

Fakulta riadenia a informatiky

Výpočet indexov dôležitosti pomocou viacrozmerného rozhodovacieho diagramu

Bakalárska práca

Peter Krajkovič

Študijný program: Informatika

Študijný odbor: Informatika

Školiace pracovisko: Žilinská univerzita v Žiline,

Vedúci bakalárskej práce: Ing. Ján Rabčan, PhD.

Žilina 2024

Namiesto tejto strany treba vložiť zadanie záverečnej práce

Do elektronickej verzie práce vložte **naskenované** zadanie záverečnej práce ako obrázok zväčšený na celú veľkosť papiera.

Čestné vyhlásenie

Vyhlasujem, že som zadanú bakalársku prácu vypracoval samostatne, pod odborným vedením vedúceho práce/školiteľa a používal som len literatúru uvedenú v práci.

Žilina 11. apríla 2024

podpis

Poďakovanie

Rád by som venoval poďakovanie vedúcemu bakalárskej práce Ing. Janovi Rabčanovi PHD. a taktiež Ing. Michalovi Mrenovi za pravidelnú komunikáciu, poskytnuté materiály a spoločnú pomoc pri riešení vzniknutých problémov.

Abstrakt

KRAJKOVIČ, Peter: Výpočet indexov dôležitosti pomocou viacrozmerného rozhodovacieho diagramu. [Bakalárska práca] / Peter Krajkovič. – Žilinská univerzita. Fakulta riadenia a informatiky; Katedra informatiky. – Školiteľ: Ing. Ján Rabčan, PhD. Žilina: FRI ŽU, 2024.

Baklárska práca prezentuje výsledky implementácie algoritmu pre výpočet štrukturálneho indexu komponentov vo viacstavových systémoch. Viacstavové systémy sú reprezentované ako viacrozmerné rozhodovacie diagramy. Práca analyzuje a porovnáva rýchlosť výpočtu štrukturálnych indexov v systémoch reprezentovaných pravdivostnými tabuľkami a systémoch reprezentovaných viacrozmernými rozhodovacími diagramami.

Cieľom práce je preukázať efektívnosť a výhody použitia viacrozmerných rozhodovacích diagramov pri výpočte štrukturálnych indexov komponentov vo viacstavových systémoch. V rámci práce je implementovaný algoritmus, ktorý umožňuje výpočet týchto indexov pomocou smerodajných logických parciálnych derivácií.

Výsledky ukazujú, že využitie viacrozmerných rozhodovacích diagramov môže výrazne zvýšiť efektivitu výpočtu štrukturálnych indexov v porovnaní s tradičným prístupom využívajúcim pravdivostné tabuľky. V práci sú obsiahnuté obmedzenia výpočtu pomocou rozhodovacích diagramov a navrhnuté metódy, ako sa vysporiadať s obmedzeniami, alebo vylepšiť existujúce algoritmy.

Kľúčové slová: viacrozmerný rozhodovací diagram, index dôležitosti, smerodajné logické parciálne derivácie, viacstavový systém (Krajkovič, 2024).

Abstract

KRAJKOVIČ, Peter: Calculation of importance measures using a multivalued decision diagram. [Bachelor's thesis] / Peter Krajkovič. – University of Žilina. Faculty of Management Science and Informatics; Department of Informatics. – Supervisor: Ing. Ján Rabčan, PhD. Žilina: FRI ŽU, 2024.

The bachelor's thesis presents the results of implementing an algorithm for calculating the structural importance measures of components in multistate systems. Multistate systems are represented as multivalued decision diagrams. The thesis analyzes and compares the speed of calculating structural importance measures in systems represented by truth tables and systems represented by multivalued decision diagrams.

The aim of the thesis is to demonstrate the efficiency and advantages of using multivalued decision diagrams in calculating the structural importance measures of components in multistate systems. An algorithm is implemented within the thesis, allowing the calculation of these importance measures using directional partial logic derivatives.

The results show that the use of multivalued decision diagrams can significantly increase the efficiency of calculating structural importance measures compared to the traditional approach using truth tables. The thesis includes limitations of calculation using decision diagrams and proposes methods to deal with limitations or improve existing algorithms.

**Keywords:** multivalued decision diagram, importance measures, directional partial logic derivatives, multistate system

Obsah

Zoznam obrázkov............................................................................................................6  
Zoznam tabuliek ..............................................................................................................7

Zoznam skratiek ..............................................................................................................8

[Úvod 9](#_Toc164725082)

[1 ANALýza DOSTUPNých zdrojov 10](#_Toc164725083)

[1.1 MSS 10](#_Toc164725084)

[1.2 Štrukturálna funkcia 10](#_Toc164725085)

[1.2.1 Príklad štrukturálnej funkcie 11](#_Toc164725086)

[1.3 MDD 11](#_Toc164725087)

[1.3.1 Štruktúra MDD 12](#_Toc164725088)

[1.4 Indexy dôležitosti 13](#_Toc164725089)

[1.4.1 Štrukturálny index 14](#_Toc164725090)

[1.4.2 Spoľahlivostné indexy dôležitosti 15](#_Toc164725091)

[1.4.3 Celoživotné indexy dôležitosti 15](#_Toc164725092)

[1.4.4 Ďalšie indexy dôležitosti 15](#_Toc164725093)

[1.5 Smerodajné parciálne logické derivácie 16](#_Toc164725094)

[1.5.1 Cofactor 16](#_Toc164725095)

[1.5.2 Apply 18](#_Toc164725096)

[1.5.3 Transform 20](#_Toc164725097)

[2 IMPLEMENTáCIA 21](#_Toc164725098)

[2.1 Výpočet štrukturálneho indexu 21](#_Toc164725099)

[2.1.1 Návrh MDD pre výpočet indexu dôležitosti 21](#_Toc164725100)

[2.1.1.1 Získanie logických levelov 22](#_Toc164725101)

[2.1.2 Použitie DPLD 24](#_Toc164725102)

[2.1.3 Satisfy count 25](#_Toc164725103)

[2.1.4 Vylepšenie SatisfyCount 26](#_Toc164725104)

[3 Verifikácia algoritmu 28](#_Toc164725105)

[3.1 Ručná verifikácia 28](#_Toc164725106)

[3.2 Automatická verifikácia 30](#_Toc164725107)

[4 Porovnanie rýchlosti výpočtu 31](#_Toc164725108)

[Záver 32](#_Toc164725109)

Zoznam obrázkov

[Obrázok 1 Reprezentácia systému pomocou MDD / pravdivostnej tabuľky 10](#_Toc164725184)

[Obrázok 2 Neusporiadaný, nekompletný MDD 12](#_Toc164725185)

[Obrázok 3 Usporiadaný,kompletný MDD 12](#_Toc164725186)

[Obrázok 4 Príklad jenoduchého sériovo-paralelného systému 13](#_Toc164725187)

[Obrázok 5 Cofactor(2i, 0, 1) 17](#_Toc164725188)

[Obrázok 6 MDD pre cofactor 17](#_Toc164725189)

[Obrázok 7 MDD pre získanie logických levelov 23](#_Toc164725190)

[Obrázok 8 MDD pre ručnú verifikáciu 28](#_Toc164725191)

[Obrázok 9 Ručná verifikácia - tretia zmena 29](#_Toc164725192)

[Obrázok 10 Ručná verifikácia - druhá zmena 29](#_Toc164725193)

[Obrázok 11 Ručná verifikácia - prvá zmena 29](#_Toc164725194)

Zoznam tabuliek

[Tabuľka 1 Pravdivostná tabuľka pre ručnú verifikáciu 28](#_Toc164725551)

[Tabuľka 2 Porovnanie rýchlosti výpočtu SI pomocou tabuľky / MDD 31](#_Toc164725552)

Zoznam skratiek

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Skratka** | **Anglický význam** | | | **Slovenský význam** |
| MDD | multivalued decision diagram | | | viacrozmerný rozhodovací diagram |
| BDD | binary decision diagram | | | binárny rozhodovací diagram |
| DPLD | | directional partial logic derivative | smerodajná parciálna logická derivácia | |
| IDPLD | | integrated partial logic derivative | integrovaná parciálna logická derivácia | |
| MSS | | multistate system | viacstavový systém | |
| BSS | | binary-state system | binárny systém | |
| SI | | structural index | štrukturálny index | |
|  | |  |  | |
|  | |  |  | |
|  | |  |  | |
|  | |  |  | |

Úvod

V súčasnej dátovej dobe, v ktorej sa množstvo informácii neustále zväčšuje a ich skúmanie je dôležitou súčasťou viacerých odvetví, je dôležité klásť dôraz na ich efektívne spracovanie, vizualizáciu a analýzu.

Dôležitosť premenných v analýze dát je kritickým aspektom, ktorý ovplyvňuje úspešnosť a interpretáciu analytických procesov. Identifikácia a pochopenie dôležitosti jednotlivých premenných umožňuje lepšie porozumieť vzťahom v systémoch, čo vedie k presnejším modelom a zmysluplnejším výsledkom analýzy.

Premenné, ktoré majú významný vplyv na sledované javy alebo cieľovú premennú, môžu výrazne ovplyvniť výsledky analýzy. Identifikácia týchto kľúčových faktorov je dôležitá pre pochopenie dát a správne formulovanie analytických modelov.

Moderné systémy sa skladajú z mnohých vzájomne sa ovplyvňujúcich prvkov. Identifikácia dôležitých prvkov umožňuje zjednodušiť a optimalizovať modely systémov tým, že sa zameriavajú na relevantné faktory a eliminujú nepodstatné premenné. Tým sa zvyšuje efektivita a interpretovateľnosť modelov a analýz.

Dôležité premenné môžu byť kľúčové pre predikciu budúcich udalostí alebo pre prognózu vývoja sledovaných javov. Ich správna identifikácia a hodnotenie umožňuje vytvoriť spoľahlivé prediktívne modely a zlepšiť presnosť predikcií. Ich identifikácia umožňuje aj lepšie porozumieť vplyvu jednotlivých faktorov na sledované javy a identifikovať potenciálne riziká alebo nežiadúce trendy v dátach. Tieto poznatky môžu byť cenné pre plánovanie a riadenie rizík v rôznych oblastiach a odvetviach vrátane strojárstva, elektroniky, informatiky, dopravy, energetiky a mnohých ďalších.

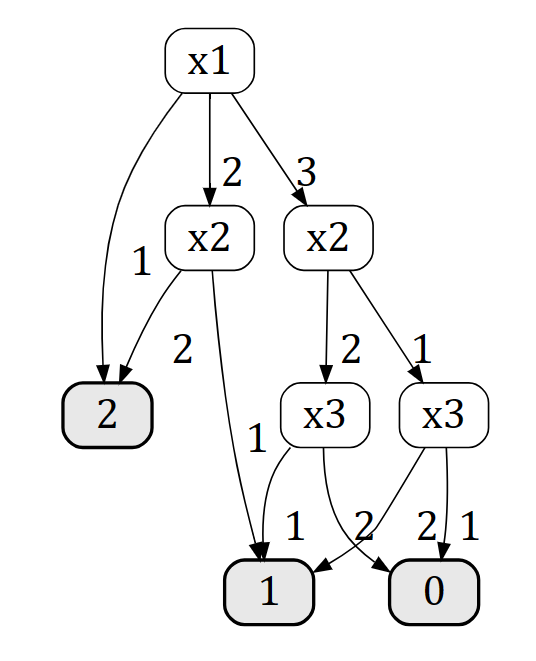
# ANALýza DOSTUPNých zdrojov

## MSS

Viacstavové systémy (MSS) sú typom systémov, ktoré môžu existovať vo viacerých stavoch. Tieto stavy môžu byť diskrétne, napríklad funkčnosť/nefunkčnosť systému alebo spojité, napríklad teplota. Každý stav predstavuje súbor podmienok, za akých systém nadobudne daný stav. Systém môže prechádzať z jedného stavu do druhého na základe zmeny v podmienkach. Akýkoľvek systém skladajúci sa z rozličných viacstavových jednotiek, ktoré majú kumulatívny efekt na výkon systému je nutné považovať za MSS. V skutočnosti, výkonnosť takého systému závisí na dostupnosti jeho jednotiek, keďže rôzne počty dostupných jednotiek môžu poskytnúť rôzne úrovne výkonu úloh. [1]

Na obrázku č.1 sa nachádza príklad reprezentácie trojstavového systému s tromi premennými, pričom premenná x1 môže nadobúdať 3 hodnoty, premenné x2 a x3 môžu nadobúdať dve hodnoty. Rovnaký systém sa dá vyjadriť pravdivostnou tabuľkou alebo pomocou MDD.

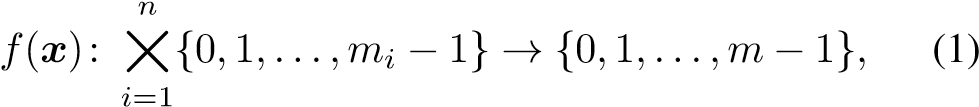
|  |  |  |  |
| --- | --- | --- | --- |
| premenné | | |  |
| x1 | x2 | x3 | stav systému |
| 1 | 1 | 1 | 2 |
| 1 | 1 | 2 | 2 |
| 1 | 2 | 1 | 2 |
| 1 | 2 | 2 | 2 |
| 2 | 1 | 1 | 1 |
| 2 | 1 | 2 | 1 |
| 2 | 2 | 1 | 2 |
| 2 | 2 | 2 | 2 |
| 3 | 1 | 1 | 0 |
| 3 | 1 | 2 | 1 |
| 3 | 2 | 1 | 1 |
| 3 | 2 | 2 | 0 |
|  |  |  |  |



Obrázok 1 Reprezentácia systému pomocou MDD / pravdivostnej tabuľky

## Štrukturálna funkcia

Štrukturálna funkcia vo viacstavových systémoch je matematická funkcia, ktorá popisuje vzťah medzi stavmi systému a jeho komponentami. Táto funkcia je základným stavebným prvkom viacrozmerných rozhodovacích diagramov a slúži na modelovanie dynamiky systému v závislosti od stavov jeho komponentov.

Matematicky je štrukturálna funkcia často definovaná ako funkcia, ktorá mapuje vstupy (rozhodnutia a stavy komponentov) na výstupy (stav systému). Nech X je množina vstupov (stavy komponentov), a Y je množina výstupov (stav systému), potom štrukturálna funkcia f:X→Y popisuje vzťah medzi vstupmi a výstupmi systému. Bližšie je štrukturálna funkcia popísaná ako [2]:

, kde n je počet komponentov, x je vektor komponentov, mi je veľkosť domény i-teho komponentu a m je počet stavov systému. X rozkladá karteziánsky súčin.

### Príklad štrukturálnej funkcie

Uvažujúc binárny systém skladajúci sa z troch komponentov: x1,x2 a x3, z ktorých každý nadobúda dva stavy 0 – nefunkčný stav, 1 – funkčný stav. Systém funguje – stav systému 1, ak fungujú všetky 3 komponenty. Systém je nefunkčný ak aspoň jeden komponent nefunguje – stav 0.

Štrukturálnu funkciu v takomto systéme možno vyjadriť ako **f(x1,x2,x3): x1 ∧ x2 ∧ x3**, kde znak ∧ vyjadruje binárnu operáciu AND.

## MDD

Viacrozmerné rozhodovacie diagramy (MDD) sú grafickými nástrojmi na reprezentáciu vzťahov medzi premennými. Rozhodovacie diagramy poskytujú komplexný pohľad na štruktúru dát a umožňujú vizualizáciu vzťahov medzi nimi. MDD sú často používané pri analýze spoľahlivosti systémov a modelovaní, kde pomáhajú identifikovať dôležité premenné a ich vzájomné interakcie. Výhodou MDD je ich schopnosť zjednodušiť komplexné dáta a umožniť lepšie pochopenie vzorcov a trendov. Vďaka MDD je možné identifikovať kritické premenné a určiť ich vplyv na stav viacstavového systému.

Z matematického hľadiska je MDD acyklický orientovaný graf, v ktorom uzly (okrem uzlov na najnižšom leveli) predstavujú premenné a hrany jednotlivé rozhodnutia. Uzly na najnižšom leveli predstavujú výsledný stav systému na základe predchádzajúcich rozhodnutí.

Prvok, alebo komponent MDD je základnou stavebnou jednotkou v rozhodovacích diagramoch. Na základe množiny komponentov, stavov systému a vzťahmi medzi nimi je možné skonštruovať rozhodovací diagram.

### Štruktúra MDD

MDD predstavuje hierarchickú dátovú štruktúru, ktorej základom sú vrcholy a hrany. Počiatočný vrchol s levelom 0 v hierarchickej štruktúre sa nazýva koreň. Vrchol je dátová jednotka, ktorá uchováva informácie o komponente, ktorý v systéme reprezentuje, jeho level v hierarchii a doménu. Doména vrchola predstavuje možné stavy alebo rozhodnutia, ktoré môže vrchol nadobudnúť. Ku každému rozhodnutiu patrí práve jedna hrana – odkaz na ďalší vrchol. V hierarchii tento odkaz predstavuje syn vrchola. Špeciálnym typom vrchola v hierarchii je list, ten nemá doménu a na rozdiel od ostatných vrcholov reprezentuje stav systému namiesto komponentu.

Pre potreby narábania s diagramom je nutné rozlišovať medzi usporiadaným a neusporiadaným diagramom. Usporiadaný diagram je taký, v ktorom poradie vrcholov vo vetve od koreňa po list nesmie byť odlišné od poradia vrcholov v akejkoľvek inej vetve od koreňa po list. Usporiadanosť vrcholov umožňuje zdieľanie pod-stromov v rôznych vetvách a minimalizuje redundanciu v diagrame.

Ďalším aspektom pri narabáni s MDD je kompletnosť diagramu. Diagram je kompletný, ak každá hrana z každého vrchola smeruje k vrcholu s levelom o 1 vyšším.

Obrázok, na ktorom je diagram, náčrt, biely, kresba

Automaticky generovaný popisObrázok, na ktorom je diagram, biely, kresba, náčrt

Automaticky generovaný popis

Obrázok 2 Neusporiadaný, nekompletný MDD

Obrázok 3 Usporiadaný,kompletný MDD

## Indexy dôležitosti

Ako prvý popísal koncept indexov dôležitosti Zygmunt Wilhelm Birnbaum.[1] V systéme, ktorého výkon závisí na výkone jeho komponentov, môžu niektoré z týchto komponentov zohrávať dôležitejšiu úlohu ako ostatné.Napríklad, ak sa systém skladá z n komponentov v sérii alebo z n komponentov paralelne, môže sa zdať, že každý komponent sa dá považovať za rovnako dôležitý pre výkon systému. Avšak, v systéme zobrazenom na obrázku č.2 je intuitívne zjavná väčšia dôležitosť komponentu x1, než komponentov x2, x3 a x4 na stav systému.[1]

Obrázok, na ktorom je rad, diagram, snímka obrazovky, biely

Automaticky generovaný popis

Obrázok 4 Príklad jenoduchého sériovo-paralelného systému

Indexy dôležitosti sú metriky, ktoré možno použiť na zoradenie komponentov v systéme podľa rôznych perspektív. Pre správcu systému môže byť potrebné identifikovať a zoradiť najdôležitejšie komponenty v rámci systému a konkrétne určiť, kde by mala prísť investícia na zvýšenie celkovej dostupnosti systému. V inom prípade môže byť cieľom redukovať náklady na systém s čo najmenším dopadom na jeho funkčnosť. V takýchto prípadoch sú indexy dôležitosti komponentov užitočnými nástrojmi na podporu rozhodovania.

Dôležitosť komponentu by mala závisieť od nasledujúcich faktorov:

* + - * Umiestnenie komponentu v systéme.
      * Spoľahlivosť komponentu.
      * Náklady na zaopatrenie, udržanie funkčnosti komponentu.
      * Neistota v odhade spoľahlivosti komponentov a súvisiacich nákladov. [1],[2]

Indexy dôležitosti možno rozdeliť na opatrenia založené na štruktúre, na spoľahlivosti, alebo na celoživotnej dôležitosti. Podľa informácii o systéme je možné spomenuté indexy vypočítať. [1]

Známa štruktúra systému a spoľahlivosti komponentov v čase

Známa štruktúra systému a spoľahlivosti komponentov

Známa štruktúra systému

Štrukturálny index

Celoživotný index

Spoľahlivostný index

Pre potreby indexov dôležitosti je nutné zadefinovať nasledovné premenné:

* 𝑋𝑖 stav 𝑖-teho komponentu
* 𝑋 = (𝑋1, 𝑋2, … , 𝑋𝑛): stavový vektor 𝑛 komponentov

### Štrukturálny index

Štrukturálny index hodnotí dôležitosť komponentu v systéme na základe jeho pozície v štruktúre. Štrukturálny index nehodnotí spoľahlivosť komponentu ani jeho náklady.

Komponent je kritický v stavovom vektore, ak zmena jeho stavu spôsobí zmenu stavu celého systému. V binárnom systéme je štrukturálny index komponentu i vyjadrený ako:

Celkový počet stavových vektorov, v ktorých je prvok i v kritickom stave

SIi =   
 2n -1

, kde n je počet komponentov.[1]

V čisto sériovom, alebo v čisto paralelnom systéme je zbytočné vyrátavať štrukturálne indexy komponentov, nakoľko všetky prvky v systéme majú rovnakú dôležitosť. Jediné systémy, pre ktoré má zmysel zaoberať sa štrukturálnou dôležitosťou komponentov sú sériovo-paralelné systémy. V systéme zobrazenom na obrázku č.2 je možné vidieť, že sériovo zapojený prvok x1 bude pre systém dôležitejší ako paralelne zapojené prvky x2, x3 a x4.

### Spoľahlivostné indexy dôležitosti

Pre spoľahlivostné indexy dôležitosti je nutné poznať štruktúru systému, ako aj spoľahlivosti jednotlivých komponentov. Spoľahlivostný index meria mieru spoľahlivosti systému vzhľadom na spoľahlivosť jeho komponentov. Pre každý stav komponentov je charakterizovaná pravdepodobnosť, že komponent sa nachádza v danom stave.[4]

### Celoživotné indexy dôležitosti

Komponenty v systéme môžu časom degradovať, znížiť priepustnosť, alebo spôsobovať častejšie výpadky. Degradácia komponentu sa dá popísať (alebo odhadnúť) distribučnou funkciou spoľahlivosti komponentu. Celoživotné indexy dôležitosti sú spresnením spoľahlivostných indexov na základe distribučných funkcií prvkov systému.

Pre potreby celoživotných indexov dôležitosti je nutné zaviesť nasledujúce premenné:

* 𝑋𝑖 (𝑡) stav 𝑖-teho komponentu v čase 𝑡
* 𝑋(𝑡) = (𝑋1 (𝑡), 𝑋2 (𝑡), … , 𝑋𝑛 (𝑡)): stavový vektor 𝑛 komponentov v čase 𝑡
* 𝐹𝑖 (𝑡) distribučná funkcia spoľahlivosti 𝑖-teho komponentu
* 𝐹̅ 𝑖 (𝑡) = 1 − 𝐹𝑖 (𝑡) = 𝐸[𝑋𝑖 (𝑡)]: spoľahlivosť komponentu 𝑖 v čase 𝑡
* 𝐹(𝑡) = (𝐹̅ 1 (𝑡), 𝐹̅ 2 (𝑡), … , 𝐹̅ 𝑛 (𝑡)): spoľahlivostný vektor 𝑛 komponentov v čase 𝑡
* 𝑅(𝐹(𝑡)) = 𝑃{𝜙(𝑋(𝑡)) = 1} = 𝐸[𝜙(𝑋(𝑡)): spoľahlivosť systému s vektorom spoľahlivosti komponentov v čase t

### Ďalšie indexy dôležitosti

## Smerodajné parciálne logické derivácie

Smerodajné parciálne derivácie (DPLD) odhaľujú zmenu stavu systému pri zmene stavu komponentu. Pomocou derivácie je možné vyčísliť efekt zmeny stavu komponentu na stav systému a tak stanoviť dôležitosť jednotlivých komponentov. Algoritmy na výpočet logických derivácii rátajú s usporiadanosťou vrcholov v diagrame.

Univerzálny algoritmus pre výpočet DPLD sa skladá z troch krokov [3].

1. Výpočet cofactoru komponentu s pôvodným stavom.
2. Výpočet cofactoru komponentu s novým stavom.
3. Spojenie cofactorov pomocou Apply algoritmu.

Pokiaľ je cieľom derivácie určiť, či nastane po zmene stavu komponentu zmena množiny stavov systému, jedná sa o integrovanú parciálnu logickú deriváciu (IDPLD). Jej podstatou je nahradenie množiny stavov systému v pôvodnom diagrame za inú množinu.

Univerzálny algoritmus na výpočet IDPLD sa skladá z piatich krokov [3].

1. Výpočet cofactoru komponentu s pôvodným stavom.
2. Výpočet cofactoru komponentu s novým stavom.
3. Transformácia prvého cofatoru.
4. Transformácia druhého cofactoru.
5. Spojenie transformovaných cofactorov pomocou Apply algoritmu.

Výsledkom určenia logickej derivácie je nový MDD reprezentujúci logickú deriváciu. Typ lambda funkcie vstupujúcej do Apply (poprípade aj Transform) algoritmu je nutné špecifikovať podľa riešenej úlohy.

### Cofactor

Algoritmus Cofactor pracuje nad tromi parametrami: MDD, index k, hrana i. Algoritmus vytvára nový MDD na základe predchádzajúceho, pričom vrchol s indexom k nahradí za vrchol, do ktorého smerovala i-tá hrana. Nový diagram musí mať zachované listy a nesmie obsahovať žiadny vrchol s indexom k.

Obrázok, na ktorom je kreslený obrázok, náčrt, biely, animák

Automaticky generovaný popisObrázok, na ktorom je diagram, kresba, náčrt, biely

Automaticky generovaný popis

Obrázok 5 Cofactor(2i, 0, 1)

Obrázok 6 MDD pre cofactor

Základom algoritmu je rekurzívny algoritmus CofactorStep s parametrami: vrchol, index k, hrana i. Algoritmus najprv skontroluje, či vrchol má index rovný k. Ak áno, vráti i-tého syna. Pokiaľ nie, pre každého syna vrchola rekurzívne spustí funckiu CofactorStep, pričom hrana a index ostávajú rovnaké, zatiaľ čo parameter vrchol je samotný syn.

CofactorStep je najprv spustený s vrcholom diagramu a prehľadávanie sa uskutočňuje do hĺbky, čo má za následok logaritmickú zložitosť. Pre dosiahnutie lineárnej zložitosti algoritmu stačí upraviť algoritmus tak, aby si pamätal, ktoré vrcholy už boli spracované, spolu s výsledkom ich spracovania. Táto úprava zaručí jedinečné spracovanie každého vrchola.

Pseudokód algoritmu [4]:

***procedure*** *COFACTOR(diagram, i, a)*

*root ← ROOT(diagram)*

***if*** *ISTERMINAL(root)* ***then***

***return*** *diagram*

***else if*** *INDEX(root) = i* ***then***

*newRoot ← SON(root, a)*

***return*** *MDD(newRoot)*

***else***

*newRoot ← COFACTORSTEP(root, i, a)*

***return*** *MDD(newRoot)*

***end if***

***end procedure***

***procedure*** *COFACTORSTEP(node, i, a)*

***if*** *CONTAINS(memo, node)* ***then***

***return*** *LOOKUP(memo, node)*

***end if***

***if*** *ISTERMINAL(node)* ***then***

***return*** *node*

***end if***

***if*** *INDEX(node) = i* ***then***

***return*** *SON(node, a)*

***end if***

***if*** *INDEX(node) > i* ***then***

***return*** *node*

***end if***

*j ← INDEX(node)*

*sons ← MAKETUPLE(mj)*

***for*** *k = 0 to mj* ***do***

*oldSon ← SON(node, k)*

*sons[k] ← COFACTORSTEP(oldSon, i, a)*

***end for***

*newNode ← CREATEINTERNALNODE(j, sons)*

*PUT(memo, node, newNode)*

***return*** *newNode*

***end procedure***

### Apply

Algoritmus Apply vytvára nový MDD spájaním dvoch MDD pomocou lambda funkcie. Postupne prehľadáva oba diagramy, pričom spája vrchol z prvého diagramu s vrcholom z druhého diagramu.

Základom algoritmu je rekurzívny algoritmus ApplyStep, ktorý spracováva dvojice vrcholov, vracajúc nový vrchol. Prvé spustenie algoritmu ApplyStep je nad koreňmi oboch diagramov. Počas spracovávania dvoch vrcholov môžu nastať viaceré situácie, ktoré je nutné kaskádovo ošetriť:

1. Vrcholy už boli spracované:

Vráti výsledok predchádzajúceho spracovania.

1. Oba vrcholy sú listy:

Uplatní sa nad listami lambda funkcia a jej výsledok sa priradí novému listu v novom diagrame.

1. Vrcholy reprezentujú rovnaký komponent:

Pokračuje spájaním synov oboch prvkov – i-tý syn prvého vrchola s i-tým synom druhého vrchola. Týchto synov priradí novému internému vrcholu v novom diagrame.

1. Ani jedna z vyššie spomenutých:

Pokračuje spájaním synov vrchola s nižším logickým levelom s druhým vrcholom. Týchto synov priradí novému internému vrcholu v novom diagrame.

Tak isto, ako pri algoritme Cofactor, aj algoritmus Apply si pamätá už spracované vrcholy spolu s výsledkom ich spracovania. Zabezpečí tak zložitosť algoritmu rovnej O(s1 \* s2), kde s1 a s2 sú počty vrcholov v diagramoch [6].

Pseudokód algoritmu [4]:

***procedure*** *APPLYSTEP(left, right, ⊙)*

***if*** *CONTAINS(applyCache, (left, right))* ***then***

***return*** *LOOKUP(applyCache, (left, right))*

***end if***

***if*** *ISTERMINAL(left) ∧ ISTERMINAL(right)* ***then***

*node ← CREATETERMINALNODE(VALUE(left) ⊙ VALUE(right))*

***else if*** *ISABSORBINGTTERMINAL(⊙, left) ∨ ISABSORBINGTTERMINAL(⊙, right)* ***then***

*node ← CREATETERMINALNODE(ABSORBINGELEMENT(⊙))*

***else***

*ilhs ← LEVEL(left)*

*irhs ← LEVEL(right)*

*i ← min(ilhs, irhs)*

*sons ← MAKETUPLE(mi)*

***for*** *k = 0* ***to*** *mi* ***do***

***if*** *ilhs < irhs* ***then***

*sons[k] ← APPLYSTEP(SON(left, k), right)*

***else***

*sons[k] ← APPLYSTEP(left, SON(right, k))*

***end if***

***end for***

*node ← CREATEINTERNALNODE(i, sons)*

***end if***

*PUT(applyCache, (left, right), node)*

***return*** *node*

***end procedure***

***procedure*** *APPLY(left, right, ⊙)*

*root ← APPLYSTEP(ROOT(left), ROOT(right), ⊙)*

***return*** *MDD(root)*

***end procedure***

### Transform

Algoritmus Transform vytvára nový diagram pomocou lambda funkcie uplatnenej nad listami diagramu. Jeho cieľom je vytvoriť novú množinu listov na základe špecifikovanej funkcie.

Algoritmus rekurzívne prehľadáva diagram, podobne ako predchádzajúce algoritmy. Pokiaľ je aktuálne spracovávaný vrchol interný – má synov / reprezentuje komponent systému, tak tento prekopíruje do nového diagramu. Ak je spracovávaný vrchol listom, uplatní nad jeho výstupnou hodnotou lambda funkciu. Ak sa v novom diagrame nenachádza list s výstupnou hodnotou rovnou výsledku funkcie, vytvorí nový. Následne presmeruje odkazy smerujúceho do aktuálne spracovávaného vrchola na list v novom diagrame.

Pseudokód algoritmu [4]:

**procedure** TRANSFORMSTEP(*node*, *γ*)

**if** ISTERMINAL(*node*) **then**

**return** CREATETERMINALNODE(*γ*(VALUE(*node*))) **end if**

**if** CONTAINS(*memo*, *node*) **then**

**return** LOOKUP(*memo*, *node*)

**end if**

*i* ← INDEX(*node*)

*sons* ← MAKETUPLE(*mi*)

**for** *k* = 0 **to** *mi* **do**

*oldSon* ← SON(*node*, *k*)

*sons*[*k*] ← TRANSFORMSTEP(*oldSon*, *γ*)

**end for**

*newNode* ← CREATEINTERNALNODE(*j*, *sons*)

PUT(*memo*, *node*, *newNode*)

**return** *newNode*

**end procedure**

***procedure*** *TRANSFORM(diagram, γ)*

*root ← ROOT(diagram)*

*newRoot ← TRANSFORMSTEP(root, γ)*

***return*** *MDD(newRoot)*

***end procedure***

### 

# IMPLEMENTáCIA

## Výpočet štrukturálneho indexu

Pre výpočet štrukturálneho indexu dôležitosti daného komponentu v systéme pri zmene z jedného stavu do druhého je nutné vyčísliť počet zmien stavov systému, ktoré zmena stavu komponentu ovplyvní. Následne vyčísliť počet všetkých možných kombinácii stavov komponentov, pri ktorých daný komponent ovplyvňuje stav systému. Pre výpočet celkového štrukturálneho indexu dôležitosti komponentu je nutné spočítať štrukturálne indexy pre každú zmenu stavu komponentu.

Samotný výpočet indexu dôležitosti pozostáva z dvoch krokov a ku každému kroku je priradený spôsob ich výpočtu:

1. Zistiť zmeny stavov systému po zmene stavu komponentu (DPLD)
2. Priradiť zmenám váhu (Satisfy count)

Prečo je dôležité priradiť zmenám váhu? Aby bol MDD efektívny, je nutné aby obsahoval čo najmenej vrcholov. Menej prvkov v diagrame ovplyvňuje aj rýchlosť operácii vykonávaných nad diagramom. Preto niektoré vrcholy diagramu môže nahradiť ich synami. Stane sa tak, pokiaľ všetci synovia vrchola odkazujú na rovnaký vrchol, a teda stav komponentu nemá v danej vetve vplyv na výsledný stav systému. Takto vytvára neúplný diagram, v ktorom sú po ceste od vrchola k listu odstránené niektoré vrcholy. Jedna cesta od vrchola k listu preto nemusí obsahovať len jednu kombináciu všetkých komponentov a je treba počet kombinácii, ktoré daná cesta zastupuje dopočítať.

V reprezentácii MDD, ktorý by bol reprezentovaný úplnou hierarchiou, t.j. každý level diagramu predstavuje jeden komponent, pričom sú v diagrame reprezentované všetky komponenty a synovia každého vrchola odkazujú na vrcholy s levelom o jeden vyšším(ak koreň je na leveli 0), nebolo by potrebné priraďovať váhu k zmenám. Každá zmena by bola obsiahnutá v diagrame a reprezentovala by práve jednu kombináciu všetkých komponentov.

### Návrh MDD pre výpočet indexu dôležitosti

Ak sa dá zaručiť úplnosť diagramu od vytvorenia MDD a následne zachovania úplnosti diagramu až po koniec výpočtu derivácii, existuje korelácia medzi levelom komponentu v diagrame a levelom vrchola v hierarchii. V tejto variante nie je potrebné pridávať zmenám váhu a výsledný počet zmien sa rovná počtu ciest vedúcich k listu predstavujúcemu zmenu stavu systému.

Aby bol spôsob výpočtu indexov dôležitosti čo najviac univerzálny, je potrebné uvažovať o neúplných diagramoch, ako boli spomenuté vyššie. Je niekoľko možností, ako sa dá s takouto situáciou vysporiadať.

Prvý možný spôsob je pri prehľadávaní počtu zmien pamätať si komponenty po vetve od vrcholu až po list. Následne zistiť, ktoré komponenty po ceste chýbali a výsledný počet zmien, ktoré daná cesta zastupuje je rovný súčinu veľkosti domén chýbajúcich komponentov. Tento spôsob je jednoduchý na implementáciu, za cenu vyššej pamäťovej náročnosti, pretože vyžaduje presúvať zoznamy prvkov pri rekurzívnom prehľadávaní diagramu, čo môže mať za následok aj dlhší čas spracovania. Výhodou tohto prístupu je, že umožňuje prehliadať aj neusporiadané rozhodovacie diagramy.

Druhý spôsob je dopočítavať si počet zmien postupne. Keď pri prehľadávaní nastane situácia, v ktorej má syn vrchola logický level o viac ako o jeden vyšší, je zjavný chýbajúci komponent. Veľkosť domény chýbajúceho prvku sa dá nájsť podľa logického levelu, keďže každý komponent zastupuje jedinečný level v diagrame, alebo indexu prvku. Štandardné štruktúry MDD poskytujú zoznam unikátnych komponentov, preto vyhľadanie chýbajúceho komponentu nie je problém a netreba vytvárať nové zoznamy.

Pokiaľ štruktúra neposkytuje zoznam unikátnych prvkov, alebo nie je možné získať poradie prvkov, je nutné prejsť diagramom a logické levely dopočítať.

#### Získanie logických levelov

Získanie logických levelov je náročná operácia, keďže je nutné prejsť každú vetvu v diagrame, identifikovať jedinečné komponenty a každému priradiť jedinečný level. Na konci je ešte nutné prejsť jedinečné komponenty a zistiť, či niektoré nemajú rovnaký logický level. Takáto situácia nastane keď sa v diagrame na rovnakom leveli nachádzajú komponenty x1 a x2, z ktorých pokračujúce vetvy z komponentu x1 nesmerujú do druhého komponentu x2 a ani opačne. Na obrázku č.3 je jednoduchý príklad, v akom sa dvom komponenty x2 a x3 priradí rovnaký logický level.

Obrázok, na ktorom je biely, kreslený obrázok, diagram, dizajn

Automaticky generovaný popis

Obrázok 7 MDD pre získanie logických levelov

Logické levely po základnom prejdení diagramu: x1 – 0, x2 – 1, x3 – 1, listy – 2. V takomto prípade je nutné vybrať jeden komponent, ktorému logický level ostane a druhému ho zvýšiť o 1, pričom nezávisí na tom, ktorý to bude. Všetkým vrcholom s logickým vrcholom väčším ako 1 je potrebné tiež navýšiť logický level. Situácia, pri ktorej sa opakovanie logického levelu vyskytne je unikátna, ale je potrebné ju ošetriť, inak pre každý prvok bude výpočet indexu dôležitosti nesprávny.

Pre ďalšie algoritmy môže byť potrebné vedieť aj logický level listov. Logický level listov je vždy o jeden väčší ako najväčší logický level spomedzi logických levelov priradených prvkom. Najjednoduchší spôsob jeho zistenia je na úplnom konci algoritmu identifikovať ten najväčší prejdením prvkov, poprípade pri kontrole správnosti levelov a inkrementovaný následne priradiť do globálnej premennej.

Pseudokód algoritmu:

**procedure** GETLOGICALLEVELS(diagram)

root ← ROOT(diagram)

GETLOGICALLEVELSSTEP(root,0)

REPAIRLOGICALLEVELS()

**end** **procedure**

**procedure** REPAIRLOGICALLEVELS

**while** levelsOrdered = false **do**

levelsOrdered ← true

CLEAR(levelList)

**for** k = 0 to SIZE(memo) **do**

level ← LOOKUP(VALUES(memo), k)

**if** CONTAINS(levelList, level) **then**

levelsOrdered ← false

**for** i = k to SIZE(memo) **do**

**if** LOOKUP(VALUES(memo), i) >= level **then**

LOOKUP(VALUES(memo), i) ← LOOKUP(VALUES(memo), i) + 1

**end if**

**end for**

**break**

**end if**

**end for**

**end while**

**end procedure**

**procedure** GETLOGICALLEVELSSTEP(node, level)

**if** ISTERMINAL(node) **then**

**return**

**end** **if**

**if** CONTAINS(memo, INDEX(node)) **then**

**if** LOOKUP(memo, INDEX(node)) <= level **then**

LOOKUP(memo, INDEX(node)) = level

**else**

level = LOOKUP(memo, INDEX(node))

**end if**

**else**

PUT(memo, INDEX(node), level)

**end if**

LEVEL(node) = level

**for** k = 0 to mi **do**

son ← SON(node, k)

GETLOGICALLEVELSSTEP(son, level + 1)

**end for**

**end procedure**

### Použitie DPLD

Pre získanie počtu zmien stavov systému po zmene stavu komponentu je možné využiť univerzálny algoritmus pre výpočet logických derivácii. Lambda funkciu vstupujúcu do algoritmu je potrebné nastaviť tak, aby z jej výsledku bolo jasné, či vstupujúce prvky funkcie sa rovnali, alebo nerovnali. V práci bola použitá funkcia s nasledujúcou štruktúrou: .

{

0, ak x1 rovná sa x2  
f(x1,x2) =   
 1, ak x1 nerovná sa x2

Po uplatnení univerzálneho algoritmu vznikne nový MDD, v ktorom budú listy s výstupnou hodnotou „0“ odkazovať na zmenu stavu systému, a naopak listy s výstupnou hodnotou „1“ budú odkazovať na nezmenený stav systému.

### Satisfy count

Satisfy count je algoritmus, ktorý efektívne prehľadá MDD reprezentujúci logickú deriváciu a vypočíta počet zmien stavov systému tým, že priradí každej zmene stavu systému váhu – počet kombinácii stavov komponentov, ktoré zmena zastupuje. Základom algoritmu je rekurzívny algoritmus SatisfyCountStep začínajúci od vrchola, prehľadávajúci MDD do hĺbky. Na začiatku inicalizuje premennú počet, ktorá bude uchovávať počet skutočných zmien, ktoré jedna zmena v MDD zastupuje. Keď po ceste narazí na „neúplnosť“ diagramu – syn prvku je viac ako o 1 logický level vyššie ako aktuálne spracovávaný vrchol, pre každý vynechaný logický level medzi vrcholmi vynásobí premennú počet o počet stavov ktoré môže nadobúdať komponent prislúchajúci vynechanému logickému levelu. Akonáhle narazí na list, je nutné zistiť, či list reprezentuje zmenu. Algoritmus musí nadväzovať na funkciu vloženú do algoritmu Apply v predchádzajúcom kroku pri výpočte derivácie. V práci listy s výstupnou hodnotou „0“ odkazujú na zmenu, preto keď objaví takýto list, vráti počet zmien 1, ak je výstupná hodnota „1“ vráti počet zmien 0.

Pseudokód algoritmu [4]:

***Procedure*** *DOMAINPRODUCT(i1, i2)*

*product ← 1*

*i ← i1*

***while*** *i < i2 do*

*product ← product ∗ mi*

*i ← i + 1*

***end while***

***return*** *product*

***end procedure***

***procedure*** *SATISFYCOUNTSTEP(node, value)*

***if*** *ISTERMINAL(node) ∧ VALUE(node) = j* ***then***

***return*** *1*

***end******if***

***if*** *ISTERMINAL(node) ∧ VALUE(node) ≠ j* ***then***

***return*** *0*

***end if***

***if*** *CONTAINS(memo, node)* ***then***

***return*** *LOOKUP(memo, node)*

***end if***

*count ← 0*

*i ← LEVEL(node)*

***for*** *k = 0 to mi* ***do***

*son ← SON(node, k)*

*ison ← LEVEL(son)*

*sonCount ← SATISFYCOUNTSTEP(son, value)*

*diff ← DOMAINPRODUCT(i, ison)*

*count ← diff ∗ sonCount*

***end for***

*PUT(memo, node, count)*

***return*** *count*

***end procedure***

***procedure*** *SATISFYCOUNT(diagram, value)*

*root ← ROOT(diagram)*

*iroot ← INDEX(root)*

*diff ← DOMAINPRODUCT(1, iroot)*

*count ← diff ∗ SATISFYCOUNTSTEP(root, value)*

***return*** *count*

***end******procedure***

### Vylepšenie SatisfyCount

Originálny algoritmus SatisfyCount prehľadáva diagram od koreňa smerom k listom a ráta zmeny od listov smerom ku koreňu. Ak je vo výslednom diagrame známy list, ktorý reprezentuje zmeny, a existuje k nemu referencia (poprípade sa môže nájsť prejdením diagramu) je zmyselnejšie začať priamo od daného listu. Priradenie váhy nastane na koreni diagramu.

Aby si algoritmus rekurzívne neposielal počet zmien, je zmyselnejšie zaviesť si globálnu premennú, ktorá bude na začiatku algoritmu inicializovaná na 0. Keď algoritmus nájde zmenu, zväčší túto globálnu premennú. Posledným krokom algoritmu bude následne prístup ku vzniknutej globálnej premennej a jej odovzdanie. Po ceste od listu vyššie ku koreňu nie je možné spracovať rovnaký vrchol dvakrát, takže sa môže upustiť od používania tabuľky na pamätanie si už prejdených vrcholov, čo vylepší aj čas spracovania.

Pseudokód algoritmu:

**procedure** SATISFYCOUNT(diagram, value)

leaf ← LEAFWITHVALUE(value)

this.count ← 0

SATISFYCOUNTSTEP(leaf,1)

**return** this.count

**end** **procedure**

**procedure** SATISFYCOUNTSTEP(node, level, count)

newCount ← count \* DOMAINPRODUCT(LEVEL(node), level)

**if** ISROOT(node) **then**

this.count ← this.count + newCount

**return**

**end** **if**

parents *←* PARENTS(node)

**for** k = 0 to SIZE(parents) **do**

parent ← PARENT(node, k)

level ← LEVEL(node)

SATISFYCOUNTSTEP(parent, level, newCount)

**end for**

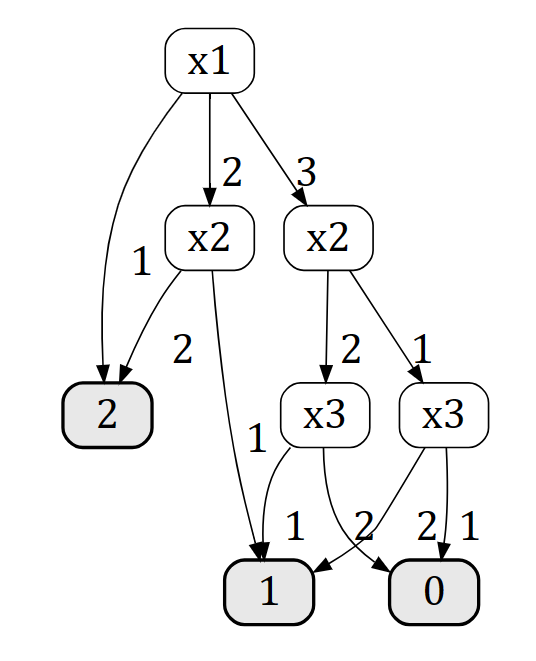
**end procedure**

# Verifikácia algoritmu

V projekte, do ktorého sa práca implementovala sa rozhodovacie diagrami vytvárajú z tabuliek. Takisto sa v ňom nachádzajú metódy na výpočet indexu dôležitosti cez tabuľku. Táto skutočnosť sa dá využiť a použiť predpripravané metódy pre verifikáciu. Nakoľko derivácie pracujú nad usporiadanými rozhodovacími diagrami, je nutné si túto skutočnosť overiť pred pokusom o výpočet indexov dôležitosti nad diagramom. Verifikovať algoritmus sa dá aj ručne, pokiaľ je k dispozícii tabuľka a diagram je dostatočne kompaktný a čitateľný. Vyznačením riadkov tabuľky alebo aplikovaním filtra sa identifikujú riadky tabuľky s jednou hodnotou komponentu i a následne riadky tabuľky s druhou hodnotou komponentu i. Porovnaním vyznačených riadkov sa ukáže počet zmien, ktorý následne treba predeliť karteziánskym súčinom domén všetkých komponentov, okrem komponentu i.

## Ručná verifikácia

Pre ručnú verifikáciu je potrebná pravdivostná tabuľka – obrázok č.12, viacrozmerný rozhodovací diagram – obrázok č.13 a zadefinovať hľadaný index.



Obrázok 8 MDD pre ručnú verifikáciu

|  |  |  |  |
| --- | --- | --- | --- |
| prvky systému | | |  |
| x1 | x2 | x3 | stav systému |
| 1 | 1 | 1 | 2 |
| 1 | 1 | 2 | 2 |
| 1 | 2 | 1 | 2 |
| 1 | 2 | 2 | 2 |
| 2 | 1 | 1 | 1 |
| 2 | 1 | 2 | 1 |
| 2 | 2 | 1 | 2 |
| 2 | 2 | 2 | 2 |
| 3 | 1 | 1 | 0 |
| 3 | 1 | 2 | 1 |
| 3 | 2 | 1 | 1 |
| 3 | 2 | 2 | 0 |
|  |  |  |  |

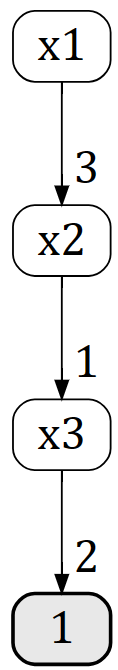
Tabuľka 1 Pravdivostná tabuľka   
pre ručnú verifikáciu

1. **krok identifikácia zmien**

Pre hľadaný štrukturálny index pre prvok s indexom 2 a zmenou stavu z 1 do 2 je potrebné nájsť prislúchajúce vetvy v diagrame a riadky v tabuľke.

**Pre diagram:**

V diagrame nastane prvá zmena po vetve x1=2, x2 pôvodne smeroval do stavu systému 1, nový stav systému je 2.

Obrázok, na ktorom je typografia

 Automaticky generovaný popis so strednou spoľahlivosťouObrázok, na ktorom je typografia

 Automaticky generovaný popis so strednou spoľahlivosťouObrázok, na ktorom je biely, dizajn, typografia

 Automaticky generovaný popis so strednou spoľahlivosťouObrázok, na ktorom je typografia

 Automaticky generovaný popis so strednou spoľahlivosťouObrázok, na ktorom je náčrt, kresba, biely, dizajn

Automaticky generovaný popis

Obrázok 9 Ručná verifikácia - tretia zmena

Obrázok 10 Ručná verifikácia - druhá zmena

Obrázok 11 Ručná verifikácia - prvá zmena

Druhá zmena po vetve x1=3, pôvodne x2=1, ďalej pre x3=1 bol stav systému 0, pre x2=2 a x3=1 je nový stav systému 1.

Tretia zmena v rovnakej vetve, ale pre x3=2 pôvodný stav systému 1, nový 0.

**Pre tabuľku:**

Pravdivostnú tabuľku stačí preusporiadať nasledujúcim spôsobom a vyznačiť zmeny:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| prvky systému | | |  |  |
| x1 | x2 | x3 | stav systému | zmena |
| 1 | 1 | 1 | 2 |  |
| 1 | 2 | 1 | 2 |  |
| 1 | 1 | 2 | 2 |  |
| 1 | 2 | 2 | 2 |  |
| 2 | 1 | 1 | 1 | 1 |
| 2 | 2 | 1 | 2 |
| 2 | 1 | 2 | 1 | 2 |
| 2 | 2 | 2 | 2 |
| 3 | 1 | 1 | 0 | 3 |
| 3 | 2 | 1 | 1 |
| 3 | 1 | 2 | 1 | 4 |
| 3 | 2 | 2 | 0 |
|  |  |  |  |  |

1. **krok porovnanie**

V tabuľke sa nachádzajú 4 zmeny, v diagrame sa nachádzajú 3, ale zmena č.3 zastupuje 2 celkové zmeny vďaka neúplnosti diagramu. Keby bol diagram úplný, na vetve pod x2 by sa nachádzali dva vrcholy predstavujúce komponent x3, ktoré by obe smerovali do stavu 2.

Celkový počet stavových vektorov, ktoré prvok x2 ovplyvňuje je 6 (súčin domén prvkov okrem prvku x2), preto SIi=2(1→2) je rovný 2/3.

## Automatická verifikácia

Pre automatickú verifikáciu je nutné rovnako správne uplatniť algoritmy nad tabuľkou a rozhodovacím diagramom. V tejto časti práce je použitý výpočet celkového štruktúrneho indexu pre každý komponent v systéme.

# Porovnanie rýchlosti výpočtu

Jedným z dôvodov, prečo je vhodné na výpočet indexov dôležitosti použiť rozhodovací diagram namiesto pravdivostnej tabuľky je rýchlosť výpočtu. Pre porovnanie sa dá opäť použiť algoritmus pracujúci nad tabuľkou, ktorý bol použitý aj na verifikáciu algoritmu.

Pre porovnanie rýchlosti výpočtu bol použitý notebook hp pavilion gaming 15 s štvor-jadrovým procesorom Intel(R) Core(TM) i5-10300H. Oba algoritmy sú aplikované na rovnakých systémoch, pričom je meraný celkový čas výpočtu indexov dôležitosti pre všetky prvky systému a pre všetky zmeny stavov prvku systému. Výsledné hodnoty sú priemerom nameraných hodnôt pri 10 spusteniach oboch algoritmov.

|  |  |  |
| --- | --- | --- |
| Počet prvkov | Rýchlosť [ms] | |
| 10 | Pomcou MDD | Pomocou tabuľky |
| 100 |  |  |
| 1000 |  |  |
| 10000 |  |  |
| 20000 |  |  |

Tabuľka 2 Porovnanie rýchlosti výpočtu SI pomocou tabuľky / MDD

Algoritmus pracujúci nad MDD je podľa meraní signifikantne rýchlejší s nárastom počtu komponentov v systéme.

Záver

Cieľom práce bolo navrhnúť a implementovať algoritmus na výpočet štrukturálnych indexov dôležitosti vo viacstavovom systéme reprezentovanom viacrozmerným rozhodovacím diagramom.

Výsledkom práce je funkčná implementácia algoritmu na výpočet štrukturálneho indexu dôležitosti pre prvky vo viacstavovom systéme. Pre výpočet indexov bol systém reprezentovaný viacrozmerným rozhodovacím diagramom. Samotný algoritmus využíva na výpočet smerodajné parciálne logické derivácie.

Cieľom práce bolo navrhnúť taký algoritmus s efektívnou zložitosťou výpočtu a čo najmenšou časovou náročnosťou. Viacrozmerné rozhodovacie diagramy sú časovo efektívnejšiou reprezentáciou systému pre výpočet indexov dôležitosti ako pravdivostná tabuľka.

Pri práci bolo nájdené obmedzenie výpočtu – usporiadanosť vstupného rozhodovacieho diagramu. Toto obmedzenie pochádza z obmedzenia pod-algoritmov, ktoré sú na výsledný výpočet použité – konkrétne algoritmy Apply a SatisfyCount. Algoritmus SatisfyCount je možné upraviť tak, aby vkladaný rozhodovací diagram nemusel byť usporiadaný, čo je spomenuté v kapitole [2.1.1](#_Návrh_MDD_pre) . Problémom ostáva algoritmus Apply, keďže zlučuje dva diagramy do jedného, pričom ráta práve s usporiadanosťou prvkov. Odstrániť celkové obmedzenie bez poškodenia zložitostí algoritmov môže byť námetom na pokračujúcu výskumnú úlohu.

Zoznam použitej literatúry

|  |  |
| --- | --- |
| [1] | Birnbaum, Z. W. 1969. On the importance of different components in a multicomponent system, 2.vid. Academic Press, New York, USA, s. 581–592. |
| [2] | Mrena, M. – Kvassay, M.2024. Efficient Computation of Logic Derivatives Using Multi-valued Decision Diagrams. |
| [3] | Wu, S. – Coolen, F. 2022. Importance Measures in Reliability Engineering: An Introductory Overview. V: Salhi, S., Boylan, J. (eds) The Palgrave Handbook of Operations Research . Palgrave Macmillan.  Dostupné na: <https://doi.org/10.1007/978-3-030-96935-6_19>. |
| [4] | Kuo, W. – Zhu, X. 2012. Some Recent Advances on Importance Measures in Reliability. In IEEE Transactions on Reliability, 2012, roč. 61, č. 2, s. 344-360 doi: 10.1109/TR.2012.2194196.  Dostupné na: https://ieeexplore.ieee.org/document/6198317 |
| [5] | Bryant, R. E. Graph-Based Algorithms for Boolean Function Manipulation. V: IEEE Transactions on Computers,1986 vol. C-35, č. 8, s. 677-691, doi: 10.1109/TC.1986.1676819. Dostupné na: https://ieeexplore.ieee.org/document/1676819 |
| [6] | Zhang, B. – Liu Y. – Xiahou, T. Importance Measure for Multilevel Inspections of Multistate Systems: A Value of Information Perspective. V: IEEE Transactions on Reliability, doi: 10.1109/TR.2023.3312688 Dostupné na: https://ieeexplore.ieee.org/document/10255370 |
| [7] | Kostolny, J. – Kvassay, M. – Zaitseva, E. Analysis of Algorithms for Computation of Direct Partial Logic Derivatives in Multiple-Valued Decision Diagrams. 2014 Ninth International Conference on Availability, Reliability and Security, Fribourg, Switzerland, 2014, s. 356-361, doi: 10.1109/ARES.2014.54. Dostupné na: https://ieeexplore.ieee.org/document/6980303 |
|  |  |

**Príloky**

Zoznam príloh

[Príloha A | Obsah DVD 36](#_Toc164725472)

1. Obsah DVD

Priložené DVD obsahuje:

* Prácu v elektronickej podobe (formát PDF)
* Zdrojový kód aplikáci