VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

KNIŽNICA PRE BOOLOVE FUNKCIE V ALGEBRAICKEJ NORMÁLNEJ FORME

LIBRARY FOR BOOLEAN FUNCTIONS IN ALGEBRAIC NORMAL FORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MAROŠ VASILIŠIN**
AUTHOR

VEDOUCÍ PRÁCE **Ing. ROLAND DOBAI, Ph.D.**
SUPERVISOR

BRNO 2017

Abstrakt
Do tohoto odstavce bude zapsán výtah (abstrakt) práce v českém (slovenském) jazyce.

Abstract
Do tohoto odstavce bude zapsán výtah (abstrakt) práce v anglickém jazyce.

Kľúčové slová
Sem budou zapsána jednotlivá klíčová slova v českém (slovenském) jazyce, oddělená čárkami.

Keywords
Sem budou zapsána jednotlivá klíčová slova v anglickém jazyce, oddělená čárkami.

Citácia
VASILIŠIN, Maroš. *Knižnica pre Boolove funkcie v Algebraickej Normálnej Forme.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Roland Dobai, Ph.D.

Knižnica pre Boolove funkcie v Algebraickej Normálnej Forme

Prehlásenie

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana X... Další informace mi poskytli... Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

....
Maroš Vasilišin
13. mája 2017

Poďakovanie

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant, apod.).

Obsah

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. Takisto sa používa v teórií grafov pre návrh orientovaných grafov či vo vysokoúrovňovom programovaní ako vyjadrenie rôznych stavov systému.

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 viaceré reprezentácie Boolovych 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. BDD umožňujú veľmi dobre zisťovať ekvivalenciu a splniteľnosť Boolovych funkcií.

Reprezentácia pomocou BDD je v informatike síce najrozšírenejšia, ale nie je jediná. Táto práca sa zaoberá podrobnejšie reprezentáciou Boolovych 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.

Vytvorená knižnica poskytuje prostriedky pre efektívnu manipuláciu a zobrazovanie Boolovych funkcií v ANF. Motiváciou pre vytvorenie knižnice bolo vytvorenie slušnej alternatívy voči klasickým reprezentáciám pomocou BDD a zistenie, či táto reprezentácia poskytuje výhody oproti reprezentácií cez BDD.

V kapitole 2 si povieme najskôr niečo o rôznych reprezentáciách Boolovych 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

V tejto kapitole sa nachádza teoretický úvod do problematiky Boolovych funkcií, postupne bude definované, čo vlastne sú Boolove funkcie, čo sa dá pomocou nich popísať a aký môže byť ich obsah. Ďalej budú popísané rôzne možnosti zobrazenia Boolovych funkcií, napríklad pravdivostné tabuľky a ďalšie. Kapitola takisto definuje rôzne normalizované formy zápisu Boolovych funkcií, pričom dôraz bude kladený hlavne na Algebraickú Normálnu Formu, ktorej reprezentácia je cieľom celej práce. Podrobnejšie bude popísaná aj reprezentácia binárnymi rozhodovacími diagramami, ktoré sú momentálne najpoužívanejšou reprezentáciou Boolovych funkcií v oblasti počítačovej vedy.

2.1 Definícia Boolovej funkcie

Ako uvádza Crama [6], Boolova funkcia je každá funkcia $f : \mathcal{B}^n \to \mathcal{B}$, 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 $X^* = (x_1, x_2, \ldots, x_n)$ naberá hodnotu buď logická 0 alebo logická 1 z množiny $\mathcal{B}$.

Celkový počet rôznych Boolovych 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 Boolovych funkcií pre niektoré hodnoty n sa nachádza v Tabuľke 2.1.

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

n	počet funkcií
1	4
2	16
3	256
5	4.29497×10^9
6	1.84467×10^{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 {true,false}, {1,-1}, {on,off}, {áno,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 $\{+, \cdot, -\}$ [9].

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

Tabuľka 2.2: Pravidlá Boolovej algebry, x, y, z označujú navzájom rôzne premenné v Boolovej funkcií, $\wedge, \vee, \neg$ označujú operácie konjunkciu, disjunkciu a negáciu.

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

Tieto pravidlá sa využívajú pri zjednodušovaní Boolovych funkcií a pri zisťovaní ekvivalencie.

Operáciou, ktorá nepatrí do trojice základných Boolovych 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 konštrukcií obvodov alebo v generátoroch pseudonáhodných čísel. Exkluzívny súčet je zároveň jednou z dvoch operácií, ktorá je obsiahnutá v Algebraickej Normálnej Forme, spolu s logickým súčinom.

$$A \oplus B = (A \wedge \neg B) \vee (B \wedge \neg A) \tag{2.1}$$

2.2 Spôsoby reprezentácie Boolovych funkcií

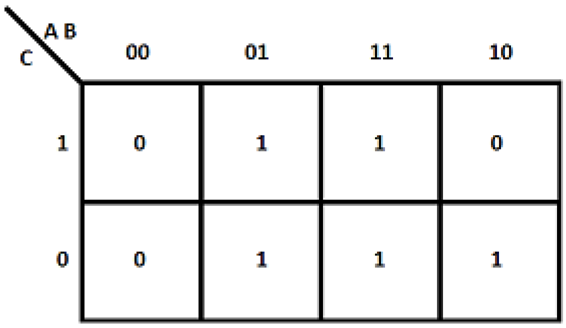
Boolove funkcie môžu byť vyjadrené rôznymi spôsobmi. Záleží hlavne na tom, na čo bude daná funkcia využitá, a aké operácie s ňou budú vykonávané. Niektoré zápisy sú vhodnejšie na matematické výpočty, iné zase na prehľadné prezeranie dát.

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

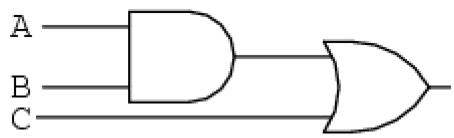
Upravenou formou pravdivostnej tabuľky je Karnaughova mapa. Je to forma zápisu, ktorá prevádza n-rozmernú Boolovu funkciu do dvojrozmernej. Jej výhodou je, že sa pomocou nej dá funkcia pekne vizualizovať, do 5 premenných poskytuje stále dobrú predstavu. Využíva sa hlavne pri minimalizácii funkcií. Je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



Obr. 2.1: Príklad Karnaughovej mapy pre funkciu $f = B \vee (A \wedge \neg C)$.

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 zobraziť 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 $(A \wedge B) \vee C$ 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 sekcia 2.3.



Obr. 2.2: Príklad logického obvodu funkcie $(A \wedge B) \vee C$.

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

2.3 Normálne formy Boolovych funkcií

Normálna forma je každý výraz v tvare:

$$T_1 \quad op \quad T_2 \quad op \quad T_3 \quad op \quad \dots \quad op \quad T_n$$

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 [8].

- Disjunktívna - termy sú konjunkciou premenných a operáciou je disjunkcia.
- Konjunktívna - 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 Boolovych funkcií

Algebraická normálna forma (skrátene ANF) je jeden z možných spôsobov reprezentácie Boolovych funkcií. Ďalším používaným označením pre ANF je Reed-Mullerova expanzia [11, 12]. Dnešné vedomosti o ANF pomáhali formovať aj Davio [7] a Zhegalkin [15]. Je to jeden z najpoužívanejších sposobov reprezentácie v kryptografií. Podľa definície z knihy *Boolean Functions and Their Applications in Cryptography* [14] 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 $\bigoplus$ reprezentuje operáciu exkluzívny súčet (XOR).

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

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 neuvidíme.

$$1 \quad \bigoplus \quad A \quad \bigoplus \quad B \quad \bigoplus \quad AB \quad \bigoplus \quad ABC \tag{2.3}$$

Ďalej Wu a Feng uvádzajú [14], že počet premenných jedného termu sa nazýva algebraický stupeň termu. 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], afínna týmto bodom prechádzať nemusí. Afínna 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 dodržujeme 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 $3^x 2^y 5^z$ 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 $\bigoplus$ 1. Týmto spôsobom dokážeme previesť do ANF aj funkcie z iných normálnych foriem, prípadne aj z iných reprezentácií.

2.4.1 Splniteľnosť Boolovych funkcií

Problém splniteľnosti Boolovych 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 funkcia v rovnici 2.4, keďže nie je možné, aby premenná mala zároveň hodnotu logickej 0 a logickej 1.

$$f = A \wedge \neg A \tag{2.4}$$

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.

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 nesplniteľný.

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 [10].

Vstupom pre SAT solver je ale konjunktívna normálna forma (CNF), takže dôležitou vecou, na ktorú sa treba zamerať je konverzia ANF na CNF, prípadne ďalšie konverzie.

2.4.2 Konverzie medzi normálnymi formami

Konverzia z ANF na CNF, ako je popísaná v článku od Courtoisa [5], v ktorom sa odkazuje aj na [1], sa skladá z dvoch krokov:

- 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.
- 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

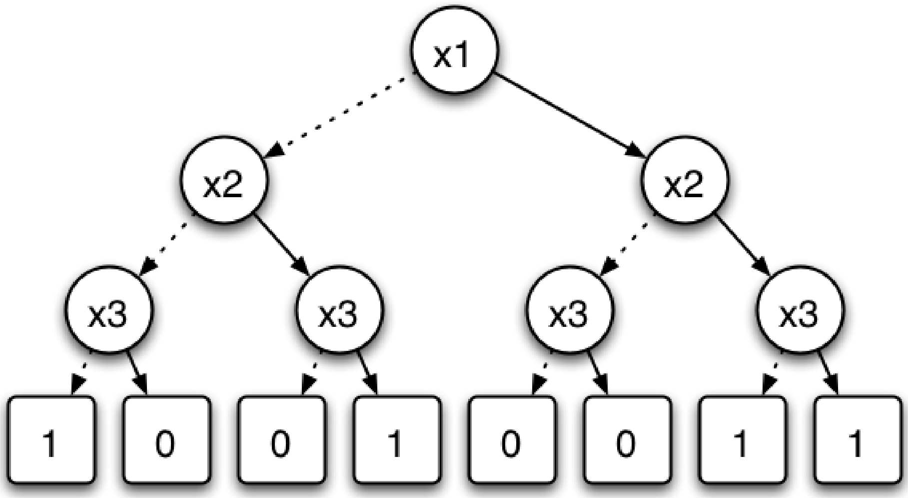
Binárne rozhodovacie diagramy (BDD) sú triedou grafov, ktorá je prevažne využívaná ako dátová štruktúra pre reprezentáciu Boolovych funkcií v dnešnej dobe. Existujú viaceré implementácie, ktoré sú postavené práve na BDD. Používajú sa na riešenie problémov ekvivalencie a splniteľnosti výrazov. Sú veľmi dôležité v oblastiach designu hardvéru a optimalizácie. Informácie v tejto podkapitole sú prevzaté prevažne z [2, 3].

BDD má podobu orientovaného koreňového acyklického grafu. 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 odlíšenie 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 štvorcom a vpísanú majú svoju hodnotu. Low hrany sú označené prerušovanou čiarou, high hrany sú označené plnou čiarou. Obrázok reprezentuje funkciu vyjadrenú pravdivostnou tabuľkou z Tabuľky 2.4.



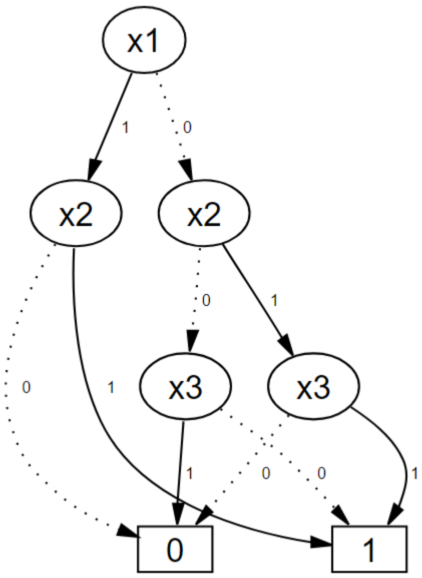
Obr. 2.3: Príklad binárneho rozhodovacieho diagramu. Hodnoty uzlov pochádzajú z pravdivostnej tabuľky v Tabuľke 2.4.

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. 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.

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.



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

Kapitola 3

Existujúce knižnice

Existujú viaceré knižnice vytvorené za účelom manipulácie s Boolovymi funkciami. Nasledujúca kapitola sa zaoberá niektorými vybranými, hlavne tými, ktoré využívajú binárne rozhodovacie stromy (BDD).

Okrem nižšie spomenutých knižníc ešte za zmienku stoja BDD knižnice TiGeR [4] alebo CAL [13];

3.1 Colorado University Decision Diagram Package - CUDD

CUDD je verejne dostupná knižnica[1], 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 doplňujúce 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. Udržuje 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.

Ďalšie informácie o knižnici sa dajú dohľadať v manuáli[2].

3.2 CacBDD

Knižnica CacBDD je verejne dostupná[3] 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 dôjde pamäť. Fun-

[1] http://vlsi.colorado.edu/~fabio/

[2] http://vlsi.colorado.edu/~fabio/CUDD/cudd.pdf

[3] http://www.kailesu.net/CacBDD/

guje trošku 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 rýchlosti.

Ďalšie informácie sú popísané v manuáli[4], kde aj ukázané, že knižnica pracuje rýchlejšie než knižnica CUDD.

3.3 BuDDy

Knižnica BuDDy[5] je ďalšou knižnicou na prácu s Boolovymi funkciami. Je naprogramovaná v jazyku C, ale obsahuje obaľovacie 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, 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[6] pre manipuláciu s Boolovymi funkciami vytvorená v jazyku C#, je vhodná pre využitie v jazykoch z rodiny .NET Framework.

Obsahuje viaceré interné reprezentácie Boolovych 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[7] napísaná v jazyku C++, ktorá bola zamýšľaná na použitie v logických programovacích jazykoch, ale aj v iných. Podobne ako ostatné knižnice využíva ROBDD - Reduced Ordered BDD. Knižnica je zameraná hlavne na pamäťovú efektivitu a na optimalizáciu.

3.6 BDD

Knižnica napísaná v C, primárne zameraná na operačné systémy UNIX, pre prácu mimo UNIX je potrebné upraviť správcu pamäte. Knižnica je rozsahovo veľmi malá[8].

[4] http://www.kailesu.net/CacBDD/CacBDD.pdf
[5] http://buddy.sourceforge.net/manual/main.html
[6] http://dispatcher.swu.bg/BCL/
[7] http://www.cs.unipr.it/Informatica/Tesi/Fabio_Bossi_20090225.pdf
[8] http://www.cs.cmu.edu/afs/cs/project/modck/pub/www/bdd.html

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 všetko okrem x". Takisto používateľ dokáže nastaviť limit na počet uzlov, operácie samé zmažú pamäť ak by museli prekročiť tento limit. Knižnica poskytuje aj možnosť dynamického preusporiadania premenných.

Kapitola 4

Konceptuálny návrh knižnice

V tejto kapitole si bližšie popíšeme návrh knižnice pre manipuláciu s Boolovymi funkciami v ANF. Vysvetlíme si, ako efektívne reprezentovať všetky časti Boolovej funkcie v programe. Takisto si navrhneme všetky potrebné operácie pre reprezentáciu funkcie v ANF.

Základné vlastnosti, ktoré sa od knižnice požadujú, sú efektívna reprezentácia a manipulácia s Boolovymi funkciami. Knižnica by mala obsahovať nástroje použiteľné na vytváranie, úpravu, zobrazovanie a mazanie funkcií a ich jednotlivých súčastí.

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

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 pristúpiť. To by bolo využiteľné, ak by premenné v Boolovej funkcií 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. 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 platených samostatných implementácií hashovacej tabuľky. Nástroj SMHasher[1] a jeho rozšírená verzia[2] 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[3]. 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 všetkych rebríčkoch na najvyšších pozíciách, ale bude si viesť nadpriemerne, pričom bude poskytovať vyššiu prehľadnosť kódu a dobrú dokumentáciu. Podrobnejšie skúšané boli knižnice UTHash[4] a Judy[5], 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

[1] https://github.com/aappleby/smhasher
[2] https://github.com/rurban/smhasher
[3] http://www.tommyds.it/doc/benchmark
[4] http://troydhanson.github.io/uthash/
[5] http://judy.sourceforge.net/

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 obohatený 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 hashovaciu tabuľku, pričom bude potrebné zmeniť volania funkcií len na jedinom mieste, a vo zvyšku súborov našej knižnice zostanú volania bezo zmeny.

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

4.4 Reprezentácia termov vo funkcii

Každá Boolova funkcia v ANF sa skladá z 0 až n 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 koľko ich je dokopy. Keďže medzi všetkým premennými je rovnaká operácia, nie je potrebné si uchovávať informáciu o tom, na ktorej pozícií 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 vyzerať 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.

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 funkcií.

Ďalej je v štruktúre obsiahnutá aj hashovacia tabuľka, obsahujúca všetky hodnoty premenných, ktoré sa v Boolovej funkcií 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 tabuľky, 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.

$$A \wedge A \wedge A \wedge A = A \tag{4.1}$$

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 4.1, 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 počítať novú hodnotu.

Tabuľka 4.1: Pravdivostná tabuľka pre funkciu obsahujúcu tri premenné a dva AND operátory.

x_1	x_2	x_3	$x_1 \wedge x_2 \wedge x_3$
false	false	false	false
false	false	true	false
false	true	false	false
false	true	true	false
true	false	false	false
true	false	true	false
true	true	false	false
true	true	true	true

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.

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[6]. Tento formát bol vyvinutý Microsoftom a je používaný pri vizualizácií 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.

[6] https://msdn.microsoft.com/en-us/library/dn966108.aspx

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="a" Label="a" Size="10" />
      <Node Id="b" Label="b" />
    </Nodes>
    <Links>
      <Link Source="a" Target="b" />
    </Links>
    <Properties>
      <Property Id="Label" Label="Label" DataType="String" />
      <Property Id="Size" DataType="String" />
    </Properties>
  </DirectedGraph>
```

Z textových formátov je veľmi rozšírený formát DOT[7]. 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.

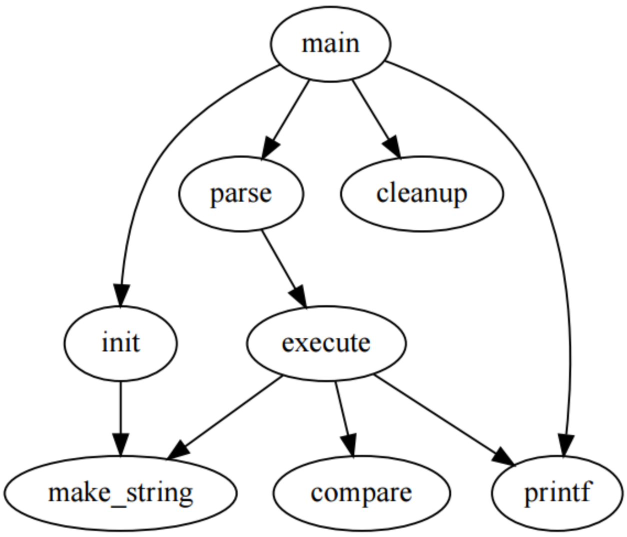
Zdrojový kód 4.6: Ukážka formátu DOT. Obsahuje niekoľko uzlov a prepojení medzi nimi.

```
digraph G {
  main -> parse -> execute;
  main -> init;
  main -> cleanup;
  execute -> make_string;
  execute -> printf
  init -> make_string;
  main -> printf;
  execute -> compare;
}
```

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 Boolovych 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, ktorý nemajú prístup k žiadnemu nástroju pre vykreslenie obrázku vo formáte DOT.

[7] http://www.graphviz.org/pdf/dotguide.pdf



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

Kapitola 5

TODO impl

Kapitola 6

TODO result

Kapitola 7

Záver

Literatúra

[1] 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

[2] 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.

[3] Bryant, R. E.: Symbolic Boolean Manipulation with Ordered Binary-decision Diagrams. *ACM Comput. Surv.*, ročník 24, č. 3, Září 1992: s. 293–318, ISSN 0360-0300.

[4] Coudert, I.; Madre, J. C.; Touati, H.: *TiGeR Version 1.0 User Guide*. Digital Paris Research Lab, 1993.

[5] 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.

[6] Crama, Y.; Hammer, P. L.: *Boolean Functions: Theory, Algorithms, and Applications*. NY, New York: Cambridge University Press, 2011, ISBN 9780521847513, doi:10.1017/CBO9780511852008.

[7] Davio, P.; Deschamps, J. P.; Thayse, A.: *Discrete and Switching Functions*. New York: McGraw-Hill, 1978.

[8] Hazewinkel, M.: *Encyclopaedia of Mathematics*. Springer, 1994, ISBN 9781556080104.

[9] Koppelberg, S.: *Handbook of Boolean algebras Volume 1*. North Holland, 1989, ISBN 044470261X.

[10] Moore, C.; Mertens, S.: *The Nature of Computation*. Oxford University Press, 2011, ISBN 0199233217.

[11] 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.

[12] Reed, I. S.: *A Class of Multiple-Error-Correcting Codes and Their Decoding Scheme*. IRE Trans. Information Theory, vol. 4, 1954, pp. 38-42.

[13] Sanghavi, J. V.; Ranjan, R. K.; Brayton, R. K.; aj.: *High Performance BDD Package by Exploiting Memory Hierarchy*. DAC '96, ACM, 1996, ISBN 0-89791-779-0, 635–640 s.

[14] 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.

[15] Zhegalkin, I. I.: *On the Technique of Calculation the Sentences in Symbolic Logic.* Matem. Sbornik, vol. 34, 1927, pp. 9-28, v ruštine.