software reliability. *IEEE Transaction on Reliability*, 45:95–105, March 1996.

[78] Arne Nordmann Peter Munk. Model-based safety assessment with sysml and component fault trees: application and lessons learned. *Software and Systems Modeling*, 19:889–910, February 2020.

[79] Bernhard Kaiser, Catharina Gramlich, and Marc Förster. State/event fault trees—a safety analysis model for software-controlled systems. *Reliability Engineering & System Safety*, 92:1521–1537, November 2007.

[80] Christoph A. Thieme, Ali Mosleh, Ingrid B. Utne, and Jeevith Hegde. Incorporating software failure in risk analysis – part 1: Software functional failure mode classification. *Reliability Engineering & System Safety*, 197:106803, 2020.

[81] Michal Duracik, Emil Krsak, and Patrik Hrkut. Issues with the detection of plagiarism in programming courses on a larger scale. In *2018 16th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. IEEE, November 2018.

[82] Michal Duracik, Emil Krsak, and Patrik Hrkut. Source code representations for plagiarism detection. In *Communications in Computer and Information Science*, pages 61–69. Springer International Publishing, 2018.

[83] Iulian Neamtiu, Jeffrey S. Foster, and Michael Hicks. Understanding source code evolution using abstract syntax tree matching. In *Proceedings of the 2005 international workshop on Mining software repositories - MSR '05*. ACM Press, 2005.

[84] Qiuyuan Chen, Han Hu, and Zhaoyi Liu. Code summarization with abstract syntax tree. In *Communications in Computer and Information Science*, pages 652–660. Springer International Publishing, 2019.

[85] Vartika Agrahari and Sridhar Chimalakonda. AST[AR] – towards using augmented reality and abstract syntax trees for teaching data structures to novice programmers. In *2020 IEEE 20th International Conference on Advanced Learning Technologies (ICALT)*. IEEE, July 2020.

[86] Sohag Kabir. An overview of fault tree analysis and its application in model based dependability analysis. *Expert Systems with Applications*, 77:114–135, July 2017.

[87] Jichuan Kang, Liping Sun, and C. Guedes Soares. Fault tree analysis of floating offshore wind turbines. *Renewable Energy*, 133:1455–1467, April 2019.

[88] Svetlana Yanushkevich, Michael Miller, Vlad Shmerko, and Radomir Stankovic. *Decision Diagram Techniques for Micro- and Nanoelectronic Design Handbook*, volume 2. CRC Press, Boca Raton, FL, dec 2005.

[89] Hepatitis data set. `https://archive.ics.uci.edu/ml/datasets/hepatitis`. [Accessed 2019-04-16].

[90] Tsutomu Sasao. On a minimization of variables to represent sparse multi-valued input decision functions. In *2019 IEEE 49th International Symposium on Multiple-Valued Logic (ISMVL)*. IEEE, May 2019.

[91] Bicycle crashes. `https://catalog.data.gov/dataset/bicycle-crashes`. [Accessed 2019-09-05].

[92] The .net compiler platform sdk (roslyn apis) | microsoft docs. https://docs.microsoft.com/sk-sk/dotnet/csharp/roslyn-sdk/. (Accessed on 11/02/2020).

[93] Class ast. https://www.ibm.com/support/knowledgecenter/SS5JSH_9.5.0/org.eclipse.jdt.doc.isv/reference/api/org/eclipse/jdt/core/dom/AST.html. (Accessed on 11/07/2020).

[94] Z. W. Birnbaum and J. D. Esary. Modules of coherent binary systems. *Journal of the Society for Industrial and Applied Mathematics*, 13(2):444–462, June 1965.

[95] Miroslav Kvassay, Patrik Rusnak, and Jan Rabcan. Time-dependent analysis of series-parallel multistate systems using structure function and markov processes. In *Advances in System Reliability Engineering*, pages 131–165. Elsevier, 2019.

Appendices

List of Publications

1. Erik Parso ... [et al.]. FRIMAN. In *Central European researchers journal*, vol. 2, iss. 2, pp. 70-76, 2016

2. Peter Sedlacek. Tool for topological reliability analysis of reversible logic circuits. In *Central European Researchers Journal*, vol. 4, iss. 2, pp 10-16, 2018

3. Peter Sedláček and Monika Václavková. Tool for supporting Education process in Information technology. In *16th IEEE International Conference on Emerging eLearning Technologies and Applications*, pp. 483-488, 2018

4. Elena Zaitseva, Vitaly Levashenko, and Peter Sedlacek. Reliability analysis based on incompletely specified data. In *Pattern recognition and information processing*, pp. 20-32, 2019

5. Patrik Rusnak ... [et al.]. Structure function based methods in evaluation of availability of healthcare system. In *10th International Conference on Dependable Systems, Services and Technologies*, pp. 13-18, 2019

6. Miroslav Kvassay, Patrik Rusnak, Peter Sedlacek. Computation of Birnbaum's importance using logic differential calculus. In *42nd International conference on telecommunications and signal processing*, pp. 613-616, 2019

7. Patrik Rusnak ... [et al.]. Logic differential calculus in time-dependent importance analysis based on minimal cut vectors. In *TELSIKS 2019*, pp. 74-77, 2019

8. Michal Mrena, Peter Sedlacek and Miroslav Kvassay. Practical applicability of advanced implementations of priority queues in finding shortest paths. In *Information and digital technologies 2019*, pp. 335-344, 2019

9. Peter Sedlacek, Jan Rabcan and Jozef Kostolny. Importance analysis of multi-state system based on incompletely specified data by multi-valued decision diagrams. In *Information and digital technologies 2019*, pp. 409-416, 2019

10. Monika Václavková, Marek Kvet and Peter Sedlacek. Graphical development environment for object programming teaching support. In *15th International Scientific Conference on Informatics*, pp 435-440, 2019

11. Peter Sedlacek, Marek Kmec and Patrik Rusnak. Software Visualization Application for Threads Synchronization Handling in Operating Systems. In *18th International Conference on Emerging eLearning Technologies and Applications*, pp. 580-585, 2020

12. Dominik Smalik ... [et al.] Software Tool for Importance Measures Computation Used in Reliability Analysis. In *18th International Conference on Emerging eLearning Technologies and Applications*, pp. 620-627, 2020