



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**

FACULTY OF INFORMATION TECHNOLOGY

**ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ**

DEPARTMENT OF COMPUTER SYSTEMS

**KNIHOVNA PRO BOOLOVSKÉ FUNKCE V ALGEBRAICKÉ  
NORMÁLNÍ FORMĚ**

LIBRARY FOR BOOLEAN FUNCTIONS IN ALGEBRAIC NORMAL FORM

**BAKALÁŘSKÁ PRÁCE**

BACHELOR'S THESIS

**AUTOR PRÁCE**

AUTHOR

**MAROŠ VASILIŠIN**

**VEDOUCÍ PRÁCE**

SUPERVISOR

**Ing. ROLAND DOBAI, Ph.D.**

**BRNO 2017**

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav počítačových systémů

Akademický rok 2016/2017

**Zadání bakalářské práce**

Řešitel: **Vasilišin Maroš**

Obor: Informační technologie

Téma: **Knihovna pro boolovské funkce v algebraické normální formě**

**Library for Boolean Functions in Algebraic Normal Form**

Kategorie: Algoritmy a datové struktury

**Pokyny:**

1. Seznamte se s reprezentací boolovských funkcí v algebraické normální formě (ANF).
2. Analyzujte knihovny pro boolovské funkce, zejména ty, které jsou založeny na binárních rozhodovacích diagramech.
3. Navrhněte a implementujte knihovnu pro manipulace boolovských funkcí v ANF.
4. Ověřte funkčnost implementované metody na příkladě vytvoření paralelní obvodové struktury ze sekvenčního nelineárního zpětnovazebního posuvného registru.
5. Zhodnoťte dosažené výsledky.

**Literatura:**

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 a 2 zadání, demonstrace rozpracovanosti bodu 3 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese

<http://www.fit.vutbr.cz/info/szz/>

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Dobai Roland, Ing., Ph.D., UPSY FIT VUT**

Datum zadání: 1. listopadu 2016

Datum odevzdání: 17. května 2017

**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

Fakulta informačních technologií

Ústav počítačových systémů a sítí

602 00 Brno, Božetěchova 2



prof. Ing. Lukáš Sekanina, Ph.D.  
vedoucí ústavu

## Abstrakt

Táto bakalárska práca sa zaoberá návrhom a implementáciou knižnice v jazyku C pre manipuláciu Boolových funkcií v Algebraickej Normálnej Forme. Väčšina existujúcich reprezentácií Boolových funkcií je založená na binárnych rozhodovacích diagramoch. Algebraická Normálna Forma poskytuje oproti binárnym rozhodovacím diagramom určité výhody, napríklad tú, že sa dá z nej v lineárnom čase určiť Boolova hodnota funkcie. Implementovaná knižnica za pomoci jednoduchých štruktúr poskytuje efektívnu reprezentáciu Boolovej funkcie v programe. Výskumom sme zistili, že reprezentácia pomocou Algebraickej Normálnej Formy má svoj využitie, a v určitých prípadoch dosahuje lepšie výsledky ako reprezentácia pomocou binárnych rozhodovacích diagramov.

## Abstract

This bachelor thesis focuses on design and implementation of library in C language for manipulation of Boolean functions in Algebraic Normal Form. Majority of existing libraries for representation of Boolean functions is based on binary decision diagrams. Algebraic Normal Form presents several advantages over binary decision diagrams, for example Boolean value of function can be determined in linear time. Implemented library uses simple structures to effectively represent Boolean function in program. After experiments we determined that representation in Algebraic Normal Form has its applications, and in some cases it provides better results than representation in binary decision diagrams.

## Klíčové slová

Boolova funkcia, Boolova algebra, normálna forma, algebraická normálna forma, binárny rozhodovací diagram, logické operácie, C

## Keywords

Boolean function, Boolean Algebra, normal form, algebraic normal form, binary decision diagram, logical operations, C

## Citácia

VASILÍŠIN, Maroš. *Knihovna pro Boolovské funkce v algebraické normální formě*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Roland Dobai, Ph.D.

# Knihovna pro Boolovské funkce v algebraické normální formě

## Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Rolanda Dobaia, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....

Maroš Vasilišin

17. mája 2017

## Podakovanie

Ďakujem pánovi Ing. Rolandovi Dobaiovi, Ph.D za cenné rady, odborné vedenie a dohľad nad touto bakalárskou prácou.

# Obsah

<b>1</b>	<b>Úvod</b>	<b>3</b>
<b>2</b>	<b>Boolove funkcie a ich reprezentácia</b>	<b>4</b>
2.1	Definícia Boolovej funkcie . . . . .	4
2.2	Spôsoby reprezentácie Boolových funkcií . . . . .	6
2.3	Normálne formy Boolových funkcií . . . . .	7
2.4	Algebraická normálna forma Boolových funkcií . . . . .	7
2.4.1	Splniteľnosť Boolových funkcií . . . . .	8
2.4.2	Konverzie medzi normálnymi formami . . . . .	9
2.5	Binárne rozhodovacie diagramy . . . . .	9
<b>3</b>	<b>Knižnice pre reprezentáciu Boolových funkcií</b>	<b>12</b>
3.1	Colorado University Decision Diagram Package - CUDD . . . . .	12
3.2	CacBDD . . . . .	12
3.3	BuDDy . . . . .	13
3.4	BCL - Class Library for Boolean Function Manipulation . . . . .	13
3.5	CORAL . . . . .	13
3.6	BDD . . . . .	14
<b>4</b>	<b>Konceptuálny návrh knižnice</b>	<b>15</b>
4.1	Voľba technológií pre vývoj knižnice . . . . .	15
4.2	Reprezentácia premenných . . . . .	15
4.3	Hashovacia tabuľka premenných funkcie . . . . .	16
4.4	Reprezentácia termov vo funkcii . . . . .	17
4.5	Reprezentácia Boolovej funkcie . . . . .	18
4.6	Optimalizácia a minimalizácia . . . . .	19
4.7	Grafické zobrazenie funkcií . . . . .	20
<b>5</b>	<b>Implementácia navrhovaného riešenia a použité algoritmy</b>	<b>22</b>
5.1	Počítanie hodnoty termu . . . . .	23
5.2	Možnosti práce s Boolovými funkciami . . . . .	23
5.3	Grafická reprezentácia štruktúr . . . . .	25
<b>6</b>	<b>Porovnanie knižnice s existujúcimi riešeniami</b>	<b>28</b>
6.1	Experiment 1 - Boolova funkcia malého rozsahu . . . . .	28
6.2	Experiment 2 - Boolova funkcia s vysokým počtom premenných . . . . .	29
<b>7</b>	<b>Záver</b>	<b>31</b>

<b>Literatúra</b>	<b>32</b>
<b>A Obsah CD</b>	<b>34</b>

# Kapitola 1

## Úvod

Boolova algebra má značné využitie vo viacerých oblastiach vedy a techniky. Jej základným a dnes hlavným využitím je binárna reprezentácia stavov tranzistorov v počítačoch, a tým pádom využitie jediného bitu pre reprezentáciu informácie. Okrem toho ale svoje využitie nachádza aj pri návrhu číslicových obvodov ako efektívna reprezentácia správania sa jednotlivých hardvérových komponentov.

Bohaté využitie má aj v matematike, konkrétne vo výrokovej logike či kombinatorike. Uplatnenie Boolovej algebry je možné vidieť aj v oblastiach umelej inteligencie, teórie mechanického učenia a teórie hier. Z netechnických odborov stojí za zmienku oblasť legislatívy, kde sa využíva Boolova logika napríklad pri voľbách do štátnych funkcií (výber z dvoch možných kandidátov).

Existujú [4] viaceré reprezentácie Boolových funkcií, ktoré sa líšia svojim použitím. Klasické reprezentácie formou pravdivostných tabuliek nachádzajú svoje využitie v matematike, ale v informatike sa ukázali ako nevhodné. V priebehu času boli vytvorené rôzne metódy pre ich symbolizáciu v počítačovom programe. Najrozšírenejšia z nich je reprezentácia binárnymi rozhodovacími diagramami (skrátene BDD z anglického Binary Decision Diagram). Jednou z výhod reprezentácie pomocou BDD je fakt, že pomocou BDD je možné vytvoriť kanonickú formu funkcie, teda formu, ktorá je jedinečná pre každú Boolovu funkciu. BDD umožňujú veľmi dobre zisťovať ekvivalenciu a splniteľnosť Boolových funkcií.

Reprezentácia pomocou BDD je v informatike síce najrozšírenejšia, ale nie je jediná. Táto práca sa zaoberá podrobnejšie reprezentáciou Boolových funkcií pomocou Algebraickej Normálnej Formy (skrátene ANF). ANF poskytuje výhodu oproti BDD v tom, že obsahuje len dve logické operácie, logický súčin a exkluzívny súčet, a tým pádom sa jej implementácia značne zjednodušuje. Takisto je z ANF veľmi rýchlo možné zistiť Boolovu hodnotu danej funkcie a takisto vypočítať jej splniteľnosť [11].

Cieľom bakalárskej práce bolo navrhnúť a implementovať knižnicu pre efektívnu manipuláciu s Boolovými funkciami v Algebraickej Normálnej Forme. Druhotným cieľom je zistenie a analýza vytvorenej knižnice, a jej porovnanie s existujúcimi riešeniami. Implementovaná knižnica poskytuje prostriedky pre vytváranie a manipuláciu s Boolovými funkciami. Využíva výhody, ktoré poskytuje ANF a poskytuje alternatívu voči klasickým reprezentáciám založeným na BDD.

V kapitole 2 si povieme najskôr niečo o rôznych reprezentáciách Boolových funkcií, o ich výhodách a nevýhodách. V kapitole 3 si popíšeme existujúce knižnice a ich využitie v praxi. V kapitole 4 sa budeme zaoberať technickým návrhom knižnice, v kapitole 5 jej konkrétnou implementáciou. Na záver si v kapitole 6 porovnáme knižnicu s existujúcimi riešeniami a vyvodíme z toho závery.

## Kapitola 2

# Boolove funkcie a ich reprezentácia

Boolove funkcie sú také funkcie, ktorých výsledkom je logická nula alebo logická jednotka. Využívajú sa na mnohých miestach v počítačovej vede, napríklad pri vyhodnocovaní podmienok v programe alebo ako návratový typ nejakej funkcie alebo testu, ktorý určuje či funkcia alebo test prebieha ako sa očakáva. Ich primárnym účelom je popísať stav alebo jav len dvomi hodnotami, ktoré sú si navzájom opačné, čím programátor získava ďalšiu možnosť práce s informáciami. Existuje viacero možných reprezentácií Boolových funkcií, napríklad pravdivostné tabuľky alebo Karnaughove mapy. Okrem toho existuje aj viacero komplexnejších reprezentácií, ako je napríklad Algebraická Normálna Forma či binárne rozhodovacie diagramy.

### 2.1 Definícia Boolovej funkcie

Ako uvádza Crama [8], Boolova funkcia je každá funkcia  $f : \mathcal{B}^n \rightarrow \mathcal{B}$ , kde  $\mathcal{B}$  je množina  $\{0, 1\}$ , v ktorej  $n$  je kladné prirodzené číslo, a  $\mathcal{B}^n$  označuje  $n$ -násobný kartézsky súčin množiny  $\mathcal{B}$  samej so sebou. Každý bod funkcie  $X^* = (x_1, x_2, \dots, x_n)$  nabera hodnotu buď logická 0 alebo logická 1 z množiny  $\mathcal{B}$ .

Celkový počet rôznych Boolových funkcií pre  $n$  premenných je  $2^{2^n}$ . Je to dané tým, že všetkých možných kombinácií vstupných parametrov je  $(2^n)$  a parametre môžu mať hodnotu z množiny  $\{0, 1\}$ . Tento počet obsahuje aj kombináciu o 0 prvkoch, ktorá ale pre nás nemá využitie, takže sa častejšie uvádza číslo  $2^{2^n-1}$ . Počet možných Boolových funkcií pre niektoré hodnoty  $n$  sa nachádza v Tabuľke 2.1.

Tabuľka 2.1: Počet všetkých Boolových funkcií pre vybrané hodnoty  $n$ , kde  $n$  označuje počet premenných.

n	počet funkcií
1	4
2	16
3	256
5	$4.29497 \times 10^9$
6	$1.84467 \times 10^{19}$

Je vidieť že počet možných kombinácií prudko narastá s počtom premenných, a teda efektívna reprezentácia je nutnosťou.



V mnohých aplikáciách sa pre predstavu hodnôt množiny  $\mathcal{B}$  namiesto dvojice  $\{0,1\}$  používa iná dvojica, napríklad  $\{\text{true}, \text{false}\}$ ,  $\{1, -1\}$ ,  $\{\text{on}, \text{off}\}$ ,  $\{\text{áno}, \text{nie}\}$ , vždy to ale označuje navzájom opačné hodnoty.

Množina  $\mathcal{B}$  spolu so základnými Boolovymi operáciami konjunkciou  $\wedge$ , disjunkciou  $\vee$  a negáciou  $\neg$  tvorí Boolovu algebru. Tieto tri operácie majú podobne ako dvojica  $\{0,1\}$  viacero používaných zápisov, napríklad  $\{\cap, \cup, -\}$  alebo  $\{+, \cdot, -\}$  [12].

Boolovu algebru tvorí niekoľko základných pravidiel, ktoré sú popísané v Tabuľke 2.2.

Tabuľka 2.2: Pravidlá Boolovej algebry,  $x_1, x_2, x_3$  označujú navzájom rôzne premenné v Boolovej funkcii,  $\wedge, \vee, \neg$  označujú operácie konjunkciu, disjunkciu a negáciu.

názov pravidla	znenie pravidla
asociatívnosť	$(x_1 \vee x_2) \vee x_3 = x_1 \vee (x_2 \vee x_3)$ $(x_1 \wedge x_2) \wedge x_3 = x_1 \wedge (x_2 \wedge x_3)$
komutatívnosť	$x_1 \vee x_2 = x_2 \vee x_1$ $x_1 \wedge x_2 = x_2 \wedge x_1$
absorpcia	$x_1 \vee (x_1 \wedge x_2) = x_1$ $x_1 \wedge (x_1 \vee x_2) = x_1$
distributívnosť	$x_1 \vee (x_2 \wedge x_3) = (x_1 \vee x_2) \wedge (x_1 \vee x_3)$ $x_1 \wedge (x_2 \vee x_3) = x_1 \wedge x_2 \vee x_1 \wedge x_3$
komplementárnosť	$x_1 \vee \neg x_1 = 1$ $x_1 \wedge \neg x_1 = 0$
agresivita nuly	$x_1 \wedge 0 = 0$
agresivita jednotky	$x_1 \vee 1 = 1$
idempotencia	$x_1 \vee x_1 = x_1$ $x_1 \wedge x_1 = x_1$
absorpcia negácie	$x_1 \vee (\neg x_1 \wedge x_2) = x_1 \vee x_2$ $x_1 \wedge (\neg x_1 \vee x_2) = x_1 \wedge x_2$
dvojitá negácia	$\neg(\neg x_1) = x_1$
De Morganove zákony	$\neg x_1 \wedge \neg x_2 = \neg(x_1 \vee x_2)$ $\neg x_1 \vee \neg x_2 = \neg(x_1 \wedge x_2)$

Tieto pravidlá sa využívajú pri zjednodušovaní Boolových funkcií a pri zisťovaní ekvivalencie. Ako píše Comerford [6], Boolove funkcie sú ekvivalentné ak existuje súbor afínnych a lineárnych transformácií, po ktorých vykonaní budú funkcie zhodné.

Operáciou, ktorá nepatrí do trojice základných Boolových operácií, ale v programovaní má svoje veľké využitie je operácia exkluzívneho súčtu, ktorý sa v literatúre označuje aj ako XOR. Je možné ho vytvoriť kombináciou ostatných operácií, napríklad tak, ako ukazuje Rovnica 2.1. V tejto rovnici je operácia exkluzívny súčet označená symbolom  $\oplus$ . Využíva sa napríklad pri konštrukcii obvodov alebo v generátoroch pseudonáhodných čísel.

$$x_1 \oplus x_2 = (x_1 \wedge \neg x_2) \vee (x_2 \wedge \neg x_1) \quad (2.1)$$

## 2.2 Spôsoby reprezentácie Boolových funkcií

Boolove funkcie môžu byť vyjadrené rôznymi spôsobmi. Záleží hlavne na tom, na čo bude daná funkcia využitá, a aké operácie s ňou budú vykonávané. Niektoré zápisy sú vhodnejšie na matematické výpočty, iné zase na prehľadné prezeranie dát.

Prvým možným zápisom je pravdivostná tabuľka. Je to tabuľka, v ktorej na každom riadku je hodnota funkcie pre inú kombináciu vstupných hodnôt. Pravdivostné tabuľky majú dobré využitie pre funkcie do 3–4 parametrov. Pre vyšší počet parametrov sa stávajú neprehľadnými pre vysoký počet možných kombinácií. Príklad pravdivostnej tabuľky pre dve vstupné hodnoty sa nachádza v Tabuľke 2.3.

Tabuľka 2.3: Príklad pravdivostnej tabuľky pre dve vstupné premenné  $x_1, x_2$ .

$(x_1, x_2)$	$f(x_1, x_2)$
(0, 0)	0
(0, 1)	1
(1, 0)	1
(1, 1)	0

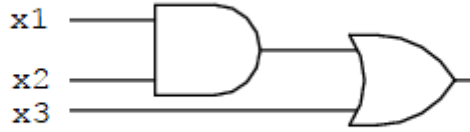
Upravenou formou pravdivostnej tabuľky je Karnaughova mapa. Je to forma zápisu, ktorá prevádza  $n$ -rozmernú Boolovu funkciu do dvojrozmernej. Jej výhodou je, že sa pomocou nej dá funkcia pekne vizualizovať, do 5 premenných poskytuje stále dobrú predstavu. Využíva sa hlavne pri minimalizácii funkcií. Je vhodná pre ľudskú predstavu funkcie, pre počítač existujú efektívnejšie alternatívy. Príklad Karnaughovej mapy sa nachádza na Obr. 2.1, zobrazuje Karnaughovu mapu pre funkciu  $f = x_2 \vee (x_1 \wedge \neg x_3)$ .

$x_1 x_2$		00	01	11	10
$x_3$	1	0	1	1	0
	0	0	1	1	1

Obr. 2.1: Príklad Karnaughovej mapy pre funkciu  $f = x_2 \vee (x_1 \wedge \neg x_3)$ .

Ďalším zo zápisov je logický obvod. Ide o schému, ktorá graficky zobrazuje Boolovu funkciu. Tento zápis je vhodnejší pre fyzikálne zamerané úlohy, alebo pre pokročilejšie úlohy, ktoré obsahujú zložitejšie funkcie, a tie sa dajú prehľadne zobrazit logickým obvodom. Logický obvod na rozdiel od predošlých reprezentácií neukazuje všetky možné kombinácie hodnôt, ale len štruktúru danej funkcie. Dá sa prehľadne použiť aj pre reprezentáciu funkcie o väčšom množstve premenných, čo ostatné reprezentácie nedokážu. Príklad zobrazenia funkcie  $(x_1 \wedge x_2) \vee x_3$  vidíme na Obr. 2.2.

V technických odvetviach sa využívajú určité štandardné výrazy, ktoré sa dajú dobre využiť pri vytváraní kombinačných obvodov. Tieto výrazy sa nazývajú normálne formy a existuje ich niekoľko. Rôznymi typmi normálnych foriem sa zaoberá podkapitola 2.3.



Obr. 2.2: Príklad logického obvodu funkcie  $(x_1 \wedge x_2) \vee x_3$ .

Pre strojovú reprezentáciu Boolových funkcií sa ukázali vhodné aj binárne rozhodovacie diagramy (BDD) a ich rôzne modifikácie, bude im venovaná samostatná podkapitola 2.5.

## 2.3 Normálne formy Boolových funkcií

Normálna forma je každý výraz v tvare:

$$T_1 \text{ op } T_2 \text{ op } T_3 \text{ op } \dots \text{ op } T_n$$

kde množina  $\{T_1, T_2, T_3 \dots T_n\}$  sú navzájom rôzne termy rovnakého typu a  $op$  je operácia v Boolovej algebre. Podľa typu termov a typu operácie poznáme niekoľko základných normálnych foriem [10].

- Disjunktívna (DNF) - termy sú konjunkciou premenných a operáciou je disjunkcia.
- Konjunktívna (CNF) - termy sú disjunkciou premenných a operáciou je konjunkcia.

Ak sa v každom terme v spomenutých normálnych formách vyskytuje premenná práve raz, tieto normálne formy nazývame úplná disjunktívna/konjunktívna normálna forma. Ak vynecháme redundantné členy, nazývame ich iredundantné normálne formy.

## 2.4 Algebraická normálna forma Boolových funkcií

Algebraická normálna forma (skrátene ANF) je jeden z možných spôsobov reprezentácie Boolových funkcií. Ďalším používaným označením pre ANF je Reed-Mullerova expanzia [15, 16]. Dnešné vedomosti o ANF pomáhali formovať aj Davio [9] a Zhegalkin [19]. Je to jeden z najpoužívanejších spôsobov reprezentácie v kryptografií. Podľa definície z knihy *Boolean Functions and Their Applications in Cryptography* [18] od Wu a Fenga, je funkcia v ANF, ak je napísaná vo forme ako ukazuje Rovnica 2.2, kde  $f(x)$  je daná funkcia,  $c_0, c_i, c_{ij}, \dots, c_{1,\dots,n}$  sú koeficienty o hodnote z množiny  $\{0, 1\}$  a  $\oplus$  reprezentuje operáciu exkluzívny súčet (XOR).

$$f(x) = c_0 \bigoplus_{1 \leq i \leq n} c_i x_i \bigoplus_{1 \leq i < j \leq n} c_{ij} x_i x_j \bigoplus \dots \bigoplus c_{1,\dots,n} x_1 x_2 \dots x_n \quad (2.2)$$

Matematicky je dokázané, že pre každú Boolovu funkciu s danými konkrétnymi koeficientami sa dá vytvoriť unikátna ANF.

Celá ANF má taktiež hodnotu z množiny  $\{0, 1\}$ . Jednotlivé výrazy medzi operátormi XOR nazývame termy. Termy v ANF vytvárame buď kombináciou premenných spojených operáciou logickú súčin (AND) a vynásobením koeficientami, prípadne to môže byť jeden samostatný koeficient, ak sa v terme premenná nevyskytuje. Príklad ANF môžeme vidieť v Rovnici 2.3. Ako vidíme, ANF sa skladá len z kombinácie operácií AND a XOR, žiadna ďalšia Boolova operácia nie je povolená. Špecificky je dobré spomenúť operáciu negácia (NOT), ktorá sa bežne vyskytuje v ostatných normálnych formách ako sú DNF a CNF, ale v ANF ju nevidíme.

$$1 \oplus x_1 \oplus x_2 \oplus x_1x_2 \oplus x_1x_2x_3 \quad (2.3)$$

Počet premenných jedného termu sa nazýva algebraický stupeň termu [18]. Celkový algebraický stupeň celej ANF je stupeň termu s najvyššou hodnotou z danej ANF, ale berú sa len termy s nenulovými koeficientami. Používaná notácia pre algebraický stupeň funkcie je  $\deg(f)$ . Najvyšší možný stupeň Boolovej funkcie o  $n$  premenných je  $n$ , a to len vtedy, ak sa v ANF nachádza term, ktorý obsahuje všetkých  $n$  premenných.

Algebraický stupeň funkcie sa používa na určenie typu funkcie. Ak je stupeň nulový, funkcia je konštantná (neobsahuje žiadne premenné). Ak je stupeň 1, funkciu nazývame afínnou, a existuje ešte prípad, ak máme afínnu funkciu bez konštantného termu  $c_0$  z definície 2.2, vtedy funkciu nazývame lineárnou. Lineárna funkcia teda prechádza bodom  $[0,0]$ , afínná týmto bodom prechádzať nemusí. Afínná Boolova funkcia môže byť lineárna funkcia, alebo vo forme exkluzívneho súčtu lineárnej funkcie a konštanty logická 1, čo je vlastne znova len daná lineárna funkcia, ak dodržíme pravidlá Boolovej algebry. Takže obe varianty sa vlastne môžu považovať za lineárne funkcie.

Z programátorského pohľadu môžeme hodnotu každého termu reprezentovať ako integer modulo 2. Každý term je jednoduchým polynómom, ktorý v sebe neobsahuje koeficienty ani exponenty. Koeficienty nepotrebujeme, pretože 1 je jediný nenulový koeficient. Exponenty nie sú potrebné z dôvodu, že každá individuálna premenná v ANF má algebraický stupeň najviac 1, keďže platí, že  $x^n = x$ , v nezávislosti na tom, či  $x = 1$  alebo  $x = 0$ . Preto napríklad aj zložitejší polynóm ako  $3^x2^y5^z$  môžeme prepísať na  $xyz$  a jednoducho ho reprezentovať v programe.

Pomocou operácií logického súčinu  $\wedge$  a negácie  $\neg$  dokážeme vytvoriť všetky ostatné operácie v Boolovej algebre. Ďalšie operácie sú tvorené len kombináciou týchto dvoch operácií. Keďže v ANF nie je povolená operácia negácia, musíme si ju nejako vytvoriť, ak chceme reprezentovať aj opačné hodnoty k premenným. Negácia v ANF vzniká vykonaním operácie exkluzívneho súčtu nad premennou a logickou jednotkou:  $x \oplus 1$ . Týmto spôsobom dokážeme previesť do ANF aj funkcie z iných normálnych foriem, prípadne aj z iných reprezentácií.

### 2.4.1 Splniteľnosť Boolových funkcií

Problém splniteľnosti Boolových funkcií (z anglického Boolean satisfiability problem, skratka SAT) sa zaoberá tým, či existuje taká kombinácia premenných v Boolovej funkcii, ktorým by sa priradili hodnoty logická 0 a logická 1, a výsledná funkcia by sa vyhodnotila ako logická 1.

Ak takáto kombinácia premenných existuje, funkciu nazývame *splniteľnou*. Naopak, ak neexistuje žiadna kombinácia premenných, pre ktoré by funkcia mala hodnotu logická 1, funkciu nazývame *nesplniteľnou*. Typickým príkladom nesplniteľnej funkcie môže byť fun-

kcia v rovnici 2.4, keďže nie je možné, aby premenná mala zároveň hodnotu logickej 0 a logickej 1.

$$f = x_1 \wedge \neg x_1 \quad (2.4)$$

Dnes existujú viaceré algoritmy (tiež nazývané v literatúre SAT solvery), ktoré riešia rôzne druhy SAT problémov, napríklad z oblasti umelej inteligencie či tvorby logických obvodov.

Ako vstup pre SAT solver sa bežne používa konjunktívna normálna forma (CNF), ale existujú aj výnimky, ktoré využívajú napríklad DNF. CNF sa primárne využíva z dôvodu, že každá Boolova funkcia sa dá prepísať do formy CNF v lineárnom čase. Výsledná funkcia v CNF si zachováva splniteľnosť z pôvodnej formy. Je preto dôležité zamerať sa na konverziu ANF na CNF, prípadne ďalšie konverzie. Využitie SAT solverov pre prácu s ANF je možné len vtedy, ak je možné konvertovať danú ANF na CNF, ktorá by bola vstupom pre SAT solver pri zachovaní splniteľnosti.

Ak je Boolova funkcia zapísaná vo forme Algebraickej Normálnej Formy, môžeme v nej vidieť dve časti, na ktoré sa vzťahuje SAT problém. Pre jednotlivé termy, ktoré obsahujú len operáciu logický súčin (sú teda v disjunktívnej normálnej forme), je zistenie riešenia SAT problému triviálne. Ak majú všetky premenné hodnotu logická 1, je daný term splniteľný, ak aspoň jedna premenná má hodnotu logickej 0, je daný term nespĺniteľný.

Druhým SAT problémom ANF sú klauzuly exkluzívneho súčtu XOR medzi jednotlivými termami. Keďže funkcia obsahujúca tieto klauzuly sa dá prepísať ako systém lineárnych rovníc modulo 2, je možné tento SAT problém vyriešiť v kubickom čase pomocou Gaussovej eliminácie [14].

### 2.4.2 Konverzie medzi normálnymi formami

Konverzia z ANF na CNF sa skladá z dvoch krokov [2, 7]:

1. Každý term rovnice, ktorý má váhu väčšiu ako 1, sa premení na systém CNF klauzúl, ktoré vzniknú ako ekvivalent daného termu a budú reprezentované pomocnou premennou vo väčšom lineárnom systéme.
2. Tento lineárny systém sa nakoniec prevedie do ekvivalentného systému v CNF forme.

Menším obmedzením je, že CNF neobsahuje žiadne konštanty, na rozdiel od ANF. Ak chceme teda pridať klauzulu, ktorá obsahuje konštantu, bude musieť byť premenná reprezentujúca túto konštantu pravdivá (prípadne nepravdivá, ak chceme konštantu logická 0) pre všetky splniteľné varianty funkcie. Ak je táto podmienka splnená, bude môcť táto premenná vystupovať ako konštanta. Ďalšou vecou, ktorá je pri konverzii zachovaná, je, že ak máme dva identické termy, bude pre ne použitá spoločná premenná.

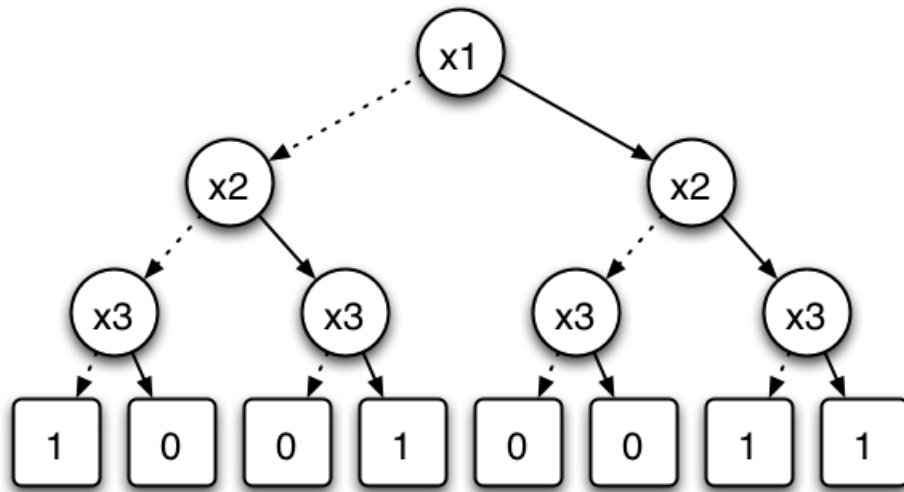
## 2.5 Binárne rozhodovacie diagramy

Binárne rozhodovacie diagramy (BDD) sú druhom grafov, ktorá je prevažne využívaná ako dátová štruktúra pre reprezentáciu Boolových funkcií v dnešnej dobe. Existujú viaceré implementácie, ktoré sú postavené práve na BDD. Používajú sa na riešenie problémov ekvivalencie a splniteľnosti výrazov. Sú veľmi dôležité v oblastiach designu hardvéru a optimalizácie.

BDD má podobu orientovaného koreňového acyklického grafu [5]. Skladá sa z viacerých uzlov. BDD má práve jeden uzol, ktorý nazývame koreňom. Koreň je jediný uzol, ktorý nemá predchodcov. Každý uzol je jeden z dvoch typov.

Uzol môže byť *neterminálny*, to znamená že nemá hodnotu, a vychádzajú z neho dva dcérske uzly. Uzly sú označované ako *low* a *high*, pre odlišenie jednotlivých podvetví stromu. Hrana smerujúca k *low* uzlu reprezentuje priradenie hodnoty logickej 0, hrana smerujúca k uzlu *high* reprezentuje priradenie hodnoty logickej 1.

Druhým typom je *terminálny* uzol, ktorý už nemá žiadnych potomkov, a má hodnotu z množiny  $\{0, 1\}$ . Príklad BDD je na obrázku 2.3. Neterminálne uzly sú označené kruhom a vpísaný majú index premennej ktorú reprezentujú, terminálne uzly sú označené štvorcami a vpísanú majú svoju hodnotu. Hrany low sú označené prerušovanou čiarou, hrany high sú označené plnou čiarou. Obrázok reprezentuje funkciu vyjadrenú pravdivostnou tabuľkou z Tabuľky 2.4.



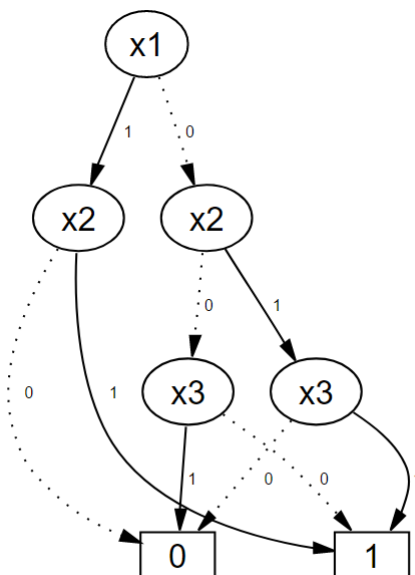
Obr. 2.3: Príklad binárneho rozhodovacieho diagramu. Hodnoty uzlov pochádzajú z pravdivostnej tabuľky v Tabuľke 2.4.

Tabuľka 2.4: Pravdivostná tabuľka pre funkciu na Obr. 2.3.

$x_1$	$x_2$	$x_3$	$f(x_1, x_2, x_3)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

V praxi sa často namiesto klasických BDD využívajú redukované binárne rozhodovacie diagramy ROBDD (Reduced Ordered Binary Decision Diagram). Sú špecifické tým, že všetky izomorfické podgrafy sú spojené do jedného [4]. Izomorfizmus dvoch grafov znamená, že grafy sú identické, ale len inak usporiadané. Pre ROBDD takisto platí, že ak uzol má

dva izomorfické podstromy, tento uzol je z grafu v rámci minimalizácie odstránený. Príklad ROBDD, ktorý je redukovaný z grafu na Obr. 2.3 je na Obr. 2.4.



Obr. 2.4: Redukovaný binárny rozhodovací diagram z BDD na Obr. 2.3.

Pre každú Boolovu funkciu existuje práve jeden ROBDD, ktorý je unikátny. ROBDD je teda kanonickou formou pre Boolove funkcie a preto je veľmi často využívaný v knižniciach reprezentujúcich Boolove funkcie.

## Kapitola 3

# Knižnice pre reprezentáciu Boolových funkcií

Existujú viaceré knižnice vytvorené za účelom manipulácie s Boolovými funkciami. Nasledujúca kapitola sa zaoberá niektorými vybranými, hlavne tými, ktoré využívajú binárne rozhodovacie stromy (BDD). Okrem toho existujú aj ďalšie implementácie, ktoré nevyužívajú BDD, ale sú založené napríklad na SAT. Tie sa využívajú v oblastiach overovania modelov, a vo väčšine prípadov majú vyššie pamäťové a časové nároky.

### 3.1 Colorado University Decision Diagram Package - CUDD

CUDD je verejne dostupná knižnica<sup>1</sup>, ktorej vývoj sa začal už v 70. rokoch a naďalej pokračuje. Je založená na prehľadávaní do hĺbky.

Balíček je možné využívať ako tzv. *black box*, teda používať len exportované funkcie, ale aj ako tzv. *clean box*, kde si programátor vie dodať vlastné dopĺňujúce funkcie.

Je napísaná v jazyku C a poskytuje funkcie pre manipuláciu s BDD, s algebraickými rozhodovacími diagramami (ADD, MTBDD) a s diagramami s potlačenou nulou (ZDD). Takisto poskytuje možnosť prevádzať medzi jednotlivými typmi diagramov.

CUDD využíva ukazovatele na uzly BDD [17]. Udržiava si počítadlo referencií. Počet premenných ovplyvňuje počet tabuliek. Knižnica využíva heuristiku, ktorá sprístupní tabuľku výpočtov len vtedy, ak aspoň jeden argument má hodnotu počítadla referencií väčšiu než jedna.

V CUDD existuje veľmi efektívny správca pamäte. Volá sa len vtedy, ak využitie pamäte prekročí určitú hranicu. Garbage Collector podľa počítadla referencií maže *mŕtve uzly*, teda uzly, ktoré majú hodnotu 0 v počítadle referencií.

### 3.2 CacBDD

Knižnica CacBDD je verejne dostupná<sup>2</sup> podobne ako knižnica CUDD, na rozdiel od nej je ale implementovaná v jazyku C++. Je založená na prehľadávaní do hĺbky.

Poskytuje základné operácie pre manipuláciu s BDD. BDD uzly sú uložené v jednom poli a využíva indexy uzlov v tomto poli namiesto ukazateľov na uzly, ako tomu je v CUDD. Nevyužíva počítadlo referencií na uzly. Garbage collector je volaný len ak je vyčerpaná

---

<sup>1</sup><http://vlsi.colorado.edu/~fabio/>

<sup>2</sup><http://www.kailesu.net/CacBDD/>



dostupná pamäť. Funguje inak ako v prípade CUDD, prechádza všetky uzly v poli, a tie na ktoré sa nikto neodkazuje a ani nie sú koreňmi, označí ako voľné uzly, nemaže ich a tým šetrí výpočtový čas. Knižnica využíva dynamické zväčšovanie tabuľky výpočtov podľa potreby, ak dôjde počet voľných miest. V knižnici je veľmi dobre implementované ukladanie medzivýsledkov, čo takisto pridáva na rýchlosť.

Experimentálne bolo dokázané, že knižnica v určitých prípadoch pracuje rýchlejšie a efektívnejšie [13] ako najrozšírenejšia knižnica CUDD. Testovaním porovnávacích štandardov ISCAS85 a smv-bdd-traces98 bolo zistené, že knižnica CacBDD je efektívnejšia a rýchlejšia, pričom v porovnávacom teste smv-bdd-traces98 sa pamäťové nároky takmer nelíšili. Hlavne v testoch *dpd75*, *furnace17* a *mmgt20* dosahuje knižnica niekoľkonásobné zrýchlenie.

### 3.3 BuDDy

Knižnica BuDDy je ďalšou knižnicou na prácu s Boolovymi funkciami. Je naprogramovaná v jazyku C, ale obsahuje aj C++ rozhranie pre jednoduchšiu prácu.

Obsahuje vlastný Garbage Collector, cache pamäť na uchovanie medzivýsledkov. Takmer každé nastavenie činnosti sa dá ručne nastaviť, ale obsahuje aj základné nastavenia pre užívateľov, ktorí nastavenia nechcú modifikovať.

Knižnica obsahuje veľké množstvo funkcií a operácií [1], ktoré sa dajú použiť na prácu s Boolovymi funkciami. Všetky výsledky v BuDDy sú reprezentované vektormi, a tým pádom sa s nimi v C++ ľahšie manipuluje.

### 3.4 BCL - Class Library for Boolean Function Manipulation

Knižnica<sup>3</sup> pre manipuláciu s Boolovymi funkciami vytvorená v jazyku C#, je vhodná pre využitie v jazykoch z rodiny .NET Framework.

Obsahuje viaceré interné reprezentácie Boolových funkcií, ako sú pravdivostné tabuľky, Boolove výrazy a BDD. Každá z reprezentácií obsahuje metódy na zjednodušenie funkcie, vytvorenie novej funkcie aplikovaním operátoru na dve funkcie, na nahradenie premennej konštantou a pre nahradenie premennej inou funkciou.

Knižnica sa využíva hlavne na výskumné účely, pretože obsahuje užitočné funkcie na určenie Shannonovho rozvoja, zistenie linearity a monotónnosti funkcie a mnohé ďalšie. Takisto obsahuje metódy konverzie medzi reprezentáciami, okrem iných aj konvertor z pravdivostnej tabuľky na ANF, DNF, CNF a BDD.

### 3.5 CORAL

Knižnica napísaná v jazyku C++, ktorá bola zamýšľaná na použitie v logických programovacích jazykoch [3], ale aj v iných. Podobne ako ostatné knižnice využíva ROBDD - Reduced Ordered BDD. Knižnica je zameraná hlavne na pamäťovú efektívnosť a na optimalizáciu.

---

<sup>3</sup><http://dispatcher.swu.bg/BCL/>

## 3.6 BDD

Knižnica<sup>4</sup> napísaná v C, primárne zameraná na operačné systémy UNIX, pre prácu mimo UNIX je potrebné upraviť správcu pamäte.

Obsahuje nástroje na sekvenčné overovanie, cache pamäť, na ukladanie výsledkov, kam sa ukladajú úplne všetky medzivýsledky, kvantifikácie viacerých premenných a substitúcie. Okrem toho obsahuje nástroje na analýzu BDD, napríklad histogram, možnosť uloženia BDD do súborov.

Garbage collector funguje na báze počítadla referencií alebo na princípe "zmaž všetky záznamy okrem". Takisto používateľ dokáže nastaviť limit na počet uzlov, operácie samé zmažú pamäť ak by museli prekročiť tento limit. Knižnica poskytuje aj možnosť dynamickeho preusporiadania premenných.

---

<sup>4</sup><http://www.cs.cmu.edu/afs/cs/project/modck/pub/www/bdd.html>

## Kapitola 4

# Konceptuálny návrh knižnice

V tejto kapitole si bližšie popíšeme návrh knižnice pre manipuláciu s Boolovymi funkciami v ANF. Požiadavkami na knižnicu sú možnosti presne a efektívne reprezentovať Boolove funkcie v ANF. Každá takáto funkcia sa skladá z viacerých častí, takže je potrebné vytvoriť štruktúry, ktoré ich budú verne reprezentovať. To všetko je potrebné dosiahnuť s čo najmenšími pamäťovými nárokmi. Každá Boolova funkcia obsahuje určitý obor premenných, ktoré sa môžu vo funkcii aj opakovať. Je potrebné navrhnúť efektívny spôsob práce s týmito premennými, aby sa dalo rýchlo pristupovať k ich hodnotám, ale pritom sme sa zbavili duplicít. Musí prebehnúť analýza dostupných možností a výber vhodného riešenia. Nad Boolovou funkciou sú definované rôzne operácie ako pridávanie či odoberanie premenných a termov. Od knižnice je požadované aby tieto operácie obsahovala.

Ďalšou požiadavkou na knižnicu je čo najefektívnejšia práca s pamäťou, a takisto aj rýchlosť pri manipulácii s veľkým počtom funkcií, prípadne premenných vo funkciách.

V neposlednom rade by knižnica mala obsahovať prostriedky pre grafickú reprezentáciu Boolovej funkcie, je teda potrebné sa zamerať na výber vhodného nástroja. Z vytvorenej grafickej reprezentácie by mala byť jasne odvoditeľná štruktúra danej Boolovej funkcie, ako aj hodnoty jej jednotlivých častí.

### 4.1 Voľba technológií pre vývoj knižnice

Väčšina existujúcich knižníc pre manipuláciu s Boolovymi funkciami bola vytvorená v programovacom jazyku C, prípadne C++. Pri tvorbe knižnice je treba dbať na rýchlosť a pamäťové nároky, preto sme si zvolili jazyk C ako najvhodnejšiu variantu. Programovacie jazyky vyššej úrovne sme odmietli z dôvodu, že v C sa dajú dosiahnuť najlepšie výsledky práve v spomenutých kategóriách (napríklad neriešime virtuálny stroj v Jave a podobne).

Ako platformu pre vývoj knižnice sme podobne ako existujúce riešenia zvolili UNIX.

### 4.2 Reprezentácia premenných

Každá Boolova funkcia obsahuje 0 až  $n$  premenných, ktoré je potrebné efektívne reprezentovať. Každá premenná má svoje pomenovanie a Boolovu hodnotu. Povolená dĺžka názvu premennej by mala byť dostatočná, aby sme dokázali unikátne reprezentovať veľký počet premenných. Príklad takejto premennej môžeme vidieť v Zdrojovom kóde 4.1.

Zdrojový kód 4.1: Návrh štruktúry reprezentujúcej premennú. Obsahuje pole pre svoje pomenovanie a pole pre hodnotu typu Boolean.

```
typedef struct variable {  
    char* name;  
    bool value;  
} tVar;
```

Premenné sa môžu vyskytovať v jednotlivých termoch funkcie opakovane, a takisto premenná môže byť súčasťou viacerých termov vo funkcii. Keďže má premenná vo všetkých svojich výskytoch rovnakú Boolovu hodnotu, je potrebné zaistiť, aby sa takéto duplicitné výskyty neukladali do pamäte opakovane.

Obor všetkých premenných je možné reprezentovať viacerými spôsobmi. Klasické pole poskytuje výhodu, že operácia vyhľadávania je rýchla, ak vieme presný index, na ktorý chceme prísť. To by bolo využiteľné, ak by premenné v Boolovej funkcii mali len číselný index a generovali sa od najnižších indexov po najvyššie (aby sme zbytočne nealokovali pamäť o väčšej veľkosti než je potrebná). V našom prípade by premenné mali mať ľubovoľné pomenovanie, a tento postup sa teda ukázal ako nevhodný.

Druhým spôsobom je použitie hashovacej tabuľky. Tá rieši vyššie spomenutý problém, pretože ak poznáme kľúč k danému záznamu, prístup k jeho hodnote je veľmi rýchly, v závislosti na hashovacej funkcii. Hashovacia tabuľka využíva na pozadí pole, pričom index daného záznamu nám poskytuje hashovacia funkcia, ktorá konvertuje kľúč záznamu na index v poli a naopak. V hashovacej tabuľke je možné zaistiť aj riešenie problému s ukladaním duplicitných záznamov, a to kontrolou, či záznam s daným kľúčom už v tabuľke existuje.

### 4.3 Hashovacia tabuľka premenných funkcie

Keďže v jazyku C neexistuje štruktúra hashovacia tabuľka, je potrebné nejakú vytvoriť. Kľúčom ku korektnému správaniu je voľba správnej hashovacej funkcie pre účely knižnice.

Primárnym účelom knižnice je jej využitie v obvodovej štruktúre s registrami, teda hardvérové využitie. Pre tieto účely nie je potrebné šifrovať záznamy v hashmape, keďže by sa malo jednať o čo najjednoduchšiu implementáciu. Preto sme sa rozhodli vyberať z nešifrovaných hashovacích funkcií, a podľa požadovaných parametrov zvoliť ideálnu alternatívu.

Existuje veľké množstvo voľne dostupných a takisto aj komerčných samostatných implementácií hashovacej tabuľky. Nástroj SMHasher<sup>1</sup> a jeho rozšírená verzia<sup>2</sup> poskytujú dobré porovnanie existujúcich hashovacích algoritmov. SMHasher testuje síce veľké množstvo hashovacích algoritmov, ale len niekoľko z nich má implementáciu v C, ktorá nás zaujíma. Na porovnanie hashovacích algoritmov implementovaných hlavne v C a C++ sa zameriava aj porovnávací test pre knižnicu TommyDS<sup>3</sup>. Poskytuje podrobné porovnanie rôznych operácií nad hashovacím tabuľkami naprieč viacerými implementáciami.

Na základe dvoch vyššie spomenutých porovnaní boli vyskúšané viaceré varianty hashovacej tabuľky. Niektoré knižnice, ktoré si viedli veľmi dobre v daných testoch, sú optimalizované na vysokej úrovni, ale chýba im prehľadnosť a príklady praktického využitia (konkrétne xxHash, ktorý si viedol výborne v teste SMHasher). Preto sme sa rozhodli pre využitie knižnice, ktorá síce nebude vo všetkých rebríčkoch na najvyšších pozíciách, ale bude si viesť nadpriemerne, pričom bude poskytovať vyššiu prehľadnosť kódu a dobrú

<sup>1</sup><https://github.com/aappleby/smhasher>

<sup>2</sup><https://github.com/rurban/smhasher>

<sup>3</sup><http://www.tommyds.it/doc/benchmark>

dokumentáciu. Podrobnejšie skúšané boli knižnice UTHash<sup>4</sup> a Judy<sup>5</sup>, pričom ako finálna knižnica bola zvolená práve knižnica UTHash. Jej hlavnou výhodou je, že je to len jediný hlavičkový súbor, obsahuje množstvo testovacích scenárov a v testoch si viedla na dobrej úrovni. Ďalším dôvodom, prečo sme si zvolili práve UTHash, je možnosť výberu ľubovoľného dátového typu pre kľúč, a takisto aj hodnoty hashovacej tabuľky, čo nie je bežné pri všetkých ostatných implementáciách hashovacej tabuľky.

Keď chceme využívať možnosti knižnice UTHash, je potrebné niečím naznačiť, ktorá štruktúra v programe má byť hashovateľná do hashovacej tabuľky. V UTHash sa to dosahuje pridaním jedného riadku do definície štruktúry. V tomto duchu teda bude pozmenený návrh štruktúry premennej zo Zdrojového kódu 4.1 tak, ako je naznačené v Zdrojovom kóde 4.2.

Zdrojový kód 4.2: Návrh štruktúry reprezentujúcej premennú obohatenú o pole, ktoré povolí hashovanie pomocou knižnice UTHash.

---

```
typedef struct variable {  
    char* name;  
    bool value;  
    UT_hash_handle hh;  
} tVar;
```

---

Výsledná hashovacia tabuľka teda využíva hlavičkový súbor z knižnice UTHash, obaľuje jeho potrebné funkcie vlastnými, pre jednoduchšie použitie, a takisto pridáva nejaké nové. Tento princíp dovoľuje prípadnú zmenu knižnice pre hashovacu tabuľku, pričom bude potrebné zmeniť volania funkcií len na jedinom mieste, a vo zvyšku súborov našej knižnice zostanú volania bezo zmeny.

UTHash využíva systém zretazovania záznamov (chaining z angl.), takže na jedno pamäťové miesto sa teoreticky môže mapovať viacero záznamov. Pri takejto kolízii sa potom podľa kľúča vyhľadáva správny záznam.

## 4.4 Reprezentácia termov vo funkcii

Každá Boolova funkcia v ANF sa skladá z 0 až  $m$  termov, ktoré obsahujú premenné. Premenné v terme sú medzi sebou prepojené operáciou logický súčin AND. Každý term by mal v sebe obsahovať informácie, ktoré premenné obsahuje a aký je ich celkový počet. Keďže medzi všetkým premennými je rovnaká operácia, nie je potrebné si uchovávať informáciu o tom, na ktorej pozícii v terme sa nachádza ktorá premenná. Je teda možné premenné reprezentovať jednoduchým zoznamom alebo poľom.

Premenné by malo byť do termu možné dynamicky vkladať, takisto ich z neho odoberať. Term môže obsahovať jednu premennú aj viackrát, prípadne žiadne premenné.

Každý term má aj svoju celkovú výslednú Boolovu hodnotu, ktorá je vypočítaná vykonaním operácie AND medzi všetkými premennými. Je dôležité myslieť na to, že ak budú do termu pridávané, alebo z neho odoberané premenné, mala by sa prepočítať aj táto výsledná hodnota.

Návrh štruktúry termu by mohol vyzeráť tak, ako ukazuje Zdrojový kód 4.3. Term už nemusí obsahovať priamo celé premenné, tie sú už uložené v hashovacej tabuľke celej Boolovej funkcie v ANF, v terme postačuje mať záznamy o názvoch premenných, ich hodnoty sa vyberú z hashovacej tabuľky.

---

<sup>4</sup><http://troydhanson.github.io/uthash/>

<sup>5</sup><http://judy.sourceforge.net/>

Zdrojový kód 4.3: Návrh štruktúry reprezentujúcej term (uzol). Štruktúra obsahuje zoznam identifikátorov premenných, ktoré obsahuje, ich celkový počet a výslednú Boolovu hodnotu.

---

```
typedef struct node {
    char** variables;
    int varCount;
    bool value;
} tNode;
```

---

## 4.5 Reprezentácia Boolovej funkcie

Štruktúra reprezentujúca Boolovu funkciu obsahuje všetky potrebné informácie o svojom obsahu. Návrh tejto štruktúry ukazuje Zdrojový kód 4.4. Štruktúra obsahuje zoznam termov, ktoré sa vo funkcii nachádzajú, a takisto informáciu o tom, koľko je termov dohromady vo funkcii. Keďže všetky termy v Boolovej funkcii vo forme ANF sú spojené operáciou exkluzívneho súčtu XOR, nie je potrebné si uchovávať informáciu o poradí termu vo funkcii. Operácia XOR je asociatívna, teda nezáleží na tom, v akom poradí budeme výraz s viacerými operandami počítat, a takisto nezáleží, kam dáme zátvorky. Túto vlastnosť operácie XOR môžeme vidieť v tabuľkách 4.1 a 4.2, ktoré obsahujú pravdivostnú tabuľku pre Boolovu funkciu v dvoch variantách, ktoré sa od seba líšia zátvorkami.

Tabuľka 4.1: Pravdivostná tabuľka pre funkciu  $x_1 \oplus (x_2 \oplus x_3)$

$x_1$	$x_2$	$x_3$	$x_1 \oplus (x_2 \oplus x_3)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Tabuľka 4.2: Pravdivostná tabuľka pre funkciu  $(x_1 \oplus x_2) \oplus x_3$

$x_1$	$x_2$	$x_3$	$(x_1 \oplus x_2) \oplus x_3$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Ďalej je v štruktúre obsiahnutá aj hashovacia tabuľka, obsahujúca všetky hodnoty premenných, ktoré sa v Boolovej funkcii vyskytujú. Hashovacia tabuľka je spoločná pre celú Boolovu funkciu.

Okrem spomenutých je potrebné v štruktúre zachovať informáciu o aktuálnej hodnote celej funkcie, vypočítanú vykonaním operácie XOR nad jednotlivými termami funkcie. Hodnota sa musí meniť správne podľa toho, ako sa manipuluje s termami. Či už sa termy pridávajú alebo odoberajú, alebo sa menia hodnoty premenných, hodnota celej funkcie musí byť uchovaná správne po celý čas.

Zdrojový kód 4.4: Návrh štruktúry Boolovej funkcie v ANF, obsahuje zoznam termov, ich celkový počet, hashovaciu tabuľku, v ktorej sú uchované hodnoty premenných a celkovú hodnotu funkcie.

---

```
typedef struct anf {
    tNode** nodeList;
    tHashMap* hashMap;
    int nodeCount;
    bool value;
} tAnf;
```

---

## 4.6 Optimalizácia a minimalizácia

Jednou z požiadaviek na knižnicu je jej efektívnosť. Je teda potrebné zaistiť čo najvyššiu úroveň optimalizácie funkcií. Prvou, už spomenutou optimalizáciou, je reprezentácia premenných pomocou hashovacej tabuľky. Týmto spôsobom sa predchádza duplikovaniu premenných.

Keďže každý term v sebe obsahuje odkazy na premenné, ktoré v sebe obsahuje, je potrebné optimalizovať aj tento zoznam. Keďže premenné majú medzi sebou vždy operáciu logický súčin AND, a platí Rovnica 4.1, nie je potrebné v zozname premenných pre daný term uchovávať túto informáciu dvakrát. Z toho dôvodu pred vložením premennej do termu prebieha kontrola, či sa tam už záznam nenachádza.

$$x_1 \wedge x_1 \wedge x_1 \wedge x_1 = x_1 \quad (4.1)$$

Ďalšou možnosťou, ako optimalizovať chod programu, je spôsob počítania celkovej Boolovej hodnoty termu. Keďže sa v terme nachádzajú len operácie AND, jedinou možnou kombináciou premenných (ako vidíme v Tabuľke 4.3, ktorá obsahuje pravdivostnú tabuľku pre funkciu o troch premenných), pre ktorú bude mať celý term hodnotu rovnú logickej 1, je tá, v ktorej majú všetky premenné ako hodnotu logickú 1. Z tohoto dôvodu sa pri vkladaní premennej do termu kontroluje aktuálna hodnota termu na rovnosť s logickou 0. Ak už má term uloženú hodnotu logická 0, je jasné, že obsahuje nejakú premennú s hodnotou rovnou logickej 0. Nie je teda potrebné počítat novú hodnotu.

Podobne aj pri mazaní premennej z termu je najprv vykonaná kontrola na rovnosť hodnoty termu s hodnotou logická 1. Ak term obsahuje nejaké premenné a má hodnotu logická 1, znamená to že všetky jeho premenné majú hodnotu logická 1, teda aj naša mazaná, a preto nie je potrebné hodnotu termu znova prepočítavať.

V neposlednom rade je optimalizáciou aj to, že zoznam premenných v terme (respektíve zoznam termov v štruktúre reprezentujúcej celú funkciu v ANF) obsahuje len odkazy na dané štruktúry, nie ich kompletný obsah.

Tabuľka 4.3: Pravdivostná tabuľka pre funkciu obsahujúcu tri premenné a dva AND operátory.

$x_1$	$x_2$	$x_3$	$x_1 \wedge x_2 \wedge x_3$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

## 4.7 Grafické zobrazenie funkcií

Jednou z požiadaviek na knižnicu je aj možnosť grafického zobrazenia vytvorených funkcií. Takýto graf by mal obsahovať všetky termy pre danú funkciu a takisto správne naznačovať, ktoré premenné sa nachádzajú v ktorých termoch.

Existuje viacero prístupov, ako pristupovať k reprezentácii štruktúr grafom. Spomenieme si dva hlavné formáty: textový formát a XML.

Zástupcom formátov založených na XML je napríklad DGML<sup>6</sup>. Tento formát bol vyvinutý Microsoftom a je používaný pri vizualizácii štruktúr vo Visual Studiu. Poskytuje možnosť tvoriť orientované aj neorientované grafy, ako aj napríklad možnosť anotovať jednotlivé prvky grafu. Príklad DGML môžeme vidieť v Zdrojovom kóde 4.5.

Z textových formátov je veľmi rozšírený formát DOT<sup>7</sup>. Poskytuje možnosť vytvárať orientované grafy. DOT je možné používať v príkazovom riadku, aj cez rôzne grafické prostredia, takisto je dostupných viacero online nástrojov. V Zdrojovom kóde 4.6 môžeme vidieť jednoduchý graf vo formáte DOT, na Obr. 4.1 potom, ako taký graf vyzerá.

Pre jednoduchosť formátu DOT, a takisto veľkej možnosti si ho vyskúšať online v rôznych nástrojoch, bol nakoniec práve tento formát zvolený ako vhodný pre reprezentáciu Boolových funkcií v implementovanej knižnici.

Okrem formátu pre tvorenie grafov knižnica bude obsahovať aj možnosť vypísania obsahu danej Boolovej funkcie do príkazového riadku v jednoduchom textovom symbolickom formáte. Táto možnosť bude slúžiť pre používateľov, ktorí nemajú prístup k žiadnemu nástroju pre vykreslenie obrázku vo formáte DOT.

<sup>6</sup><https://msdn.microsoft.com/en-us/library/dn966108.aspx>

<sup>7</sup><http://www.graphviz.org/pdf/dotguide.pdf>



Zdrojový kód 4.5: Ukážka jednoduchého kódu vo formáte DGML, ktorý sa používa na vizualizáciu štruktúr. Obsahuje elementy pre jednotlivé uzly a prepojenia v grafe, takisto aj pre vlastnosti jednotlivých uzlov.

---

```
<?xml version='1.0' encoding='utf-8'?>
<DirectedGraph xmlns="http://schemas.microsoft.com/vs/2009/dgml">
  <Nodes>
    <Node Id="x1" Label="x1" Size="10" />
    <Node Id="x2" Label="x2" />
    <Node Id="x3" Label="x3" />
    <Node Id="x4" Label="x4" />
    <Node Id="x5" Label="x5" />
  </Nodes>
  <Links>
    <Link Source="x1" Target="x2" Label="AND" />
    <Link Source="x2" Target="x3" Label="OR" />
    <Link Source="x3" Target="x4" Label="AND" />
    <Link Source="x4" Target="x5" Label="OR" />
  </Links>
  <Properties>
    <Property Id="Label" Label="Label" DataType="String" />
    <Property Id="Size" DataType="String" />
  </Properties>
</DirectedGraph>
```

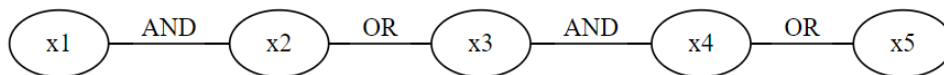
---

Zdrojový kód 4.6: Ukážka formátu DOT. Obsahuje niekoľko uzlov a prepojení medzi nimi.

---

```
1: graph LR
2:   rankdir=LR;
3:   x1 --- x2 [label="AND"];
4:   x2 --- x3 [label="OR"];
5:   x3 --- x4 [label="AND"];
6:   x4 --- x5 [label="OR"];
7: }
```

---



Obr. 4.1: Príklad grafu vytvoreného cez formát DOT zo Zdrojového kódu 4.6.

## Kapitola 5

# Implementácia navrhovaného riešenia a použité algoritmy

Táto kapitola popisuje spôsob implementácie knižnice podľa návrhu z kapitoly 4. Popisuje súborovú štruktúru a konkrétny obsah poskytovaných funkcií. Kompletný zoznam a popis funkcií sa nachádza na priloženom CD v dokumentácii, ako je popísané v prílohe A, v tejto kapitole si popíšeme tie zaujímavejšie časti implementácie.

Knižnica obsahuje viacero hlavičkových súborov, ktoré obsahujú popis štruktúr, ako boli načrtnuté v kapitole 4. Takisto hlavičkové súbory obsahujú deklarácie všetkých funkcií, ktoré majú byť dostupné pre používateľa knižnice. K väčšine hlavičkových súborov existuje aj súbor s príponou `.c`, ktorý obsahuje implementácie funkcií deklarovaných v hlavičkovom súbore, a okrem toho aj pomocné funkcie, ktoré pomáhajú prehľadnejšiemu zdrojovému kódu, ale nemajú byť dostupné pre volanie programátorom.

Okrem zdrojových súborov sa v balíčku nachádza aj testovací súbor `tests.c`, ktorý obsahuje niekoľko testov pre overenie funkčnosti väčšiny funkcií poskytovaných knižnicou. Tento testovací súbor takisto ukazuje možné použitie knižnice a tiež ukážku vykreslenia grafu Boolovej funkcie v ANF do súboru vo formáte DOT.

V knižnici sa okrem autorských súborov nachádza aj hlavičkový súbor `uthash.h`. Tento súbor je vytiahnutý z balíčka UTHash<sup>1</sup> a je bezo zmeny používaný. Súbor bol vybraný, namiesto využitia kompletného balíčka UTHash preto, že v našej knižnici nepotrebujeme ostatné štruktúry a funkcie dostupné z ostatných súborov z UTHash, vystačíme si len s týmto jedným súborom.

Knižnica je implementovaná a testovaná na systéme UNIX, konkrétne Ubuntu verzie 16.04, takže obsahuje prekladový súbor `makefile` len pre túto platformu. V aktuálnom stave sa knižnica vytvára ako zdieľaná, je teda pri preklade zdrojových súborov mať administrátorské práva, aby bolo možné nakopírovať príslušné súbory na patričné miesta v systéme. Používateľovi ale nič nebráni v tom nevyužiť poskytnutý `makefile` a používať súbory knižnice lokálne ako nezdieľanú knižnicu.

Hlavná logika čo sa týka práce s hashovacou tabuľkou sa nachádza v súbore `anfhashmap.c`. Nachádzajú sa tu funkcie, ktoré sprístupňujú funkcie z knižnice UTHash pre použitie v programe, ktoré sú vysoko optimalizované, napríklad preto, že sú implementované ako makrá. Knižnica UTHash pracuje veľmi dobre s pamäťou, za predpokladu, že si používateľ zvolí správny typ kľúča pre vybranú hashovaciu funkciu. Dostupné sú funkcie pre vytvorenie a rušenie hashovacej tabuľky, pre vkladanie, výber a odoberanie záznamov. Okrem toho

---

<sup>1</sup><https://troydhanson.github.io/uthash/>

obsahuje funkcie pre výpis obsahu hashovacej tabuľky do konzoly a takisto pre vykreslenie do grafu vo formáte DOT.

Reprezentáciu premenných a termov poskytujú súbory `anfnode.c` a `anfvariable.c` a ich príslušné hlavičkové súbory. Používateľ má možnosť vytvárania premenných a termov, pridania a odoberania premenných do/z termu a takisto vypísanie obsahu termu do konzoly.

## 5.1 Počítanie hodnoty termu

Aktuálna Boolova hodnota daného termu sa vypočíta po vložení novej premennej či premenných, a takisto pri mazaní premennej. Algoritmus pre vkladanie je naznačený v Algoritme 1. Ak sa do termu vkladá prvá hodnota, nevykonáva sa žiadna operácia a rovno sa vkladaná hodnota priradí do hodnoty celého termu. Pri vkladaní ďalších hodnot sa kontroluje, či aktuálna hodnota termu nie je logická 0. Ak by hodnota termu bola logická 0, nie je potrebné vypočítavať novú hodnotu, pretože výsledok operácie logický súčin AND s hodnotou logická 0 a akoukoľvek inou hodnotou bude vždy logická 0. Iba ak je aktuálna hodnota termu logická 1, vtedy sa vykonáva operácia AND s aktuálnou a novou vkladanou hodnotou a výsledok sa uloží do hodnoty termu.

Mazanie premennej z termu je založené na inom princípe. Tento algoritmus sa nachádza v Algoritme 2. Ak má mazaná premenná hodnotu logická 1 a aj celý term má hodnotu logická 1, za podmienky že po mazaní bude mať term aspoň jednu ďalšiu premennú, nie je potrebné znovu prepočítavať hodnotu termu. V prípade, že mazaná premenná má hodnotu logická 0 a po zmazaní bude mať term aspoň jednu ďalšiu premennú, je potrebné znovu prepočítať hodnotu termu pre každú premennú, ktorá v terme zostala podľa Algoritmu 1. Ak mažeme poslednú premennú termu, je hodnota nastavená rovno na logickú 0.

## 5.2 Možnosti práce s Boolovými funkciami

Navrhnutá a implementovaná knižnica poskytuje veľké množstvo operácií nad Boolovými funkciami v ANF. Obsahuje možnosti ako vytvárať a mazať tieto funkcie. Takisto dovoľuje vo funkciách vytvárať či mazať termy s premennými alebo bez nich. Tiež je možné meniť hodnoty premenných, a podľa toho sa prepočítavajú hodnoty všetkých termov, v ktorých sa premenná vyskytuje, a tým pádom aj hodnota celej funkcie.

Takisto je možné spojiť dve funkcie do jednej, pričom nad danými funkciami bude vykonaná operácia exkluzívny súčet XOR. V tomto prípade môže nastať situácia, že premenná jedného mena bude mať v každej funkcii rôznu hodnotu. Je teda potrebné špecifikovať, ktorá funkcia má vyššiu prioritu, a teda pri takýchto konfliktoch bude hodnota premenných z danej vybranej funkcie mať prednosť pred hodnotou z druhej funkcie.

Pri vkladaní termov do funkcie, či ich mazaní sa využívajú podobné algoritmy pre výpočet výslednej hodnoty funkcie, ako Algoritmy 1 a 2 pre vkladanie premennej do termu. Prvý rozdiel v algoritme pre vkladanie je, že sa nevykonáva operácia logický súčin AND, ale namiesto nej exkluzívny súčet XOR. Druhý rozdiel je, že hodnota, nad ktorou sa vykonáva XOR s aktuálnou hodnotou funkcie, je hodnota daného vkladaneho termu, ktorá je vypočítaná ako výsledok operácie AND medzi všetkými jeho premennými podľa Algoritmu 1.

Všetky tieto operácie je možné nájsť v súbore `anf.c`.

---

**Algoritmus 1:** Počítanie hodnoty pri vkladaní premennej do termu. *varCount* označuje počet premenných v terme, *nodeValue* označuje Boolovu hodnotu daného termu a *newValue* je hodnota premennej, ktorú do termu vkladáme.

---

```
1:  if (varCount == 0){
2:      nodeValue = newValue;
3:  }
4:  elif (nodeValue == false){
5:      return;
6:  }
7:  else{
8:      nodeValue &= newValue;
9:  }
```

---

---

**Algoritmus 2:** Počítanie hodnoty pri mazaní premennej z termu. *varCount* označuje počet premenných v terme, *nodeValue* označuje Boolovu hodnotu daného termu a *value* hodnotu premennej na aktuálnom indexe pri prehľadávaní termu.

---

```
1:  if (varCount == 0){
2:      nodeValue = false;
3:      return;
4:  }
5:  if (nodeValue == true){
6:      return;
7:  }
8:  for (i = 0; i < varCount; i++){
9:      value = getValueOfRecordOnIndex(i);
10:     if (i == 0){
11:         nodeValue = value
12:     }
13:     elif (nodeValue == false){
14:         break;
15:     }
16:     else{
17:         nodeValue &= value
18:     }
19: }
```

---

### 5.3 Grafická reprezentácia štruktúr

Podľa výsledkov analýzy dostupných formátov v podkapitole 4.7 bol ako vhodný formát pre reprezentáciu Boolových funkcií v ANF zvolený formát DOT. V tejto podkapitole si bližšie popíšeme ako je to vlastne implementované a čo všetko sa dá zobraziť.

Implementovaná knižnica poskytuje možnosť zobrazenia dvoch dátových štruktúr, a to hashovacej tabuľky a kompletnej Boolovej funkcie v ANF.

Formát DOT poskytuje možnosť tvorenia grafov rôznych veľkostí a tvarov, pre účely knižnice bol vybraný formát **record**, ktorý najpríjemnejšie reprezentuje požadovanú štruktúru. Graf je tvorený hierarchickou štruktúrou uzlov smerujúcich zľava doprava, aby sa dala ľahko rozoznať úroveň zanorenia. Takisto sú v ňom jasne dané väzby medzi uzlami.

Pre graf hashovacej tabuľky bola zvolená reprezentácia s jednou úrovňou zanorenia. Konkrétny spôsob implementácie sa nachádza v Zdrojovom kóde 5.1. Na začiatku sa nastavujú veľkosti jednotlivých uzlov, a v druhej časti kódu už ich obsah a prepojenia. Generovanie daného kódu prebieha v dvoch častiach. Najprv sa vygeneruje úvodná sekvencia konfiguračných príkazov, ktorá je rovnaká pre každý generovaný graf. V druhej časti sa generuje už samotný obsah buniek, a to cyklom, ktorý prechádza hashovaciu tabuľku a vyberá z nej záznamy. Výsledný vygenerovaný graf sa nachádza na Obr. 5.1.

Generovanie grafu pre Boolovu funkciu v ANF prebieha v podobnom duchu ako aj pre hashovaciu tabuľku. Rozdiel je v tom, že tu už máme dve úrovne zanorenia, a to zobrazenie všetkých termov vo funkcií, a v nich zobrazenie všetkých premenných. Implementovaná je možnosť zobrazovania premenných spoločne s ich hodnotou, alebo bez nej. Keďže tu existujú dve úrovne zanorenia, bolo potrebné upraviť algoritmus pre generovanie zdrojového súboru ku grafu. Namiesto jediného cyklu ako tomu bolo pri grafe hashovacej funkcie tu prebiehajú dva cykly aby sa správne vykreslili všetky uzly a spojenia medzi nimi. Príklad zdrojového súboru k Boolovej funkcii v ANF sa nachádza v Zdrojovom súbore 5.2. Vygenerovaný graf z daného súboru je na Obr. 5.2.

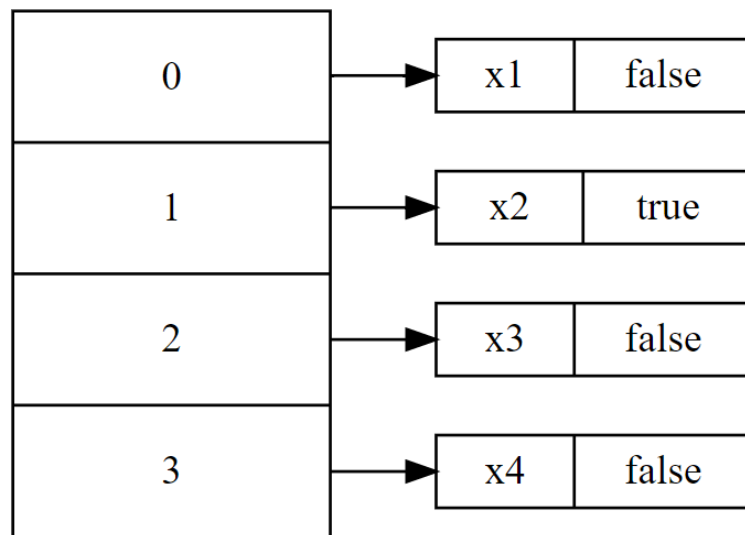
Zvolený formát dobre prezentuje štruktúru danej funkcie či hashovacej tabuľky a prehľadne zobrazuje ich obsah.

Zdrojový kód 5.1: Zdrojový súbor k hashovacej tabuľke vo formáte DOT.  
Obsahuje identifikátory jednotlivých uzlov a prepojenia medzi nimi.

---

```
digraph G {
    nodesep=.05;
    rankdir=LR;
    node [shape=record,width=.1,height=.1];
    node [width = 1.5];
    node0[label = "<n0> 0 | <n1> 1 | <n2> 2 | <n3> 3",height=2.5];
    node1[label = "{ <n> x1 | false }"];
    node0:n0 -> node1:n;
    node2[label = "{ <n> x2 | true }"];
    node0:n1 -> node2:n;
    node3[label = "{ <n> x3 | false }"];
    node0:n2 -> node3:n;
    node4[label = "{ <n> x4 | false }"];
    node0:n3 -> node4:n;
}
```

---



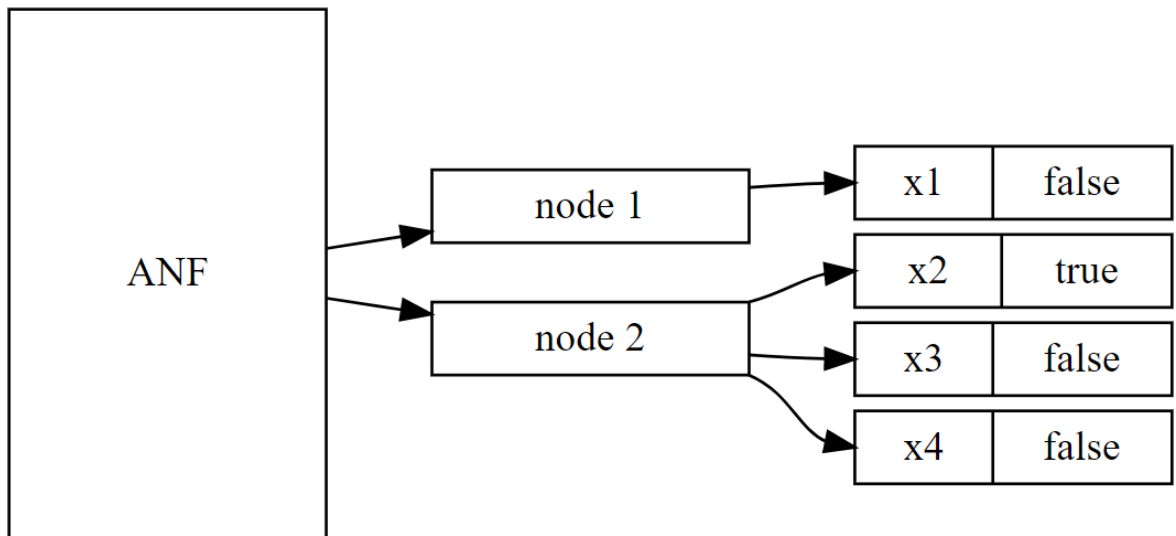
Obr. 5.1: Vygenerovaný graf zo Zdrojového kódu 5.1.

Zdrojový kód 5.2: Zdrojový súbor k Boolovej funkcii v ANF vo formáte DOT. Obsahuje identifikátory pre uzly a prepojenia medzi nimi.

---

```
digraph G {
  nodesep=.05;
  rankdir=LR;
  node [shape=record,width=.1,height=.1];
  node [width = 1.5];
  node0[label = "<n0> ANF",height=2.5];
  node1[label = "{ <n> node 1 }"];
  node0:n0 -> node1:n;
  node2[label = "{ <n> node 2 }"];
  node0:n0 -> node2:n;
  node3[label = "{ <n> x1 | false }"];
  node1:n -> node3:n;
  node4[label = "{ <n> x2 | true }"];
  node2:n -> node4:n;
  node5[label = "{ <n> x3 | false }"];
  node2:n -> node5:n;
  node6[label = "{ <n> x4 | false }"];
  node2:n -> node6:n;
}
```

---



Obr. 5.2: Vygenerovaný graf zo Zdrojového kódu 5.2.

## Kapitola 6

# Porovnanie knižnice s existujúcimi riešeniami

Táto kapitola sa bližšie zameriava popisom a vyhodnotením vykonaných experimentov nad implementovanou knižnicou, a porovnanie výsledkov s knižnicou CUDD, ktorá je spomenutá v podkapitole 3.1. Knižnica CUDD bola testovaná vo verzii 3.0.0. Všetky testy spomenuté v tejto kapitole prebehli na systéme Ubuntu 64bit.

### 6.1 Experiment 1 - Boolova funkcia malého rozsahu

V tomto experimente bola vytvorená jednoduchá Boolova funkcia o troch premenných za pomoci knižnice CUDD, a následne pomocou nami vytvorenej knižnice (ďalej v kapitole bude označovaná ako knižnica ANFlib). Experiment je zameraný na používateľa knižnice, ktorý potrebuje reprezentovať či manipulovať s Boolovou funkciou či funkciami malého rozsahu. Výsledkom experimentu je zistenie výhod a nevýhod oboch knižníc pre daný prípad. Ako testovacia Boolova funkcia bola vybraná funkcia z Rovnice 6.1.

$$f = x_1 \oplus x_2 \oplus x_3 \quad (6.1)$$

V oboch zvolených knižniciach je vytvorenie danej funkcie za pomoci prostriedkov knižníc otázkou niekoľkých desiatok riadkov kódu, takže v tomto ohľade nie je medzi knižnicami rozdiel, s obomi sa pracuje intuitívne.

Knižnica CUDD je postavená na BDD, takže už zo svojej podstaty sa v nej nenastavuje konkrétna hodnota premenných, ale je použitá všeobecne neznáma hodnota. Až vytvorený binárny rozhodovací diagram sa postará o vytvorenie všetkých dostupných kombinácií vstupných hodnôt premenných, a podľa toho sa generuje štruktúra BDD. Knižnica ANFlib nepracuje na princípe BDD, v tomto prípade je potrebné nastaviť u každej vstupnej premennej Boolovu hodnotu. Ak chceme pomocou knižnice ANFlib zistiť splniteľnosť Boolovej funkcie, musíme vytvoriť toľko samostatných Boolových funkcií, koľko je možných kombinácií vstupných hodnôt premenných, konkrétne  $2^n$  funkcií, ako bolo spomenuté v podkapitole 2.1.

U oboch knižníc v tomto experimente prebehlo meranie, koľko pamäťového miesta zabera beh programu, ktorý vytvorí Boolovu funkciu z Rovnice 6.1, vypíše o nej štatistiky do konzoly a vykreslí graf funkcie vo formáte DOT. Knižnica CUDD je implementovaná v objektovo orientovanom jazyku C++ a je rozsahovo omnoho väčšia, a tým pádom sa musia inicializovať rozsiahlejšie štruktúry ako v knižnici ANFlib, takže toto meranie nemá za



úlohu vybrať lepšiu alebo horšiu variantu, pravé výhody knižnice CUDD by sa mali prejavíť až pri funkciách s veľkým rozsahom. Toto meranie je teda skôr informatívne. Je ale dôležité si ukázať pamäťové nároky oboch knižníc aj pre úlohy menšieho rozsahu.

Pre meranie bol použitý nástroj **valgrind**, konkrétne jeho súčasť **massif**, čo je nástroj pre meranie využitia haldy počas behu programu. Záznam o maximálnom pamäťovom využití programu sa nachádza v Tabuľke 6.1.

Tabuľka 6.1: Maximálne pamäťové nároky programu, ktorý využíva knižnicu CUDD, resp. ANFlib na vytvorenie a vykreslenie Boolovej funkcie o troch premenných.

Knižnica	Maximálna obsadená časť haldy (v bytoch)
CUDD	10 535 032
ANFlib	7 184

Na meranie rýchlosti vykonania programu bol použitý UNIX príkaz **time**, ktorý ukazuje systémový aj reálny čas, ako dlho bežal daný program. Pre programy s tak malou veľkosťou ako v tomto experimente sú výsledné čísla príliš malé aby bol badateľný nejaký značný rozdiel, výsledky je možné vidieť v Tabuľke 6.2.

Tabuľka 6.2: Rýchlosť vykonania programov, ktoré vytvoria a vykreslia Boolovu funkciu o troch premenných za pomoci knižníc CUDD a ANFlib. Čísla sú priemerom z piatich meraní za sebou.

Knižnica	reálny čas	systémový čas
CUDD	0.009s	0.004s
ANFlib	0.002s	0.000s

Experiment dokazuje, že knižnica ANFlib je využiteľná pre prípady, ak používateľ potrebuje reprezentovať Boolovu funkciu malého rozsahu, hlavne z dôvodu, že knižnica CUDD si potrebuje alokovať väčšie množstvo pamäti pre svoje vnútorné štruktúry. Čo sa týka rýchlosti oboch knižníc, rozdiely sú zanedbateľné a obe knižnice sa dajú považovať za rovnako rýchle v tomto experimente.

## 6.2 Experiment 2 - Boolova funkcia s vysokým počtom premenných

Tento experiment je variáciou Experimentu 1 v podkapitole 6.1. Experiment skúma vlastnosti testovaných knižníc pre Boolove funkcie s vysokým počtom premenných.

Pamäťové nároky jednotlivých častí experimentu sú popísané v Tabuľke 6.3, časová náročnosť jednotlivých meraní je v Tabuľke 6.4.

Pre testovanie boli vybrané dve Boolove funkcie, ktoré sa nachádzajú v Rovnici 6.2 a Rovnici 6.3. Tieto dve Boolove funkcie boli vybrané z dôvodu, že sa u nich očakávajú pre knižnicu ANFlib z pohľadu výkonnosti dva extrémny. Pri reprezentácii funkcie z Rovnice 6.2 by si podľa očakávania mala viesť funkcia ANFlib dobre, pretože je založená na ANF, a pri reprezentácii pomocou ANF je operácia logický súčin AND veľmi rýchlou operáciou, keďže všetky premenné, nad ktorými je operácia AND vykonaná patria do jedného termu Boolovej funkcie.

$$f = x_1 \wedge x_2 \wedge \cdots \wedge x_n \quad (6.2)$$

$$f = x_1 \oplus x_2 \oplus \cdots \oplus x_n \quad (6.3)$$

Naopak, funkcia v Rovnici 6.3 sa v knižnici ANFlib reprezentuje spôsobom, že každá premenná sa nachádza v samostatnom terme, a tým pádom by mala mať knižnica pri jej reprezentácii vyššie pamäťové nároky.

Knižnica ANFlib poskytuje prostriedky, ako ju používať správne, a takisto ako aj nesprávne. Funkcia z Rovnice 6.3 by sa dala reprezentovať aj spôsobom, že každá premenná sa vytvorí v samostatnej Boolovej funkcii, a tieto funkcie sa následne spoja do jednej. To je príklad nesprávneho využitia prostriedkov knižnice, je to riešenie, ktoré síce bude fungovať, ale jeho nároky na pamäť a čas sa niekoľkonásobne zvýšia.

Tabuľka 6.3: Maximálne pamäťové nároky programu, ktorý využíva knižnicu CUDD, resp. ANFlib na vytvorenie a vykreslenie Boolovej funkcie o viacerých premenných. Vytvárané Boolove funkcie sú tie z rovníc 6.2 a 6.3. V stĺpci Rovnica je označené, z ktorej rovnice bola vybraná použitá Boolova funkcia, poznámka *extrém* označuje variantu, keď je každá premenná v samostatnej funkcii.

Knižnica	Počet premenných	Rovnica	Maximálna obsadená časť haldy (v bytoch)
CUDD	1000	6.2	11 663 912
CUDD	1000	6.3	11 747 320
CUDD	5000	6.2	37 068 248
CUDD	5000	6.3	37 068 248
ANFlib	1000	6.2	170 440
ANFlib	1000	6.3	175 256
ANFlib	1000	6.3 - extrém	917 016
ANFlib	5000	6.2	863 944
ANFlib	5000	6.3	869 248
ANFlib	5000	6.3 - extrém	4 600 320

Tabuľka 6.4: Rýchlosť vykonania programov, ktoré vytvoria a vykreslia Boolovu funkciu za pomoci knižníc CUDD a ANFlib. Čísla sú priemerom z piatich meraní za sebou.

Knižnica	Počet premenných	Rovnica	reálny čas	systémový čas
CUDD	1000	6.2	0.046s	0.017s
CUDD	1000	6.3	0.056s	0.020s
CUDD	5000	6.2	0.278s	0.096s
CUDD	5000	6.3	0.217s	0.116s
ANFlib	1000	6.2	0.026s	0.012s
ANFlib	1000	6.3	0.023s	0.004s
ANFlib	1000	6.3 - extrém	0.703s	0.624s
ANFlib	5000	6.2	0.338s	0.240s
ANFlib	5000	6.3	0.278s	0.048s
ANFlib	5000	6.3 - extrém	22.852s	21.740s

Z experimentu vyplýva, že knižnica ANFlib si vedie dobre v pamäťových nárokoch aj pri vyššom počte použitých premenných, naopak v časových nárokoch si vedie lepšie knižnica CUDD.

## Kapitola 7

## Záver

# Literatúra

- [1] BuDDy: A BDD package. Online 16.05.2017.  
URL <http://buddy.sourceforge.net/manual/main.html>
- [2] Bard, G. V.; Courtois, N. T.; Jefferson., C.: Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers. Cryptology ePrint Archive, Report 2007/024, 2007.  
URL <http://eprint.iacr.org/2007/024>
- [3] Bossi, F.: CORAL: a modern C++ library for the manipulation of Boolean functions. Online 16.05.2017.  
URL [http://www.cs.unipr.it/Informatica/Tesi/Fabio\\_Bossi\\_20090225.pdf](http://www.cs.unipr.it/Informatica/Tesi/Fabio_Bossi_20090225.pdf)
- [4] Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Comput.*, ročník 35, č. 8, Srpen 1986: s. 677–691, ISSN 0018-9340.
- [5] Bryant, R. E.: Symbolic Boolean Manipulation with Ordered Binary-decision Diagrams. *ACM Comput. Surv.*, ročník 24, č. 3, Zář 1992: s. 293–318, ISSN 0360-0300.
- [6] Comerford, J. D.: Affine and general linear equivalences of boolean functions. *Information and Control*, ročník 45, č. 2, 1980: s. 156 – 169, ISSN 0019-9958.
- [7] Courtois, N. T.; Bard, G. V.: Algebraic Cryptanalysis of the Data Encryption Standard. *Proceedings of the 11th IMA International Conference on Cryptography and Coding*, 2007: s. 152–169.
- [8] Crama, Y.; Hammer, P. L.: *Boolean Functions: Theory, Algorithms, and Applications*. NY, New York: Cambridge University Press, 2011, ISBN 9780521847513, doi:10.1017/CBO9780511852008.
- [9] Davio, P.; Deschamps, J. P.; Thayse, A.: *Discrete and Switching Functions*. New York: McGraw-Hill, 1978.
- [10] Hazewinkel, M.: *Encyclopaedia of Mathematics*. Springer, 1994, ISBN 9781556080104.
- [11] Janson, C.: *Boolean Functions*. Diplomová práce, Universität Bremen, 2012, online 16.05.2017.  
URL <http://crypto.math.uni-bremen.de/Arbeiten/dipljanson.pdf>
- [12] Koppelberg, S.: *Handbook of Boolean algebras Volume 1*. North Holland, 1989, ISBN 044470261X.

- [13] Lv, G.; Su, K.; Xu, Y.: *CacBDD: A BDD Package with Dynamic Cache Management*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, ISBN 978-3-642-39799-8, s. 229–234.
- [14] Moore, C.; Mertens, S.: *The Nature of Computation*. Oxford University Press, 2011, ISBN 0199233217.
- [15] Muller, D. E.: *Applications of Boolean Algebra to Switching Circuit Design and to Error Detection*. IRE Trans. Electronic Computers, vol. 3, 1954, pp. 6-12.
- [16] Reed, I. S.: *A Class of Multiple-Error-Correcting Codes and Their Decoding Scheme*. IRE Trans. Information Theory, vol. 4, 1954, pp. 38-42.
- [17] Somenzi, F.: CUDD: CU Decision Diagram Package. 2015.  
URL <http://vlsi.colorado.edu/~fabio/CUDD/cudd.pdf>
- [18] Wu, C.-K.; Feng, D.: *Boolean Functions and Their Applications in Cryptography (Advances in Computer Science and Technology)*. Springer, 2016, ISBN 978-3-662-48865-2.
- [19] Zhegalkin, I. I.: *On the Technique of Calculation the Sentences in Symbolic Logic*. Matem. Sbornik, vol. 34, 1927, pp. 9-28, v ruštine.

## Príloha A

### Obsah CD

- zdrojové súbory
- doxygen dokumentácia
- zdrojové súbory k tomuto dokumentu