VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

KNIHOVNA PRO BINÁRNÍ ROZHODOVACÍ DIAGRAMY

A LIBRARY FOR BINARY DECISION DIAGRAMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Mgr. MARTIN PAULOVČÁK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. ONDŘEJ LENGÁL, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Paulovčák Martin, Mgr.**
Program: Informační technologie
Název: **Knihovna pro binární rozhodovací diagramy**
A Library for Binary Decision Diagrams
Kategorie: Algoritmy a datové struktury

Zadání:

1. Nastudujte datovou strukturu binárních rozhodovacích diagramů (BDD) a její varianty zero-suppressed decision diagrams (ZDD), chain-reduced BDD (CBDD) a ZDD (CZDD), tagged BDD (TBDD) a BDD s hranově specifikovanou redukcí (ESRBDD).
2. Navrhněte knihovnu, která bude podporovat všech šest výše zmíněných typů binárních rozhodovacích diagramů (BDD, ZDD, CBDD, CZDD, TBDD, ESRBDD).
3. Navrženou knihovnu efektivně implementujte. Zaměřte se na vysokoúrovňové (kešování), tak i na nízkoúrovňové (efektivní rozvržení datových struktur) optimalizace.
4. Porovnejte výkon jednotlivých datových struktur na sadě benchmarků.
5. Dosažené výsledky shrňte.

Literatura:

- Randal Bryant. Binary Decision Diagrams. Handbook of Model Checking. Springer. 2018.
- Junaid Babar, Chuan Jiang, Gianfranco Ciardo, and Andrew Miner. Binary Decision Diagrams with Edge-Specified Reductions. In Proc. of TACAS'19. Springer. 2019.
- Randal Bryant. Chain Reduction for Binary and Zero-Suppressed Decision Diagrams. In Proc. of TACAS'18. Springer. 2018.
- Tom van Dijk, Robert Wille, and Robert Meolic. Tagged BDDs: Combining reduction rules from different decision diagram types. In Proc. of FMCAD'17. IEEE. 2017.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Lengál Ondřej, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 31. října 2019

Abstrakt

Cieľom tejto práce je vytvoriť jednoducho použiteľnú knižnicu, ktorá bude poskytovať základné prostriedky pre manipuláciu booleovských funckií prostredníctvom šiestich rôznych variánt binárnych rozhodovacích diagramov — BDD, ZDD, CBDD, CZDD, TBDD a ESRBDD. Knižnica je implementovaná v `ISO C`, používa zatvorené hashovanie, referencovanie uzlov založené na indexoch, garbage collector na princípe `mark and sweep` algoritmu a vytváranie diagramov založené na `depth-first` prechode. Implementované varianty týchto diagramov boli porovnané na benchmarkoch a hoci optimálna voľba varianty závisí na riešenom probléme, vo všeobecnosti sa ako najlepšia voľba z pohľadu veľkosti výsledného grafu a zároveň CPU času ukázali TBDD.

Abstract

The aim of this thesis is to create an easy-to-use library that will provide the basic means for Boolean function manipulation based on six different variants of Binary Decision Diagrams — BDD, ZDD, CBDD, CZDD, TBDD, and ESRBDD. The library is implemented in the `ISO C` programming language, uses closed hashing, index-based node referencing, `mark and sweep` based garbage collector and diagram construction is based on classical `depth-first` traversal. The implemented variants of these diagrams were compared on benchmarks and although the optimal choice of decision diagram variant depends on given problem, in general TBDD proved to be the best choice in terms of the resulting graph size and also CPU time.

Kľúčové slová

binárne rozhodovacie diagramy, booleovské funkcie, ROBDD, BDD, ZDD, CBDD, CZDD, TBDD, ESRBDD

Keywords

binary decision diagrams, boolean functions, ROBDD, BDD, ZDD, CBDD, CZDD, TBDD, ESRBDD

Citácia

PAULOVČÁK, Martin. *Knihovna pro binární rozhodovací diagramy.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Ondřej Lengál, Ph.D.

Knihovna pro binární rozhodovací diagramy

Prehlásenie

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Ondřeje Lengála, Ph.D. a uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Martin Paulovčák
28. mája 2020

Poďakovanie

Touto cestou by som sa rád poďakoval pánu Ing. Ondřeji Lengálovi, Ph.D za ochotu, trpezlivosť a rady, ktorými mi pomohol pri písaní tejto práce.

Obsah

Kapitola 1

Úvod

Boolova algebra je jedným zo základných kameňov informatiky. Okrem toho, že je dôležitá v oblasti digitálnych systémov, mnoho úloh napríklad z oblasti kombinatorickej optimalizácie, matematickej logiky alebo umelej inteligencie sa dajú vyjadriť ako séria operácií na booleovských funkciách. Z toho dôvodu je efektívna reprezentácia a manipulácia booleovských funkcií dôležitá pre mnoho algoritmov a širokú škálu aplikácií. Jedným zo spôsobov ako to dosiahnuť je použiť binárne rozhodovacie diagramy. Ich výhodou napríklad je, že na rozdiel od klasických reprezentácií ako sú pravdivostné tabuľky alebo Karnaughove mapy sa binárne rozhodovacie diagramy dokážu v mnohých prípadoch vyhnúť exponenciálnej zložitosti. Ich zložitosť totiž závisí na veľkosti diagramov a nie na veľkosti domény funkcie. Zásluhou Bryanta a jeho práce *Graph-Based Algorithms for Boolean Function Manipulation* [4] sa zvýšil záujem o túto dátovú štruktúru na báze grafov určenú na manipuláciu booleovských funkcií. To viedlo k vytvoreniu jej viacerých variánt, ktoré sa líšia v rôznych aspektoch.

Cieľom tejto práce je naštudovať dátovú štruktúru binárnych rozhodovací diagramov (BDD) a jej päť ďalších variánt — binárne rozhodovacie diagramy s implicitnou nulou (ZDD), reťazovo redukované BDD (CBDD) a ZDD (CZDD), značené binárne rozhodovacie diagramy (TBDD) a binárne rozhodovacie diagramy s hranovo špecifikovanou redukciou (ESRBDD). Na základe toho je potom ďalej cieľom práce navrhnúť a implementovať knižnicu, ktorá bude poskytovať základné prostriedky pre reprezentáciu, vytváranie a manipuláciu booleovských funkcií vo forme týchto šiestich variánt rozhodovacích diagramov. Napokon budú výkon a vlastnosti jednotlivých variánt experimentálne vyhodnotené. Niektoré práce, ktoré zavádzajú tieto novšie varianty diagramov, ako napríklad *Binary Decision Diagrams with Edge-Specified Reductions* [2] a *Chain Reduction for Binary and Zero-Suppressed Decision Diagrams* [8] síce porovnávajú a experimentálne overujú ich veľkosti, ale nie celkový čas.

Jadro práce je celkovo rozdelené do troch kapitol. Kapitola 2 sa po vysvetlení niektorých dôležitých pojmov v úvode zaoberá teóriou Binárnych rozhodovacích diagramov. Je v nej vysvetlené čo vlastne táto dátová štruktúra vo svojej základnej forme je a aký je jej význam. Takisto vysvetľuje čo znamená usporiadanosť a redukovanosť binárnych rozhodovacích diagramov, a prečo sú tieto dve vlastnosti pre túto dátovú štruktúru dôležité. Ďalšia časť pojednáva o ZDD, ktoré sú založené na rozdielom redukčnom pravidle. Je v nej popísaný ich prínos a rozdiel oproti klasickým BDD. Vo zvyšku tejto kapitoly sú popísané ďalšie štyri verzie rozhodovacích diagramov, ktoré sa snažia využiť výhody oboch vyššie spomínaných verzií — CBDD, CZDD, TBDD a ESRBDD.

Kapitola 3 sa venuje návrhu a implementácii knižnice. Na začiatku je vysvetlené čo je to hashovanie a aké sú základné hashovacie techniky. Ďalej sú popísané základné dátové štruktúry potrebné pre manipuláciu s rozhodovacími diagramami – tabuľka uzlov a tabuľka výsledkov. Kapitola takisto popisuje vnútornú štruktúru uzlov jednotlivých verzií rozhodovacích diagramov, ako sú tieto uzly reprezentované interne a ako sa javia užívateľovi knižnice. Osobitná sekcia je venovaná garbage collectoru a referencovaniu uzlov.

Napokon kapitola 4 pojednáva o experimentoch, ktoré poukazujú na špecifiká jednotlivých verzií rozhodovacích diagramov a ich implementácie. Experimenty sú zamerané na riešenie kombinatorických problémov, reprezentovanie informácií v kompaktnej forme a reprezentovanie typických digitálnych obvodov.

Kapitola 2

Teoretický úvod

2.1 Terminológia

Táto sekcia uvádza základné definície a terminológiu, ktorá bude používaná ďalej. Informácie v tejto sekcii sú prevzaté z [7] a [4].

Nech x označuje **vektor booleovských premenných** $x_1, ..., x_n$. Budeme uvažovať **booleovské funkcie** nad týmito premennými, ktoré budemem zapisovať ako $f(x)$ alebo len f, pokiaľ budú argumenty dostatočne zrejmé. Nech a označuje vektor hodnôt $a_1, ..., a_n$ takých, že pre každé a_i, kde $i = 1, ..., n$ platí $a_i \in \{0, 1\}$. Potom **ohodnotenie** funkcie f vzhľadom na a budeme zapisovať ako $f(a)$. Zatiaľ čo $f(x)$ je funkcia, $f(a)$ je hodnota 1 alebo 0.

Funkcia, ktorá vznikne, keď je nejaký argument x_i funkcie f nahradený konštantou b, sa nazýva **reštrikcia** f a zapisuje sa ako $f|_{x_i \leftarrow b}$. Teda pre akýkoľvek vektor premenných x,

$$f|_{x_i \leftarrow b}(x) = f(x_1, ..., x_{i-1}, x_i, x_{i+1}, ..., x_n) \ . \tag{2.1}$$

Dve reštrikcie funkcie f vzhľadom na premennú x_i sa označujú aj ako **kofaktory** f vzhľadom na x_i. $f|_{x_i \leftarrow 1}$ sa nazýva **pozitívny kofaktor** a $f|_{x_i \leftarrow 0}$ **negatívny kofaktor**. Pomocou dvoch kofaktorov f vzhľadom na x_i každú booleovskú funkciu môžeme rozložiť na:

$$f = (x_i \wedge f|_{x_i \leftarrow 1}) \vee (\neg x_i \wedge f|_{x_i \leftarrow 0}) \ . \tag{2.2}$$

Táto identita sa označuje ako **Shannonov rozvoj** f vzhľadom na x_i. Niekedy sa vyskytuje aj pod názvom **Boolov rozvoj** vzhľadom na to, že ju formuloval už Bool roku 1854 vo svojom spise *An Investigation of the Laws of Thought*.

Niektoré funkcie nemusia závisieť na všetkých svojich argumentoch. Funkcia f je **závislá** na argumente x_i ak platí

$$f|_{x_i \leftarrow 1} \neq f|_{x_i \leftarrow 0} \ . \tag{2.3}$$

Funkcia, ktorá pre všetky možné hodnoty svojich argumentov stále vracia 1 sa nazýva **tautológia** a je označená **1**. Funkcia, ktorá naopak pre všetky hodnoty argumentov vracia 0 sa nazýva **kontradikcia** a je označená **0**.

Funkcia je **splniteľná** ak existuje nejaký vektor a, pre ktorý $f(a) = 1$.

Podobne reštrikcii je definovaná **kompozícia**. Nech f a g sú booleovské funkcie na vektorom premenných x. Kompozícia f a g vzhľadom na x_i, ktorá sa značí ako $f|_{x_i \leftarrow g}$, je funkcia, ktorá vznikne nahradením premennej x_i funkcie f funkciou g. Teda pre vektor premenných x:

$$f|_{x_i \leftarrow g}(x) = f(x_1, ..., x_{i-1}, g(x_1, ..., x_n), x_{i+1}, ..., x_n) \ . \tag{2.4}$$

2.2 Redukované usporiadané binárne rozhodovacie diagramy - BDD

Redukované usporiadané binárne rozhodovacie diagramy (anglicky. Reduced ordered binary decision diagrams), ďalej len BDD, vytvárajú základ pre dátovú štruktúru, ktorá umožňuje efektívnu reprezentáciu a manipuláciu booleovských funkcií. Ich využitie spadá predovšetkým, no nie výlučne do oblasti verifikácie, generovania testov, simulácií zlyhaní a logickej syntézy[9]. Napriek tomu, že reprezentovanie booleovských funkcií ako rozhodovacích grafov má už svoju históriu[1] — ako prvý ich uvádza Lee v roku 1959[12] a neskôr Akers v roku 1978[1] — o ich zviditeľnenie a širšie nasadenie sa zaslúžil až Bryant v roku 1986 svojím článkom "Graph-based algorithms for Boolean function manipulation"[4], formulovaním algoritmov operujúcich nad touto štruktúrou. Jedným z kľúčových prvkov pre tieto algoritmy bolo zavedenie podmienky usporiadanosti (bude vysvetlená nižšie), čím sa výrazne znižujú výpočetné nároky.

Vo všeobecnosti **Binárny rozhodovací diagram** reprezentuje booleovskú funkciu f definovanú nad vektorom premenných $x_1, ..., x_n$ ako **acyklický orientovaný graf majúci dva typy uzlov**. Neterminálne uzly v tomto grafe prezentujú premenné, nad ktorými je funkcia definovaná a listové (terminálne) uzly zasa konštantné hodnoty 1 (pravda) a 0 (nepravda). Podľa vzoru p. Bryanta budeme premennú asociovanú s neterminálnym uzlom v označovať ako $var(v)$, zatiaľ čo pre listový uzol v jeho asociovanú hodnotu ako $val(v)$. Ďalej každý neterminálny uzol má práve dve výstupné hrany bežne označované ako **hi**, čo korešponduje s prípadom kedy má premenná daného uzla hodnotu 1 a **lo**, čo zasa korešponduje s prípadom kedy má premenná daného uzla hodnotu 0. [2] $hi(v)$ a $lo(v)$ označujú dvoch potomkov uzla v. Takto môžeme definovať booleovskú funkciu reprezentovanú BDD tým, že asociujeme funkciu f_v s uzlom v, ktorý bude slúžiť ako koreň daného grafu[7].

Definícia 1 *Binárny rozhodovací diagram G s koreňovým uzlom v reprezentuje booleovskú funkciu f_v definovanú rekurzívne ako:*

1. *Ak v je terminálny uzol:*
   (a) *ak $val(v) = 1$, tak $f_v = 1$*
   (b) *ak $val(v) = 0$, tak $f_v = 0$*
2. *Ak v je neterminálny uzol s indexom i, tak f_v je funkcia:*

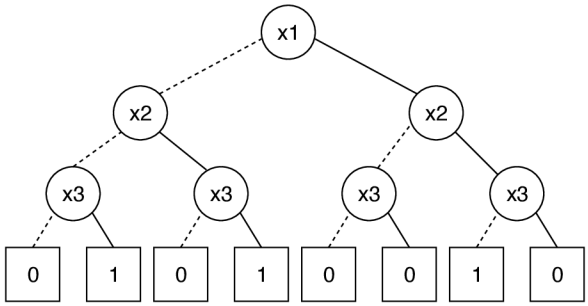
$$f_v = (var(v) \wedge f_{hi(v)}) \vee (\neg var(v) \wedge f_{lo(v)})$$

Vidíme, že každý z uzlov v BDD je sám o sebe booleovská funkcia pričom jeden (alebo aj viac koreňových uzlov pri použití zdieľanej reprezentácie, viď nižšie) sú tie funkcie, ktoré referencuje aplikácia. Takisto si môžeme všimnúť vzťah medzi BDD reprezentáciou nejakej booleovskej funkcie a Shannonovým rozvojom. Dvaja potomkovia uzlu korešpondujú s dvoma kofaktormi vzhľadom na premennú asociovanú s daným uzlom [7]. Na BDD sa tak v podstate môžeme pozerať ako na rekurzívnu dekompozíciu funkcie, ktorú reprezentuje koreň grafu pomocou Shannonovho rozvoja.

[1] v staršej literatúre sa niekedy BDD vyskytujú pod názvom "branching programs"(vetviace programy)

[2] hi sa tradične zobrazuje ako plná čiara a lo ako prerušovaná čiara, neterminálne uzly budeme ako je zvykom kresliť ako krúžky a listové uzlu ako štvorce. Hrany sú v obrázkoch stále orientované smerom dole.

Pre akékoľvek ohodnotenie $(a_1, ..., a_n)$ premenných $(x_1, ..., x_n)$ funkcie f_v je hodnota funkcie určená cestou začínajúcou v koreňovom uzle v a končiacou v jednom z listových uzlov, nasledujúc uzly podľa zadaných hodnôt. Ak je hodnota premennej aktuálneho neterminálneho uzla po ceste rovná 1, pokračuje sa do *hi* potomka, ak je hodnota rovná 0, pokračuje sa do *lo* potomka. Funkčná hodnota je pre toto zadanie napokon hodnota listového uzla, v ktorom cesta skočila. Na obrázku 2.1 môžeme vidieť úplný binárny rozhodovací diagram pre booleovskú funkciu zadanú výrazom $(x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_3)$.

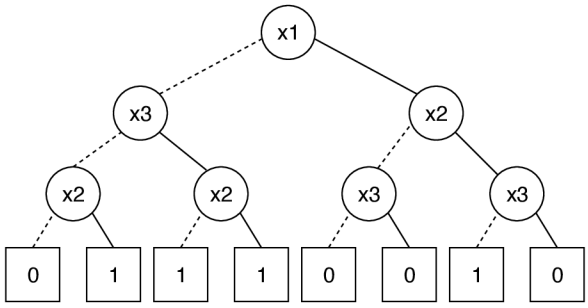


Obr. 2.1: Úplný binárny rozhodovací diagram (binárny rozhodovací strom) reprezentujúci $(x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_3)$.

Každý BDD, ako už napovedá jeho názov, musí navyše spĺňať dve dôležité podmienky - musí byť usporiadaný a redukovaný. Obe tieto vlastnosti si vysvetlíme nižšie.

2.2.1 Usporiadanosť

Usporiadanosť znamená, že existuje nejaká permutácia π nad premennými $x_1, ..., x_n$, teda nejaké usporiadanie premenných danej funkcie a na každej možnej ceste grafom sa tieto premenné musia vyskytovať v stále rovnakom poradí. Zároveň sa žiadna premenná nesmie na tejto ceste vyskytnúť viackrát ako raz[9]. Diagram, ktorý sme si ukazovali na obrázku 2.1 je príkladom usporiadaného binárneho rozhodovacieho diagramu. Obrázok 2.2 zasa zobrazuje neusporiadaný binárny rozhodovací diagram pre nejakú neznámu funkciu troch premenných, pretože na ceste grafom, keď je x_1 rovná 0 sa x_2 vyskytuje pred x_3, avšak na inej ceste, keď je x_1 rovná 1 sa x_2 vyskytuje za x_3.

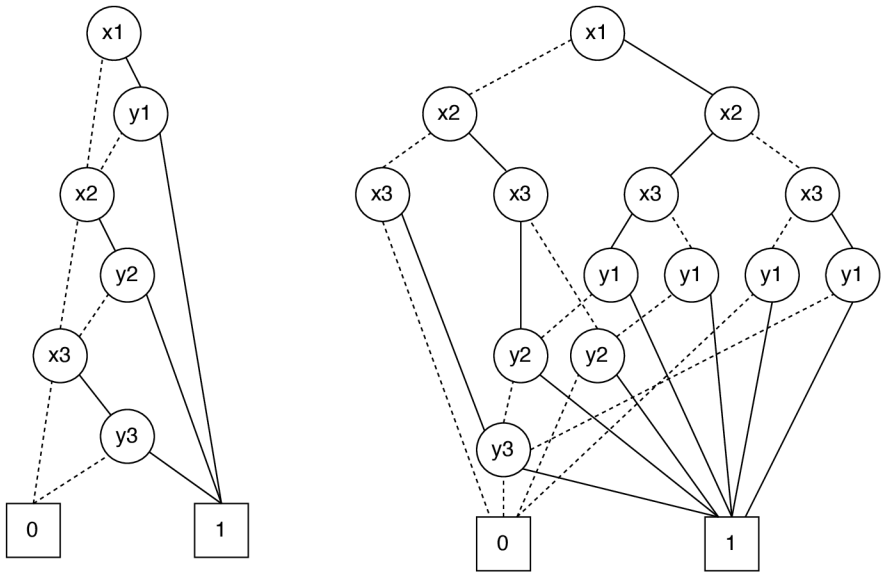


Obr. 2.2: Neusporiadaný binárny rozhodovací diagram pre nejakú booleovskú funkciu troch premenných.

V princípe sa usporiadanie premenných môže zvoliť ľubovoľne a algoritmy budú fungovať stále korektne, avšak ako uvidíme, pre rôzne usporiadania môžeme dostať rôzne BDD. Nevhodné usporiadanie môže mať značný dopad na veľkosť BDD a tým aj na efektivitu

algoritmov. Napríklad na obrázku 2.3 vidíme dva BDD reprezentujúce booleovskú funkciu zadanú výrazom $(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$. BDD naľavo má premenné usporiadané v poradí $a_1, b_1, a_2, b_2, a_3, b_3$, kdežto BDD napravo ich má v poradí $a_1, a_2, a_3, b_1, b_2, b_3$. Túto funkciu by sme mohli zovšeobecniť na funkciu $(a_1 \wedge b_1) \vee ... \vee (a_n \wedge b_n)$. Pre zovšeobecnené usporiadanie premenných $a_1, b_1, ..., a_n, b_n$ dostávame BDD s $2n$ neterminálnymi uzlami, kde n je počet premenných. Na druhej strane pre zovšeobecnené druhé usporiadanie $a_1, ..., a_n, b_1, ..., b_n$ dostaneme BDD s $2(2^n - 1)$ neterminálnymi uzlami. Je zjavné, že pre väčšie n bude dopad na pamäťové nároky a efektivitu algoritmov medzi lineárnym rastom prvého usporiadania a exponenciálnym rastom druhého dramatický[5]. Poznamenajme ešte, že oba BDD sú skutočne usporiadané a redukované.

Väčšina aplikácií využívajúca BDD zvolí nejaké fixné usporiadanie premenných a podľa toho potom konštruuje všetky grafy. Toto usporiadanie sa buď zvolí manuálne alebo nejakou vhodnou heuristickou analýzou systému, ktorý treba reprezentovať. Čo sa týka tých heuristík, tak tie nemusia nevyhnutne nájsť najlepšie možné usporiadanie, nakoľko to nemá vplyv na korektnosť výsledkov. V praxi sa ukazuje, že pokiaľ sa nájde usporiadanie, ktoré neimplikuje exponenciálny rast, tak operácie nad BDD sú stále efektívne.[5, 10]



Obr. 2.3: Dva BDD pre funkciu $(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee (x_3 \wedge y_3)$ demonštrujúce vhodné (vľavo) a nevhodné (vpravo) usporiadanie premenných.

2.2.2 Redukovanosť

To, že graf musí byť redukovaný znamená, že musí navyše spĺňať nasledujúce pravidlá. Treba poznamenať, že tieto pravidlá nijako nepozmenia reprezentovanú funkciu.

1. V celom grafe môže byť maximálne jeden listový uzol s danou hodnotou — teda v celom grafe existujú len dva listové uzly, jeden s hodnotou 1 a jeden s hodnotou 0.

2. V grafe sa nemôže vyskytnúť uzol, pre ktorý by platilo $hi(v) = lo(v)$, teda uzol, ktorého obe výstupné hrany smerujú do jedného potomka. Je zjavné, že takáto premenná nijako neovplyvňuje funkciu reprezentovanú grafom.

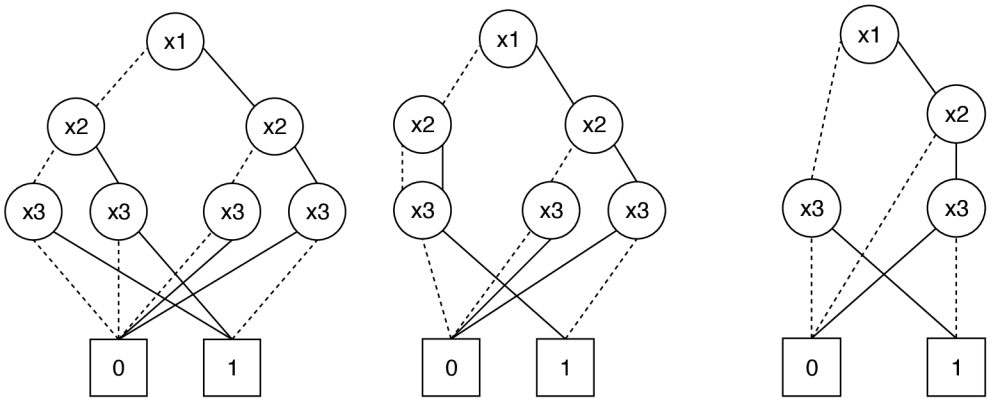
3. Graf nemôže obsahovať izomorfné podgrafy, teda nemôžu byť viaceré neterminálne uzly v a u, pre ktoré by platilo $var(v) = var(u), hi(v) = hi(u), lo(v) = lo(u)$.

Z každého neredukovaného **usporiadaného** binárneho rozhodovacieho diagramu môžeme jednoducho spraviť BDD opakovanou aplikáciou nasledujúcih troch pravidiel. Pravidlá treba aplikovať dovtedy, dokým to už nebude možné, pretože aplikácia jedného pravidla môže vytvoriť priestor pre ďalšiu redukciu.

Redukčné pravidlá:

1. ak listové uzly v a u majú rovnaké hodnoty, teda $val(v) = val(u)$, odstránime jeden z nich a presmerujeme všetky jeho vstupné hrany do toho druhého,
2. ak nejaký uzol v má $hi(v) = lo(v)$, tak ho odstránime a všetky jeho vstupné hrany presmerujem do jeho potomka,
3. ak pre uzly v a u platí $var(v) = var(u), hi(v) = hi(u), lo(v) = lo(u)$, tak jeden z nich odstránime a takisto všetky jeho vstupné hrany presmerujeme do toho druhého.

Na obrázku 2.4 môžeme vidieť príklad redukcie grafu zadaného booleovským výrazom $(x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_3)$. Pôvodný neredukovaný diagram je na obrázku 2.1. Ako to môžeme vidieť na obrázku 2.4a, po aplikácii prvého redukčného pravidla sa nám počet terminálnych uzlov zníži na dva. Aplikáciou tretieho redukčného pravidla, ako to vidíme na obrázku 2.4b odstránime z grafu všetky uzly v, pre ktoré platí $var(v) = var(u) \wedge hi(v) = lo(u) \wedge lo(v) = lo(u)$. V našom ukážkovom prípade sme mohli odstrániť najľavejší uzol prezentujúci premennú x_3. Napokon aplikáciou druhého pravidla odstránime všetky „don't care“ uzly, teda uzly, ktorých obe hrany smerujú do toho istého potomka. Takéto uzly nemajú žiaden vplyv na danú funkciu. Tým dostávame finálny BDD pre zadanú funkciu, ktorý znázorňuje obrázok 2.4c.



(a) Odstránenie duplikátnych terminálnych uzlov.

(b) Odstránenie izomorfných podgrafov.

(c) Odstránenie uzlov, ktorých obe hrany smerovali do jedného potomka.

Obr. 2.4: Redukcia binárneho rozhodovacie diagramu zadaného funkciou $(x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_3)$.

Zmyslom redukcie je odstránenie redundancií - izomorfných podgrafov a premenných, ktoré nemajú na výslednú hodnotu funkcie žiaden vplyv. Bez týchto redukcií by mali diagramy veľkosť exponenciálne závislú na počte premenných a nebolo by možné naplno využívať výhody tejto formy reprezentácie booleovských funkcií. Práve preto je dôležité aby graf

mal čo najmenšiu možnú veľkosť. V najhoršom možnom prípade, kedy nebude možné uplatniť žiadne redukčné pravidlo okrem prvého, bude mať BDD $2^n - 1$ neterminálnych uzlov, kde n je počet premenných funkcie. No v praxi má väčšina funkcií rozumnú veľkosť[7, 10].

Čo sa týka korektnosti, tak redukčné pravidlá je možné aplikovať v hocijakom poradí, stále dostaneme rovnaký BDD, avšak pri postupe od listov smerom hore je možné pri použití vhodného algoritmu túto redukciu uskutočniť v linearitmickom alebo dokonca v lineárnom čase [14]. Postup smerom z dola hore je pre efektivitu dôležitý z toho dôvodu, že nejaké redukčné pravidlo môže byť aplikovateľné na nejaký uzol v po tom, čo aplikujeme nejaké redukčné pravidlo na jeho potomka. Na druhej strane aplikácia nejakého redukčného pravidla na uzol u nikdy nevytvorí príležitosť uplatniť redukčné pravidlo na jeho potomkov. Teda uplatnenie redukčného pravidla na potomka môže vytvoriť príležitosť pre rodiča, ale nikdy nie naopak. Túto situáciu si môžeme všimnúť aj na obrázku 2.4. Uplatniť druhé pravidlo na ľavý uzol reprezentujúci premennú x_2 (obr. 2.4b) môžeme až po tom, čo odstránime jeden z dvoch najľavejších uzlov reprezentujúcich premennú x_3 (obr. 2.4a).

2.2.3 BDD ako kánonická reprezentácia booleovských funkcií

Bryant ukázal, že BDD sú kánonickou reprezentáciou booleovských funkcií. To znamená, že pre dané usporiadanie premenných π má každá booleovská funkcia definovaná nad týmito premennými práve jednu unikátnu reprezentáciu v podobe BDD [4].

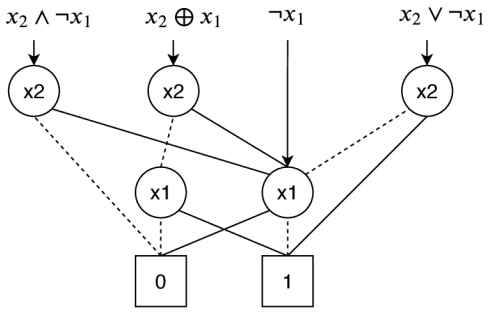
Prezentovanie booleovských funkcií ako BDD má viaceré výhody, z ktorých spomenieme len niektoré:

- dve funkcie sú ekvivalentné vtedy, ak ich BDD sú izomorfné,
- funkcia je splniteľná, ak nie je reprezentovaná jediným terminálnym uzlom s hodnotou 0,
- pre nájdenie jedného ohodnotenia, pre ktoré je funkcia splniteľná stačí nájsť cestu z koreňa do listu s hodnotou 1. Nájdenie tejto cesty si nevyžaduje žiaden backtracking, pretože s výnimkou hrán, ktoré vedú priamo do listu 0, druhé redukčné pravidlo zabezpečuje, že každá hrana je súčasťou cesty vedúcej do listu s hodnotou 1,
- funkcia je tautológia ak je reprezentovaná jediným terminálnym uzlom s hodnotou 1 a naopak, funkcia je kontradikcia ak je reprezentovaná jediným terminálnym uzlom s hodnotou 0,
- ak je nejaká funkcia nezávislá na premennej x, tak BDD reprezentácia tejto funkcie nemôže obsahovať žiadne uzly označené touto premennou,
- v najhoršom možnom prípade takáto reprezentácia booleovskej funkcie môže mať počet uzlov, ktorý je exponenciálne závislý na počte premenných danej funkcie, avšak v skutočnosti mnoho v praxi významných funkcií má relatívne malý BDD [10].

2.2.4 Prezentovanie viacerých funkcií

Z implementačného hľadiska existujú dva spôsoby pre reprezentáciu viacerých funkcií ako BDD - oddelená a zdieľaná. V oddelenej reprezentácii má každá funkcia svoj vlastný BDD, ktorý má práve jeden koreň. V zdieľanej reprezentácii, ktorej príklad môžeme vidieť na obrázku 2.5, je zasa celá kolekcia booleovských funkcií reprezentovaná ako jeden BDD, ktorý ma viacero koreňov (pre každú funkciu jeden). Okrem toho, že zdieľaná reprezentácia má

menšie pamäťové nároky, má navyše tú vlastnosť, že ekvivalencia dvoch funkcií sa dá skontrolovať v konštantnom čase. Dve funkcie sú v zdieľanej reprezentácii BDD ekvivalentné vtedy a len vtedy, ak sú reprezentované tým istým vrcholom. Takisto umožňuje efektívnejšie vykonávanie niektorých ďalších operácií. Jej nevýhodou je, že je potrebná nejaká forma „garbage collectoru“, aby uzly, ktoré už nie sú dosiahnuteľné z akéhokoľvek aktívneho koreňa zbytočne nevyčerpávali miesto. V praxi sa väčšinou uchováva počet referencií na daný uzol, či už to je referencia z rodičovského uzla alebo externá referencia na koreňový uzol. Prirodzene pokiaľ tento počet klesne na nulu, uzol možno uvoľniť.[7]



Obr. 2.5: Zdieľaná reprezentácia viacerých booleovských funkcií ako BDD.

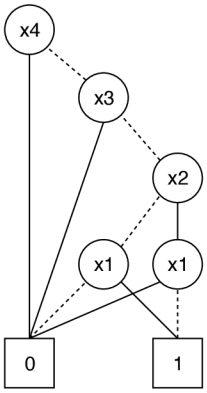
2.3 Binárne rozhodovacie diagramy s implicitnou nulou - ZDD

Binárny rozhodovací diagram s implicitnou nulou (angl. Zero-Suppressed binary decision diagram), ďalej len ZDD je varianta klasických BDD založená na novom redukčnom pravidle, ktorú zaviedol Minato roku 1993 [13]. Tým, že ZDD umožňujú unikátnu a kompaktnú reprezentáciu množín sú obzvlášť vhodné na riešenie kombinatorických problémov, reprezentovanie riedkych množín alebo akýkoľvek iný problém, ktorý sa dá efektívne riešiť manipuláciou riedkych množín bitových vektorov.

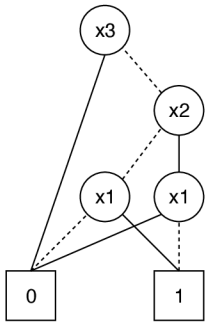
2.3.1 Kombinačné množiny

Informácie v tejto sekcii sú prevzaté z [13]. Kombinácia n objektov môže byť reprezentovaná n bitovým binárnym vektorom $(x_n x_{n-1} ... x_2 x_1)$, kde každý bit $x_k \in \{1, 0\}$ vyjadruje to či korešpondujúci objekt je súčasťou kombinácie alebo nie. Vychádzajúc ďalej z toho, množina kombinácií môže byť reprezentovaná množinou n bitových binárnych vektorov. Takéto množiny sa nazývajú **kombinačné množiny**. Kombinačné množiny môžeme takto považovať za podmnožiny potenčnej množiny n objektov. Môžeme nimi popisovať riešenia kombinatorických problémov a ich manipuláciou môžeme kombinatorické problémy riešiť.

Kombinačnú množinu môžeme reprezentovať booleovskou funkciou s n vstupnými premennými - pre každý bit vektora množiny jedna premenná. Výstupná hodnota tejto funkcie potom vyjadruje či kombinácia špecifikovaná hodnotami vstupných premenných je obsiahnutá v množine alebo nie. Takéto booleovské funkcie sa nazývajú **charakteristické funkcie**. Operácie nad množinami ako zjednotenie, prienik alebo rozdiel môžu byť takto nahradené logickými operáciami nad charakteristickými funkciami. Keďže BDD sa dajú



(a) $\{0001, 0010\}$



(b) $\{001, 010\}$

Obr. 2.6: Kombinačné množiny vyjadrené pomocou BDD.

efektívne použiť na reprezentáciu booleovských funkcií, môžeme ich vďaka charakteristickým funkciám využiť aj na kombinačné množiny. V týchto BDD cesty z koreňového uzla do terminálneho uzla s hodnotou 1 reprezentujú možné kombinácie v množine. Vďaka prvému a tretiemu redukčnému pravidlu, ktoré odstraňujú izomorfné podgrafy a umožňujú takto viacerým kombináciám zdieľať uzly majú takto reprezentované množiny kompaktnú veľkosť. V mnohých praktických prípadoch je veľkosť grafu podstatne menšia ako počet elementov množiny.

Hoci je manipulácia s množinami pomocou BDD efektívna, je tam jedna nepríjemnosť spočívajúca v tom, že forma BDD závisí na počte vstupných premenných. Tým pádom musíme mať fixný počet vstupných premenných ešte pred tým ako sa vygeneruje BDD. Táto nepríjemnosť vyplýva z toho, že v kombinačných množinách je predvolená hodnota premenných 0, pokiaľ je hodnota charakteristickej funkcie rovná 1. To je z toho dôvodu, že takéto premenné (objekty) sa nikdy nevyskytnú v žiadnej kombinácii. Žiaľ takéto premenné nevieme v BDD vynechať, tam vieme zasa vynechať premenné, na ktorých hodnote nezáleží. Z toho dôvodu musíme v prípade riedkych množín vygenerovať mnoho zbytočných uzlov kvôli objektom, ktoré sa v množine nenachádzajú. Práve to bolo motiváciou pre vytvorenie ZDD.

Pre demonštráciu vyššie uvedeného uvažujme veľmi jednoduchý príklad. Chceme pomocou BDD vyjadriť túto kombinačnú množinu: $\{0001, 0010\}$. Máme teda dáta zakódované ako bitové vektory, v našom prípade o dĺžke štyri bity a teda môžeme množinu vyjadriť ako charakteristickú funkciu definovanú nad štyrmi premennými, ktorej výsledná hodnota bude rovná 1, pokiaľ sa bitový vektor odpovedajúci zadaniu premenných nachádza v množine. BDD pre daný množinu môžeme vidieť na obrázku 2.6a. Môžeme si všimnúť, že z koreňa (uzol x_4) vedú do terminálneho uzla s hodnotou 1 len dve cesty a to v prípade kedy je $x_2 = 1$ a všetky ostatné premenné sú rovné 0 a v prípade kedy je $x_1 = 1$ a všetky ostatné premenné sú rovné 0. Ten istý príklad, ale vyjadrený na troch bitoch, teda kombinačná množina $\{001, 010\}$ je zobrazená na obrázku 2.6b. Vidíme teda, že musíme navyše vytvárať uzly pre premenné, ktoré sa nevyskytnú nikdy v žiadnej kombinácii, pretože forma BDD je závislá na počte vstupných premenných. Ako uvidíme ďalej toto nebude platiť pre ZDD vďaka novému redukčnému pravidlu.

2.3.2 Rozdiely ZDD oproti BDD

ZDD sa od BDD líšia len v interpretácii prípadu kedy nejaká hrana preskakuje jednu alebo viac premenných. Ide teda o prípad kedy hrana vychádzajúca z vrcholu v, pre ktorý platí $var(v) = x_i$ vchádza do vrcholu u, pre ktorý platí $var(u) = x_j$ a súčasne platí $j > i + 1$. V prípade BDD takáto hrana indikuje situáciu, kedy je reprezentovaná funkcia nezávislá na vynechaných premenných. Obrázok 2.4c znázorňuje príklad kedy je $f|_{x_1 \leftarrow 0}$ nezávislá na x_2. V prípade ZDD takto vynechaný vrchol znamená, že ak je hodnota jeho premennej rovná 1, tak celková hodnota reprezentovanej funkcie je rovná 0. V prípade kombinačných množín to znamená, že ak má bit vektoru asociovaný s takýmto vrcholom hodnotu 1, tak daný vektor sa v množine nenachádza.[7]

V praxi to znamená, že druhé redukčné pravidlo BDD - pre žiaden uzol v nemôže platiť $hi(v) = lo(v)$, je pri ZDD nahradené pravidlom, že žiaden uzol nemôže mať ako svojho *hi* potomka terminálny uzol s hodnotou 0. Pri redukcii sa takýto uzol odstráni a jeho vstupná hrana sa presmeruje do jeho *lo* potomka. [13] Prvé a tretie pravidlo ostávajú nezmenené. Rovnako ako BDD tak aj ZDD sú kánonickou reprezentáciou booleovských funkcií. To znamená, že pri fixnom usporiadaní premenných má každá funkcia jednoznačnú reprezentáciu v podobe ZDD.

Na obrázku 2.7a a 2.7b máme zobrazené BDD a ZDD (v tomto poradí) pre kombinačnú množinu $\{0001, 0010\}$. Až na dva prípady má každý vrchol u_i v ZDD odpovedajúci vrchol v_i v BDD. Prvý rozdiel je vo vrchole u_4, ktorého obe výstupné hrany smerujú do toho istého potomka. Takéto vrcholy sa v ZDD neodstraňujú. Druhý rozdiel je, že ZDD nemá odpovedajúci vrchol pre v_4 v BDD, keďže nové redukčné pravidlo neumožňuje, aby nejaký vrchol mal ako svojho *hi* potomka terminálny vrchol s hodnotou 0. [7]

Obrázok 2.7c ukazuje ZDD pre kombinačné množiny z obrázku 2.6. Vlastnosťou ZDD je, že ich forma nezávisí na počte vstupných premenných pokiaľ sa jedná o rovnaké kombinačné množiny vyjadrené na rôznych bitových šírkach. Z toho dôvodu je odpovedajúci ZDD rovnaký pre množinu $\{0001, 0010\}$ aj množinu $\{001, 010\}$. Pri ZDD nie je potrebné fixovať počet vstupných premenných pred generovaním grafu. ZDD automaticky „potláčajú“ premenné reprezentujúce objekty, ktoré sa nikdy neobjavujú v žiadnej kombinácii [13]. Z toho dôvodu sú ZDD obzvlášť výhodné pri reprezentácii riedkych kombinačných množín, ktoré majú tieto dve vlastnosti [6]:

1. celkový počet prvkov množiny (bitových vektorov) je podstatne menší ako 2^n, kde n je počet premenných

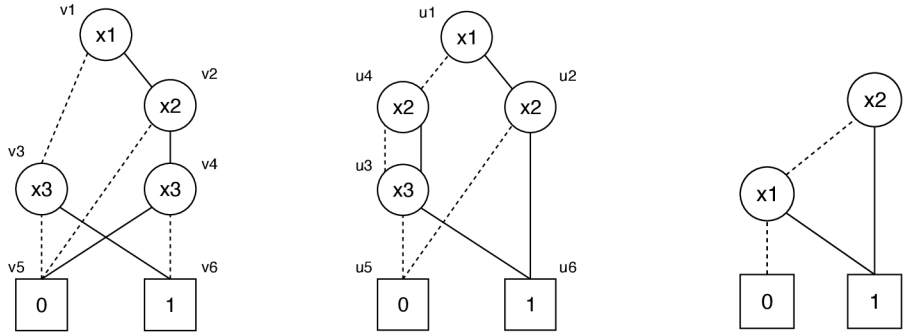
2. samotné bitové vektory majú väčšinu svojich prvkov (bitov) nastavených na 0

Ďalšou výhodou ZDD pri reprezentácii kombinačných množín je, že celkový počet všetkých ciest z koreňa do terminálneho uzla s hodnotou 1 presne odpovedá počtu prvkov v kombinačnej množine. V klasických BDD sa kvôli druhému redukčnému pravidlu táto vlastnosť stráca. [13]

Čo sa týka porovnania pamäťových nárokov BDD a ZDD, Knuth ukázal, že pre akúkoľvek booleovskú funkciu f platí:

$$\frac{N_B}{N_Z} \leq \frac{n}{2} \tag{2.5}$$

$$\frac{N_Z}{N_B} \leq \frac{n}{2} \tag{2.6}$$



(a) $\{110, 001, 011\}$ ako BDD (b) $\{110, 001, 011\}$ ako ZDD (c) množina $\{0001, 0010\}$ z obr. 2.6 ako ZDD

Obr. 2.7: Vľavo a v strede porovnanie BDD a ZDD pre množinu $\{110, 001, 011\}$. Vpravo ZDD reprezentácia množiny $\{0001, 0010\}$.

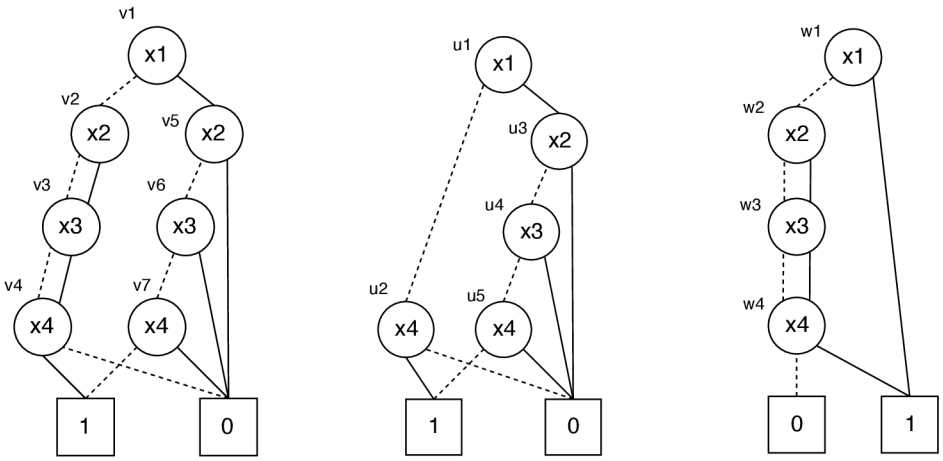
kde N_Z je počet uzlov ZDD, N_B počet uzlov BDD a n je počet vstupných premenných danej funkcie [11]. To znamená že ZDD nejakej funckie môže byť až $(n/2)$ krát kompaktnejší ako BDD a naopak BDD môže byť až $(n/2)$ krát kompaktnejší ako ZDD pre nejakú inú funkciu. Záleží to na funckii, ktorú máme reprezentovať. ZDD sú obzvlášť efektívne ak sa jedná o kombinatorický problém alebo má funkcia dve vyššie spomenuté vlastnosti „riedkosti“. Na druhej strane pri bežných funkciách sú výhodnejšie klasické BDD, obzvlášť ak má daná funkcia väčší počet premenných, ktorých hodnota neovplyvňuje výslednú hodnotu funkcie. [13]

2.4 Reťazovo redukované BDD a ZDD

BDD aj ZDD umožňujú kompaktnú reprezentáciu booleovských funkcií, avšak vzhľadom na to, že obe využívajú rozdielne redukčné pravidlo, ich veľkosť do istej miery závisí na reprezentovanej funkcii. BDD sú vhodné pre funkcie obsahujúce premenné, na ktorých hodnote nezáleží, kdežto ZDD sú vhodné pre funkcie, ktoré sú splniteľné ak má viacero premenných priradenú hodnotu 0. No existujú prípade, pre ktoré by bolo výhodné mať vlastnosti BDD a zároveň aj ZDD. Jedno z možných riešení tohoto problému priniesol Bryant a to tým, že zaviedol reťazové redukcie (*chain reductions*) pre BDD aj ZDD. Nasledujúce príklady ako aj informácie v celej tejto sekcii sú prevzaté z [8].

Obrázok 2.8 znázorňuje tri binárne rozhodovacie diagramy reprezentujúce kombinačnú množinu S definovanú ako $S = \{0001, 0011, 0101, 0111, 1000\}$. Graf A znázornený na obrázku 2.8a, reprezentuje usporiadaný binárny rozhodovací diagram, ktorý neobsahuje žiadne izomorfné podgrafy a ani nadbytočné listové uzly. Nie je to BDD, pretože neboli odstránené uzly, ktoré majú jedného a toho istého potomka (v_2, v_3) a ani ZDD, pretože obsahuje uzly, ktorých *hi* hrana smeruje do listového uzla s hodnotou 0 (v_5, v_6, v_7). Na tomto grafe si môžeme všimnúť, že jeho uzly vytvárajú dva reťazce (*chains*).

Vrcholy v_2, v_3 vytvárajú takzvaný *don't care chain* — neprerušená postupnosť uzlov, ktorých obe hrany smerujú do jedného potomka, ktorý je o jednu úroveň nižšie. Hodnoty týchto uzlov nemajú vplyv na výsledný hodnotu funkcie a ako sme videli sú implicitné v BDD. Obrázok 2.8b zobrazuje tú istú množinu reprezentovanú ako BDD — graf B, v ktorom hrana naľavo smerujúca z vrcholu u_1 do vrcholu u_2 je ekvivalentná *don't care chain* v grafe A. Vrcholy v_5, v_6, v_7 grafu A vytvárajú zasa takzvaný *OR chain*, čo je neprerušená



(a) Graf A (b) Graf B – BDD (c) Graf C – ZDD

Obr. 2.8: Tri grafy reprezentujúce kombinačnú množinu $S = \{0001, 0011, 0101, 0111, 1000\}$.

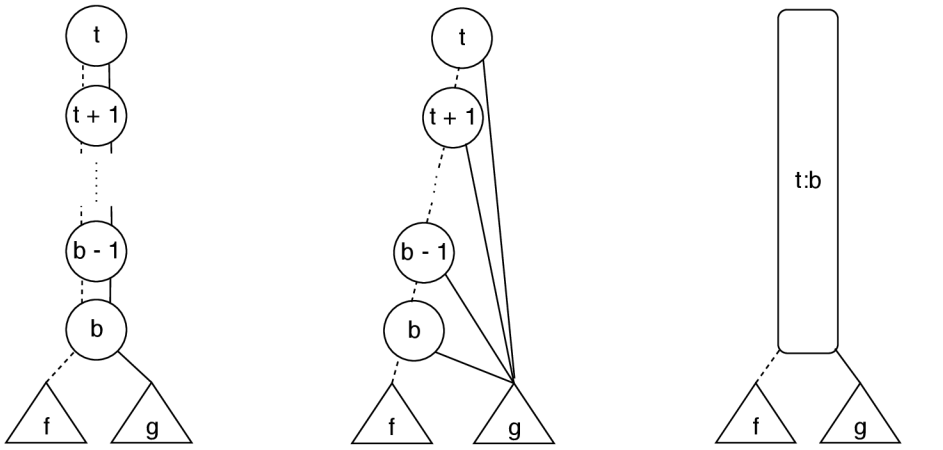
postupnosť uzlov, ktorých *hi* hrany smerujú do jedného a toho istého uzla. V prípade grafu A je to listový uzol s hodnotou 0. *OR chain*, v ktorom všetky *hi* hrany smerujú do listu s hodnotou 0 bude ďalej označovaný ako *zero chain*. Uzly formujúce *zero chain* sú zasa implicitné v ZDD a pokiaľ má niektorý z nich priradenú hodnotu 1, tak výsledná hodnota celej funkcie je 0. Na obrázku 2.8c vidíme ZDD reprezentáciu množiny S označenú ako graf C. Jeho hrana z koreňového uzla w_1 do listového uzla s hodnotou 1 je ekvivalentná so *zero chain* grafu A. Vidíme, že či už je funckia vyjadrená ako BDD alebo ako ZDD stále je prítomný jeden z týchto reťazcov. Zavedenie reťazových redukcií (*chain reductions*), tak ako ich navrhol Bryant, umožňuje BDD aj ZDD využiť obe typy týchto reťazcov a znížiť tak ešte viac pamäťové nároky.

2.4.1 Reťazové redukcie

Obrázky 2.9a a 2.9b demonštrujú zovšeobecnenú formu *don't care chain* a *OR chain*. Tieto reťazce siahajú od úrovne t po úroveň b, pričom platí $1 \leq t < b \leq n$. Za nimi na úrovni väčšej ako b sú potom uzly f a g, ktoré sú nakreslené ako trojuholníky, pretože reprezentujú korene dvoch podgrafov. V *OR chain lo* hrany jednotlivých uzlov smerujú do nasledujúceho uzla v reťazci a *hi* hrany do uzla g. V *don't care chain* obe hrany smerujú do nasledujúceho uzla v reťazci.

Ako bolo spomenuté vyššie, BDD vynechávajú *don't care chains* a ZDD zasa *zero chains* na základe svojich vlastných redukčných pravidiel, ktoré boli opísané v predošlých sekciách. Hlavnou myšlienkou reťazových redukcií je, aby BDD aj ZDD mohli využívať redukčné schopnosti toho druhého a tak kompaktne reprezentovať oba typy reťazcov, čím sa zníži pamäťové nároky i výpočetná doba. To je umožnené tým, že s každým vrcholom sú asociované dve úrovne ako to je na obrázku 2.9c. Každý neterminálny vrchol má dvojicu úrovní $t : b$, pričom platí $1 \leq t < b \leq n$. V reťazovo redukovaných usporiadaných binárnych rozhodovacích diagramoch (anglicky *chain-reduced ordered binary decision diagrams*, ďalej len CBDD) takýto vrchol nahrádza celý *or chain* ako je na obrázku 2.9b. V reťazovo redukovaných binárnych rozhodovacích diagramoch s implicitnou nulou (anglicky *chain-reduced zero suppressed binary decision diagrams*, ďalej len CZDD) takýto vrchol nahrádza *don't*

care chain ako je na obrázku 2.9a. Uzol, v ktorom pre úrovne t a b platí $t = b$ predstavuje štandardný uzol reprezentujúci jednu premennú danej funkcie.



(a) Don't care chain (b) OR chain (c) Zredukovaná forma

Obr. 2.9: Don't care chains, ktoré ostávajú v ZDD a zero chains — špeciálny prípad or chains, ktoré sú zasa prítomné v BDD môžu byť prezentované v zredukovanej forme.

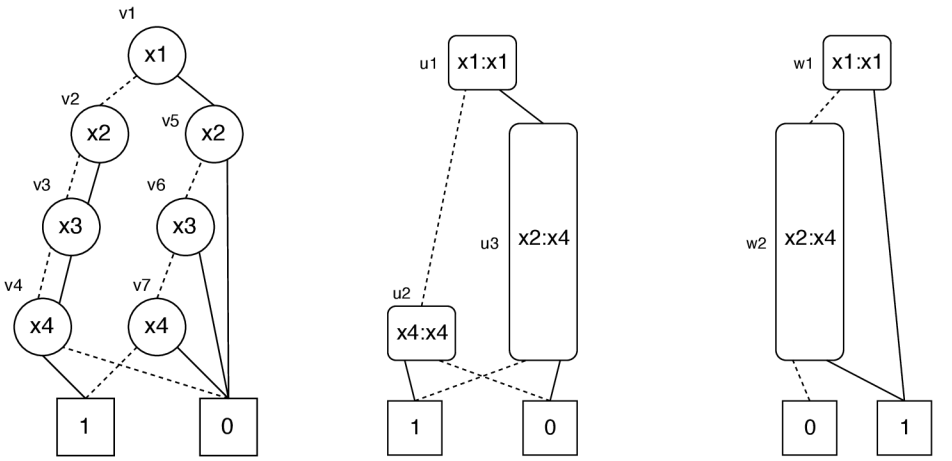
Obrázok 2.10 demonštruje efekt reťazových redukcií. Graf A je pôvodný graf z 2.8a bez týchto redukcií. Na obrázku 2.10b je CBDD reprezentácia množiny S. Vidíme, že vrcholy v_2 a v_3 sú tam implicitne ako je to v klasickom BDD z obrázka 2.8b, avšak vrcholy v_5, v_6 a v_7 formujúce *zero chain*, ktoré v 2.8b ostali ako u_3, u_4, u_5 sú v CBDD nahradené jedným vrcholom u_3. CZDD reprezentácia množiny S je na obrázku 2.10c. Opäť môžeme vidieť, že vrcholy v_5, v_6 a v_7 sú rovnako ako v ZDD z obrázka 2.8c aj v CZDD implicitne, ale *don't care chain* pozostávajúci z vrcholov v_2 a v_3, ktoré v ZDD ostali ako w_2 a w_3, je v CZDD začlenený do v_4 a formuje tak vrchol w_2. Tieto nové uzly sú nakreslené ako ovály, aby sa zvýraznilo, že siahajú cez viac ako jednu úroveň, avšak všetky uzly v CBDD aj CZDD majú dvojicu úrovní, viď napríklad u_1 alebo w_1.

2.5 Značené redukované usporiadané binárne rozhodovacie diagramy

Rovnaký problém ako CBDD a CZDD — teda ako využiť redukčné schopnosti klasických BDD aj ZDD, ale iným spôsobom riešia aj značené redukované usporiadané binárne rozhodovacie diagramy (anglicky tagged binary decision diagrams, ďalej len TBDD), ktoré zaviedli van Dijk, Wille a Meolic. Nasledujúce informácie sú prevzaté z ich práce v [16].

2.5.1 Značenie hrán

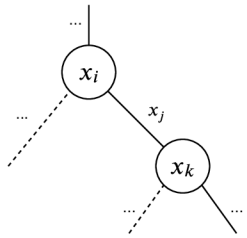
TBDD kombinujú redukčné pravidlá BDD aj ZDD, teda okrem odstránenia nadbytočných listových uzlov a izomorfných podgrafov — čo sú pravidlá spoločné pre obe varianty, redukujú aj vrcholy, ktorých obe hrany smerujú do jedného potomka (pravidlo BDD) a takisto redukujú aj vrcholy, ktorých *hi* hrana smeruje do listu s hodnotou 0 (pravidlo ZDD). Týmto dosahujú ešte kompaktnejšiu reprezentáciu booleovských funkcií. Pre akúkoľvek booleovskú funkciu je zaručené, že jej TBDD nebude mať viac uzlov ako BDD alebo ZDD reprezentácia danej funkcie [8]. Pravidlo osobitné pre BDD budeme ďalej označovať ako *bdd-pravidlo*



(a) Graf A (b) Graf D — CBDD (c) Graf E — CZDD

Obr. 2.10: Ukážka reťazových redukcií na kombinačnej množine $S = \{0001, 0011, 0101, 0111, 1000\}$. CBDD aj CZDD využívajú redukčné schopnosti toho druhého a redukujú oba typy reťazcov.

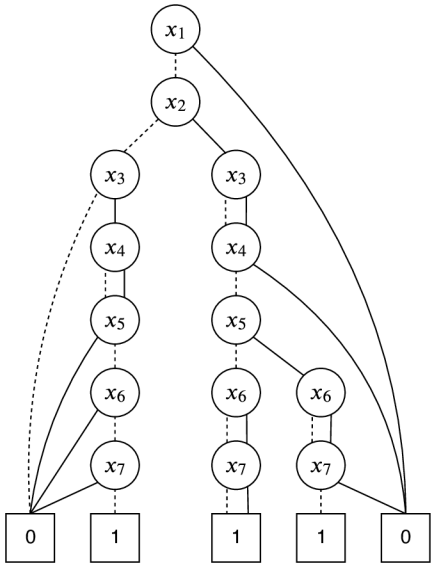
a pravidlo osobitné pre ZDD ako *zdd-pravidlo*. Na rozlíšenie toho, ktoré z týchto dvoch pravidiel bolo na odstránenie uzla alebo uzlov použité sa používa značenie hrán. Tieto značky (*tags*) sú na každej jednej hrane grafu a môžu byť dvojakého typu — môže to byť označenie premennej, teda x_i, kde $1 \leq i \leq n$ a n je počet premenných funkcie alebo symbol $\perp$. Hrany smerujúce do neterminálnych uzlov môžu byť označené len premennou x_i, hrany smerujúce do listov môžu byť označené premennou alebo symbolom $\perp$. Premenná použitá ako značka hrany musí byť podľa usporiadania premenných v grafe vždy *za* premennou uzla, z ktorého hrana vychádza a súčasne *pred* alebo *rovnaká* ako premenná uzla, do ktorého hrana smeruje, pokiaľ sa nejedná o listový uzol. Premenné, ktoré v grafe chýbajú a sú pred značkou x_i, boli odstránené na základe *bdd-pravidla* zatiaľčo chýbajúce premenné, ktoré sú za značkou x_i alebo rovnaké ako táto značka, boli odstránené na základe *zdd-pravidla*. Názornejší popis týchto pravidiel je v popise k obrázku 2.11, ktorý zobrazuje všeobecný fragment TBDD. Čo sa týka značky $\perp$, tak všetky chýbajúce premenné pred touto značkou boli odstránené na základe *bdd-pravidla*.



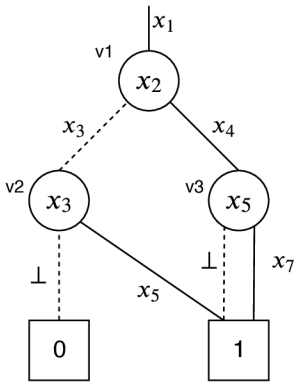
Obr. 2.11: Fragment TBDD s usporiadaním premenných $x_i < x_j \leq x_k$. Všetky uzly x_m, pre ktoré platí $x_i < x_m < x_j$ boli odstránené podľa *bdd-pravidla*. Všetky uzly x_n, pre ktoré platí $x_j \leq x_n < x_k$ boli odstránené na základe *zdd-pravidla*.

Na obrázku 2.12b môžeme vidieť konkrétny príklad TBDD, ktorý reprezentuje funkciu

$$f(x_1, ..., x_8) = \overline{x_1}\overline{x_2}x_3\overline{x_5}x_6\overline{x_7} \vee \overline{x_1}x_2\overline{x_4}x_5 \vee \overline{x_1}x_2\overline{x_4}x_5\overline{x_7} \quad .$$



(a) Graf A — všeobecný binárny rozhodovací diagram



(b) TBDD

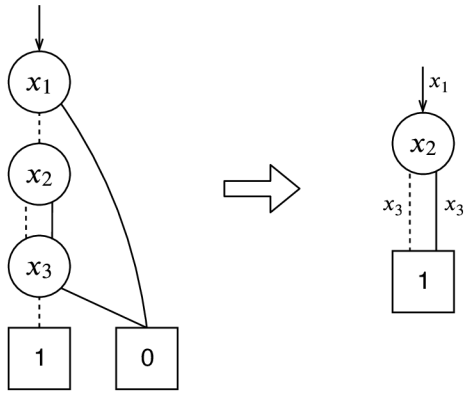
Obr. 2.12: Dva grafy reprezentujúce $f(x_1, ..., x_8) = \overline{x_1}\overline{x_2}x_3\overline{x_5}\overline{x_6}\overline{x_7} \vee \overline{x_1}x_2\overline{x_4}\overline{x_5} \vee \overline{x_1}x_2\overline{x_4}x_5\overline{x_7}$.

Vedľa neho na 2.12a je pre porovnanie graf A — všeobecný binárny rozhodovací diagram, z ktorý neobsahuje izomorfné podgrafy a reprezentuje tú istú funkciu. Pre lepšiu prehľadnosť boli v ňom ponechané viaceré listové uzly. Môžeme si napríklad všimnúť značku x_5, na hrane z vrcholu v_2 do listu s hodnotou 0. Táto značka znamená, že na tejto cesto grafom bol odstránený uzol premennej x_4 na základe *bdd-pravidla* a uzly premenných x_5, x_6 a x_7 na základe *zdd-pravidla*. Ďalej značka $\perp$ na *lo* hrane z vrcholu v_3 do listu 1 znamená, že uzly premenných x_6 a x_7 boli odstránené na základe *bdd-pravidla*.

TBDD sa snažia aplikovať redukčné pravidlá v maximálnej možnej miere, avšak v prípadoch kedy sa uzly odstránené *bdd-pravidlom* a *zdd-pravidlom* striedajú to nie je úplne možné. V takomto prípade, aby bolo možné rozlíšiť, ktorý uzol bol odstránený podľa akého pravidla je potrebné ponechať buď každý posledný uzol, ktorý by vyhovoval *zdd-pravidlu*, alebo každý prvý uzol, ktorý by vyhovoval *bdd-pravidlu*. Obrázok 2.13 demonštruje jednu takúto situáciu. V tomto prípade boli odstránené uzly vyhovujúce *zdd-pravidlu*, a preto bolo potrebné ponechať uzol, ktorého obe hrany smerujú do jedného potomka. *Zdd-pravidlo* bolo použité preto, lebo v praxi sa tieto grafy redukujú z dola nahor počas ich konštrukcie a kedykoľvek nejaký skonštruovaný uzol vyhovuje niektorému pravidlu, tak to dané pravidlo sa hneď uplatní.

2.6 Binárne rozhodovacie diagramy s hranovo špecifikovanou redukciou

Binárne rozhodovacie diagramy s hranovo špecifikovanou redukciou (anglicky *Edge-specified reduction binary decision diagrams*, ďalej len ESRBDD) sú ďalšou variantou rozhodovacích diagramov, ktoré sa snažia o vyššiu efektivitu tým, že využívajú redukčné pravidlá BDD aj ZDD. Podľa tvrdení ich autorov — Babar, Jiang, Ciardo a Miner — sú ESRBDD oproti variantám, ktoré sme spomínali vyššie (CZDD, CBDD a TBDD) konceptuálne jednoduch-



Obr. 2.13: TBDD, ktorého dva uzly majú list s hodnotou 0 ako *hi* potomka a jeden uzol, ktorého obe hrany smerujú do jedného potomka. Tento prostredný uzol nemohol byť odstránený lebo by nebolo možné rozlíšiť aké pravidlo bolo kedy použité.

šie, ich uzly zaberajú menej miesta, neuprednostňujú nejaké redukčné pravidlo pred inými a sú flexibilnejšie v tom, že v budúcnosti je možné pomerne jednoducho pridávať ďalšie redukčné pravidlá [2]. Nasledujúce informácie sú takisto prevzaté z [2].

Tak ako v predošlých variantách rozhodovacích diagramov aj ESRBDD sú definované ako acyklický orientovaný graf, v ktorom sú dva terminálne uzly, s hodnotami 0 a 1, a neterminálne uzly s dvoma hranami (*hi* a *lo*) smerujúcimi do uzlov na nižších úrovniach podľa daného usporiadania premenných. No narozdiel od variánt spomínaných vyššie hrana v ESRBDD je definovaná ako dvojica $< \kappa, p >$, kde κ je redukčné pravidlo z množiny pravidiel $\{S, L_0, H_0, X\}$ a p je uzol, do ktorého hrana smeruje. ESRBDD používa pojem **krátka** a **dlhá** hrana. V prípade, že hrana nepreskakuje žiadnu úroveň, teda spája uzly, ktoré sa líšia práve o jednu úroveň, jedná sa o krátku hranu, ktorá vyžaduje pravidlo $\kappa = S$. V prípade, že hrana preskakuje jednu alebo viac úrovní, jedná sa o dlhú hranu, ktorej $\kappa \in \{L_0, H_0, X\}$. Redukčné pravidlo hrany špecifikuje jej význam. H_0 (zero-suppressed) znamená, že uzly, ktoré hrana preskakuje boli odstránené na základe pravidla špecifického pre ZDD. Pravidlo X znamená, že vynechané uzly boli zredukované na základe pravidla špecifického pre BDD. L_0 (one-suppressed) je nové pravidlo analogické k zero-suppressed, ktoré redukuje uzly pokiaľ ich *lo* hrana smeruje do listového uzla s hodnotou 0. Napriek tomu, že S je v množine redukčných pravidiel, takto označená hrana je klasická hrana, ktorá nič neredukuje. Pre lepšiu názornosť si môžeme BDD predstaviť ako ESRBDD kde redukčné pravidlá pre hrany sú obmedzené na $\{S, X\}$. Podobne si môžeme predstaviť ZDD ako ESRBDD kde množina redukčných pravidiel je obmedzená na $\{S, H_0\}$. Verzii ESRBDD kde je množina pravidiel obmedzená na $\{S, L_0\}$ nekorešponduje žiadna doposiaľ známa varianta rozhodovacích diagramov.

V ESRBDD je booleovská funkcia reprezentovaná hranou — teda dvojicou $< \kappa, p >$. Formálne môžeme povedať:

Definícia 2 *Nech $i \in \{1, ..., n\}$, kde n je počet premenných a nech $lvl(p)$ značí úrovež uzla p. Nech $(x_{i:n})$ je skrátený zápis $(x_i, ..., x_n)$ a $p[hi], p[lo]$ sú hi a lo hrany uzla p. Ďalej predpokladajme, že do koreňa ESRBDD vedie visiaca hrana s pravidlom $\kappa = S$, pričom v prípade koreňa toto pravidlo nemá žiaden význam. Hrana ESRBDD $< \kappa, p >$ definuje rekurzívne booleovskú funkciu $f_{<\kappa,p>}(x_{1:n})$ ako:*

$$f_{<\kappa,p>}(x_{i:n}) = \begin{cases} val(p) & \text{ak } p \text{ je list,} \\ (var(x_i) \wedge f_{<\kappa,p>}(x_{i+1:n})) \vee (\neg var(x_i) \wedge f_{<\kappa,p>}(x_{i+1:n})) & \text{ak } lvl(p) < i,\ \kappa = X, \\ 0 \vee (\neg var(x_i) \wedge f_{<\kappa,p>}(x_{i+1:n})) & \text{ak } lvl(p) < i,\ \kappa = H_0, \\ (var(x_i) \wedge f_{<\kappa,p>}(x_{i+1:n})) \vee 0 & \text{ak } lvl(p) < i,\ \kappa = L_0, \\ (var(p) \wedge f_{p[hi]}(x_{i+1:n}) \vee (\neg var(p) \wedge f_{p[lo]}(x_{i+1:n})) & \text{inák} \end{cases}$$

Ďalej môžeme povedať, že ESRBDD je redukovaný, ak pre neho platia nasledujúce obmedzenia:

1. neobsahuje žiadne izomorfné podgrafy,
2. neobsahuje uzly, na ktorých je funkcia nezávislá,
3. neobsahuje žiadne uzly, ktorých *hi* hrana smeruje do listu s hodnotou 0,
4. neobsahuje žiadne uzly, ktorých *lo* hrana smeruje do listu s hodnotou 0 (nové pravidlo špecifické pre ESRBDD),
5. pre každú hranu $e =< \kappa, 0 >$ musí platiť $\kappa \in \{S, X\}$.

Posledné obmedzenie zamedzuje v redukovanom ESRBDD hrany $< H_0, 0 >$ a $< L_0, 0 >$. To je z toho dôvodu, že $f_{<H_0,0>} \equiv f_{<L_0,0>} \equiv f_{<X,0>} \equiv 0$ a to bráni kanonicite redukovaných ESRBDD. Preto autori zvolili $< X, 0 >$ ako jednotnú reprezentáciu všetkých takýchto dlhých hrán.

Kapitola 3

Návrh a implementácia knižnice

Knižnica implementuje šesť verzií binárnych rozhodovacích diagramov popísaných vyššie (BDD, ZDD, CBDD, CZDD, TBDD, ESRBDD). Všetky implementované verzie teda predstavujú usporiadané a redukované rozhodovacie diagramy pričom redukčné pravidlá sa uplatňujú hneď počas toho ako sa vytvárajú jednotlivé uzly. To znamená, že v knižnici nie je možné vytvoriť úplný (neredukovaný) alebo neusporiadaný rozhodovací diagram. Všetkých šesť verzií využíva zdieľanú reprezentáciu tak ako bola popísaná v 2.2.4. V čase písania nie sú implementované hrany s atribútmi (attributed edges)[1] a ani dynamické re-usporiadanie premenných. To znamená, že výkon aplikácie do veľkej miery závisí na tom aké usporiadanie zvolí užívateľ knižnice. Knižnica je navrhnutá tak, aby poskytovala základné prostriedky pre reprezentáciu, vytváranie a manipuláciu booleovských funkcií vo forme rozhodovacích diagramov. Je napísaná v ISO C s tým, že ju je možné používať aj v C++.

V nasledujúcich sekciách budú podrobnejšie popísané základné dátové štruktúry a algoritmy použité pri implementácii knižnice. No najprv spomenieme niečo o hashovacích tabuľkách a hashovacích funkciách, pretože tie tvoria základ implementovaných diagramov.

3.1 Hashovacie tabuľky a hashovacie funkcie

Hashovacia tabuľka (tiež nazývaná aj ako hash mapa) je dátová štruktúra implementujúca abstraktný dátový typ asociatívne pole, ktoré asociuje kľúče s odpovedajúcimi hodnotami. Na toto mapovanie kľúčov na hodnoty sa využíva hashovacia funkcia a bežne sa deje v dvoch krokoch:

$$hash = hashfunc(key)$$

$$index = hash \ \% \ table\_size$$

kde *hashfunc* je hashovacia funkcia, *key* je unikátny kľúč identifikujúci položku, ktorú chceme zahashovať, *table_size* je veľkosť hashovacej tabuľky a % je operácia modulo – zvyšok po delení.

Hashovacia funkcia na základe kľúča vypočíta hash, na ktorý sa potom použije modulo operátor s veľkosťou hashovacej tabuľky. Tým sa určí index, na ktorom sa v tabuľke nachádza hodnota k danému kľúču. Pre dobrú hashovaciu funkciu platí, že hashe sú rovnomerne rozdelené medzi indexy tabuľky a podobné kľúče majú od seba vzdialené hashe. Ideálne

[1] Hrany s atribútmi predstavujú techniku používanú v rozhodovacích diagramoch, ktorá umožňuje určiť komplement zadaného diagramu v konštantnom čase

hashovacia funkcia priradí každému kľúču unikátny hash, avšak väčšina hashovacích funkcií nie je ideálna, čo spolu s operáciou modulo môže spôsobovať kolízie. To znamená, že pre viac rôznych kľúčov dostaneme rovnaký index do tabuľky.

Podľa spôsobu riešenia kolízií sa hashovacie tabuľky delia na dva základné druhy:

1. **otvorené hashovanie**,
2. **zatvorené hashovanie**, tiež známe aj ako otvorené adresovanie.

Za špeciálne varianty sa dá považovať dokonalé hashovanie, kedy kolízie nevznikajú alebo sa jednoducho ignorujú a hodnoty sa prepisujú, no v takom prípade nie je zaručené, že uložená hodnota v tabuľke bude.

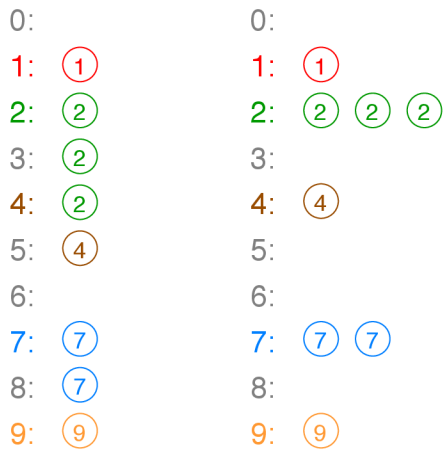
Pri otvorenom hashovaní sú jednotlivé položky, ktoré sa zahashovali na rovnaké miesto tabuľky usporiadané do nejakej zreťazenej dátovej štruktúry. Typicky je to buď jednosmerne zviazaný zoznam (linked list) alebo do binárny vyhľadávací strom. Hashovacia tabuľka tak neobsahuje priamo zahashované hodnoty, ale ukazatele na začiatok tohoto zreťazeného zoznamu kolízií. Pre dobrú hashovaciu tabuľku platí, že tieto zreťazené zoznamy kolízií sú čo najmenšie. Výhodou otvoreného hashovania je, že je flexibilnejšie pri operáciách vkladania a odstraňovania prvkov, nevznikajú v nej zhluky (clusters) a faktor naplnenia môže byť väčší ako 1. Faktor naplnenia je pomer medzi aktuálnym počtom uložených prvkov a veľkosťou hashovacej tabuľky.

Pri zatvorenom hashovaní sú všetky položky uložené priamo v poli hashovacej tabuľky. Keď pri vkladaní prvku nastane kolízia, nejakým vhodným algoritmom sa určí náhradné miesto. To sa opakuje dovtedy, dokým sa nepodarí nájsť voľné miesto. Pri vyhľadávaní sa postupuje podobne. Jednotlivé položky tabuľky sa prechádzajú rovnakou sekvenciou ako pri vkladaní dokým sa nenájde hľadaná položka alebo voľné miesto, čo znamená, že daná položka v tabuľke nie je. Výhodou tejto formy hashovania je, že sa dá lepšie využiť lokalita odkazov a cache. Takisto z pohľadu pamäte okrem samotnej tabuľky nemá žiadne režijné náklady – zaberá len toľko pamäte, koľko má hashovacia tabuľka. Pretože všetky položky sú uložené priamo v poli tabuľky, faktor naplnenia nemôže prekročiť hodnotu 1 a čím viac sa blíži k 1, tým viac degraduje výkon tabuľky. Preto ak faktor naplnenia presiahne určitý limit, je nutné celý obsah tabuľky re-hashovať do väčšej tabuľky.

Na Obrázku 3.1 môžeme vidieť znázornený rozdiel medzi otvoreným a zatvoreným hashovaním. Zatvorené hashovanie (vľavo) používa konkrétne techniku „linear probing“, kde je pri vzniku kolízie ako náhradné miesto vybrané to, ktoré je najbližšie voľné za pôvodnou pozíciou. Pri zatvorenom hashovaní (vpravo) sú kolízie zreťazené do viazaného zoznamu.

3.2 Tabuľka uzlov – Unique table

Tabuľka uzlov je pre redukované rozhodovacie diagramy základná dátová štruktúra, ktorá zabezpečuje, aby sa nikdy nevytvorili dva ekvivalentné uzly – z toho dôvodu sa nazýva „unique table“. Tým zaručuje jednu z podmienok redukovanosti – konkrétne že sa v grafe nemôžu vyskytovať žiadne izomorfné podgrafy. Vďaka tomu je tak zároveň zabezpečená kanonicita diagramov. Podľa odporúčania v [3] namiesto dvoch oddelených štruktúr pre túto tabuľku unikátnych uzlov a rozhodovací diagram ako orientovaný acyklický graf (DAG) sú tabuľka uzlov a rozhodovací diagram zlúčené do jednej štruktúry. Táto dátová štruktúra je implementovaná ako hashovacia tabuľka. Takto sú jednotlivé obsadené položky hashovacej tabuľky pospájané do jedného alebo viacerých multi-koreňových grafov. No zároveň sú takto niektoré zdanlivo náhodné uzly grafu pospájané do kolíznych reťazcov hashovacej tabuľky.



Obr. 3.1: Názorná ukážka riešenia kolízií medzi zatvoreným hashovaním (vľavo), konkrétne sa jedná o techniku zvanú „linear probing“ a otvoreným hashovaním (vpravo), kde sú kolízie zreťazené do viazaného zoznamu.

Tradične, no nie výlučne sa tabuľka uzlov implementuje ako open hashing s kolíznymi reťazcami, kvôli väčšej flexibilite[15], no v mojej implementácii som použil closed hashing z nasledujúcich dôvodov:

1. lepšie sa dá využiť lokalita odkazov a cache pamäte,
2. nie sú potrebné ukazatele, pretože uzol sa dá referencovať pomocou indexov. Vďaka tomu môžu byť v štruktúre uzla položky pre **low** a **high** potomkov 32 bitové čísla udávajúce index do poľa hashovacej tabuľky. Takto sa dá na 64 bitových architektúrach, kde ukazateľ zaberá 8 Bytov do značnej miery zmenšiť veľkosť uzla.
3. Napokon pri zatvorenom hashovaní nie sú potrebné časté alokácie pamäte spojené s vytváraním nových uzlov. Alokácia pamäte sa deje len pri inicializácii hashovacej tabuľky a jej prípadnom zväčšovaní.

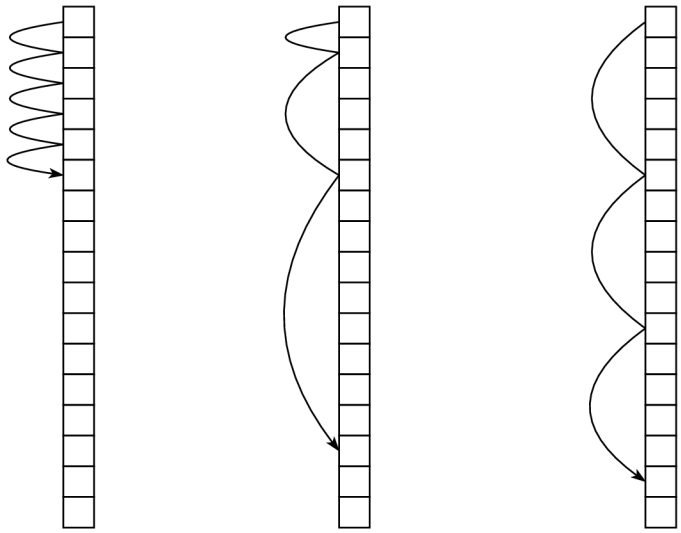
3.2.1 Technika riešenia kolízií

Ako konkrétne techniky riešenia kolízií som implementoval **linear probing**, **quadratic probing** a **double hashing**. Linear probing rieši kolízie tým, že ako náhradné miesto pre kolízny prvok vyberie miesto o jednu pozíciu ďalej. Ak ani to nie je voľné, zväčšuje indexy o 1 dokým sa voľné miesto nenájde. Vyhľadávacia sekvencia má teda tvar $i+1$, $i+2$, $i+3$, ..., kde i je pôvodný index, ktorý vrátila hashovacia funkcia. Táto technika má veľmi dobré využitie cache, keďže kolízie sú uložené v pamäti lineárne. Napríklad do 64 Bytovej cacheliny sa takto zmestia štyri BDD uzly, ktoré majú v implementácii knižnice pri použití tejto techniky veľkosť 16 Bytov. Nevýhodou je, že ako sa postupne zväčšujú kolízne reťazce, tak tieto reťazce začnú pôsobiť ako sieť, ktorá zachytáva ďalšie prvky, ktoré budú takto pridané na koniec reťazca a ďalej túto sieť zväčšovať. Tento jav sa nazýva primárne zhlukovanie (primary clustering). Aby som následky tohoto javu zmiernil, pridal som do štruktúr pre uzly položku `next`, ktorá sa v tradičnej implementácii tejto techniky neobjavuje. Tá obsahuje index ďalšieho prvku, ktorý spôsobil kolíziu a vytvára tak kolízne reťazce. To umožňuje v istých prípadoch preskočiť spomínané zhluky a urýchliť tak vyhľadávanie a vkladanie

prvkov. Pokiaľ v tejto položke `next` je hodnota rôzna od 0, pokračuje sa indexom, ktorý je v nej uložený. Ináč sa pokračuje klasickým zvyšovaním indexu o 1.

Quadratic probing zväčšuje medzery medzi kolíznymi prvkami kvadraticky. To znamená, že vyhľadávacia sekvencia má tvar $i + 1^2$, $i + 2^2$, $i + 3^2$, ..., kde i je pôvodný index, ktorý vrátila hashovacia funkcia. Takto sú medzery medzi kolíznymi prvkami väčšie a nevznikajú primárne zhluky. Avšak kvôli tomu, že vyhľadávanie náhradného miesta je nezávislé na kľúčoch hashovaného prvku, je táto technika stále náchylná na sekundárne zhlukovanie. To znamená, že ak sa pri slabej hashovacej funkcii viac prvkov zahashuje na rovnaké miesto, každý prvok pri hľadaní voľnej pozície kráča v stopách toho predošlého až dokým nedôjde na koniec kolízneho reťazca.

Double hashing používa na nájdenie náhradnej pozície pre kolízny prvok druhú hashovaciu funkciu. Tá určuje krok, ktorým sa pri hľadaní bude zväčšovať pôvodný index, ktorý vrátila prvá hashovacia funkcia. Vyhľadávacia sekvencia má teda tvar $i + 1 * j$, $i + 2 * j$, $i + 3 * j$, ..., kde i je pôvodný index, ktorý vrátila prvá hashovacia funkcia a j je výsledok druhej hashovacej funkcie. Vzhľadom na to že j je závislé na kľúči hashovaného prvku, je táto technika odolná voči primárnemu aj sekundárnemu zhlukovaniu. Nevýhodou je, že má z týchto troch techník najhoršie využitie cache a lokality odkazov. Zároveň musí platiť, že veľkosť hashovacej tabuľky a výsledok druhej hashovacej funkcie musia byť nesúdeliteľné čísla. To sa dá dosiahnuť napríklad tým, že veľkosť tabuľky bude prvočíslo. Obrázok 3.2 znázorňuje vyhľadávacie sekvencie pre spomínané techniky zatvoreného hashovania.



Obr. 3.2: Názorná ukážka postupu pri vyhľadávaní prvku alebo voľného miesta pri vkladaní nového prvku pri „linear probing“ (vľavo), „quadratic probing“ (v strede) a „double hashing“ (vpravo), pri ktorom krok je určený druhou hashovacou funkciou.

V nasledujúcich tabuľkách sú zhrnuté experimenty s hashovacími technikami v troch kategóriách:

1. celkový CPU čas (ľavý stĺpec),
2. percentuálny podiel „cache miss“ k celkovému počtu cache referencií (stredný stĺpec),
3. počet vykonaných inštrukcií (pravý stĺpec).

Jednotlivé čísla v tabuľkách udávajú pomer priemernej nameranej hodnoty hashovacej techniky v danom riadku tabuľky k hashovacej technike daného stĺpca tabuľky. Každý typ rozhodovacie diagramu má vlastné tri tabuľky pre každú z kategórií. Číslo menšie ako 0 tak znamená, že technika na príslušnom riadku dosiahla lepší výsledok ako technika v príslušnom stĺpci. Testy boli vykonané s takými počiatočnými veľkosťami tabuliek uzlov, aby nebolo potrebné ich dynamické zväčšovanie. Prebehli na sade benchmarkov, ktorá obsahovala problém 12 kráľovien (viď 4.1), slovník na Macintosh systémoch (viď 4.2), tri náhodne vygenerované funkcie v konjunktívnej normálnej forme vo formáte DIMACS CNF zo zbierky SATLIB, ktoré mali 50 premenných a 218 klauzulí a digitálne obvody c432, c3540, c499, c6288 zo sady ISCAS85 (viď 4.3) na systéme s procesorom 2.6 GHz Intel Xeon a 32 GB pamäte, a operačným systémom Debian 10.

Čas:

BDD:

	LP	QP	DH
LP	1.000	0.953	0.891
QP	1.054	1.000	0.934
DH	1.132	1.072	1.000

ZDD:

	LP	QP	DH
LP	1.000	0.967	0.920
QP	1.035	1.000	0.952
DH	1.088	1.051	1.000

CBDD:

	LP	QP	DH
LP	1.000	1.067	1.003
QP	0.940	1.000	0.940
DH	1.001	1.064	1.000

CZDD:

	LP	QP	DH
LP	1.000	1.057	1.000
QP	0.949	1.000	0.946
DH	1.003	1.058	1.000

TBDD:

	LP	QP	DH
LP	1.000	1.043	0.973
QP	0.960	1.000	0.933
DH	1.030	1.072	1.000

Cache-miss:

	LP	QP	DH
LP	1.000	0.976	0.983
QP	1.026	1.000	1.007
DH	1.020	0.995	1.000

	LP	QP	DH
LP	1.000	0.979	0.980
QP	1.023	1.000	1.001
DH	1.022	1.000	1.000

	LP	QP	DH
LP	1.000	0.993	1.004
QP	1.011	1.000	1.010
DH	1.003	0.991	1.000

	LP	QP	DH
LP	1.000	0.988	0.991
QP	1.013	1.000	1.002
DH	1.012	0.998	1.000

	LP	QP	DH
LP	1.000	1.002	0.999
QP	0.998	1.000	0.996
DH	1.002	1.004	1.000

Inštrukcie:

	LP	QP	DH
LP	1.000	0.976	0.926
QP	1.026	1.000	0.949
DH	1.081	1.054	1.000

	LP	QP	DH
LP	1.000	0.978	0.938
QP	1.023	1.000	0.959
DH	1.066	1.043	1.000

	LP	QP	DH
LP	1.000	0.990	0.922
QP	1.010	1.000	0.931
DH	1.085	1.074	1.000

	LP	QP	DH
LP	1.000	0.974	0.887
QP	1.027	1.000	0.910
DH	1.128	1.099	1.000

	LP	QP	DH
LP	1.000	1.005	0.940
QP	0.995	1.000	0.935
DH	1.065	1.070	1.000

Čas: **Cache-miss:** **Inštrukcie:**

ESRBDD:

	LP	QP	DH
LP	1.000	0.953	0.894
QP	1.050	1.000	0.937
DH	1.123	1.068	1.000

	LP	QP	DH
LP	1.000	0.977	0.981
QP	1.025	1.000	1.005
DH	1.020	0.995	1.000

	LP	QP	DH
LP	1.000	0.995	0.946
QP	1.005	1.000	0.950
DH	1.058	1.052	1.000

Z vykonaných experimentov vyplynulo, že pri dostatočne veľkých počiatočných veľkostiach tabuliek uzlov mala najlepšie výsledky pre BDD, ZDD a ESRBDD technika „linear probing“ (LP) a pre CBDD, CZDD a TBDD „quadratic probing“ (QP). To je pravdepodobne z toho dôvodu, že tieto diagramy majú pri LP uzly veľké 24 bytov (CBDD a CZDD) a 20 bytov (TBDD), no pri QP sa všetky tri zmenšia na 16 bytov. Pri týchto veľkostiach tabuliek uzlov bola lokalita odkazov približne rovnaká pre všetky tri hashovacie techniky. „Double hashing“ (DH) bola vo všetkých prípadoch najpomalšia kvôli tomu, že je na ňu potrebný najväčší počet inštrukcií, čo je pochopiteľné, nakoľko vypočítať duhý hash je náročnejšie ako inkrementovať index alebo vynásobiť dve čísla. Pri menších počiatočných veľkostiach tabuliek uzlov vykazuje najlepšie výsledky LP pre všetkých šesť typov diagramov. Príloha A ukazuje ako sa mení CPU čas a cache-miss pre jednotlivé typy diagramov so zmenšovaním tabuliek uzlov.

3.2.2 Hashovacia funkcia

Čo sa týka hashovacích funkcií, implementoval som viacero možností:

- hashovaciu funkciu pre kombináciu dvoch čísel prevzatú z Boost C++ knižníc (ďalej označenú skratkou BST), ktorá má tvar:

  $$f(k_1, k_2) = k_1 \oplus (k_2 + 0x9e3779b9 + (k_1 << 6) + (k_1 >> 2))$$

  kde $\oplus$ znamená operáciu XOR a $>>$, a $<<$ sú bitové posuny. V prípade, že má kľúč tri hodnoty, najprv sa zahashujú prvé dve a s ich výsledkom sa znova zahashuje tretia. Podobne sa postupuje pri viacerých hodnotách.

- Cantorova párovacia funkcia (ďalej označená skratkou CAN), ktorá sa používa na zakódovanie dvoch prirodzených čísiel do jedného. V prípade, že má kľúč viac hodnôt, postupuje sa ako pri predošlej funkcii. Táto funkcia má tvar:

  $$f(k_1, k_2) = \frac{1}{2}(k_1 + k_2)(k_1 + k_2 + 1) + k_2$$

- Fowler-Noll-Vo hashovacia funkcia verzia 1a (FNV-1a, ďalej označená skratkou FNV), ktorá hashuje pole bytov. V prípade, že má kľúč viac hodnôt, sú uložené v pamäti za seba a funkcia ich vníma ako jedno súvislé pole bytov. Jedná sa o nekryptografickú hashovaciu funkciu, ktorá používa operácie násobenia a XOR. Jej implementácia je dostupná na http://www.isthe.com/chongo/tech/comp/fnv/#alt-FNV-source.

- Upravená FNV-1a (ďalej označená skratkou DEF), ktorá neiteruje cez byty, ako originálna verzia ale cez celočíselné hodnoty, ktoré tvoria jeden zložený kľúč.

- Klasická DJB2 (ďalej označená skratkou DJB) nekryptografická hashovacia funckia, ktorej algoritmus vyzerá nasledovne:

```
unsigned long hash = 5381;
int c;
while (c = *str++)
        hash = ((hash << 5) + hash) + c; /* hash * 33 + c */
```

- A napokon PJW hashovacia funkcia (ďalej označená skratkou PJW). Jedná sa takisto o nekryptografickú hashovaciu funkciu, ktorej algoritmus je dostupný na https://www.cs.hmc.edu/~geoff/classes/hmc.cs070.200101/homework10/hashfuncs.html

V nasledujúcich tabuľkách sú zhrnuté experimenty s hashovacími funkciami. Jednotlivé čísla v tabuľkách na ľavej polovici strany udávajú priemerné pomery časov hashovacej funkcie daného riadka k hashovacej funkcii daného stĺpca pre daný typ rozhodovacieho diagramu. V tabuľkách na pravej polovici strany sú uvedené priemerné pomery počtov kolízií hashovacej funkcie daného riadka k funkcii daného stĺpca pre daný typ rozhodovacieho diagramu. Číslo menšie ako 0 tak znamená, že funkcia na príslušnom riadku dosiahla lepší výsledok ako funkcia v príslušnom stĺpci.

Priemerné pomery časov:

Priemerné pomery počtov kolízií:

BDD:

	DEF	CAN	BST	FNV	DJB	PJW
DEF	1.000	0.982	0.991	0.861	0.714	0.910
CAN	1.019	1.000	1.009	0.879	0.729	0.928
BST	1.010	0.991	1.000	0.871	0.722	0.919
FNV	1.164	1.144	1.154	1.000	0.824	1.056
DJB	1.487	1.462	1.475	1.269	1.000	1.338
PJW	1.104	1.086	1.096	0.948	0.780	1.000

	DEF	CAN	BST	FNV	DJB	PJW
DEF	1.000	0.987	0.957	0.940	0.474	0.369
CAN	1.032	1.000	0.971	0.975	0.491	0.363
BST	1.066	1.035	1.000	1.010	0.494	0.373
FNV	1.073	1.066	1.035	1.000	0.503	0.400
DJB	3.335	3.362	3.117	3.160	1.000	0.989
PJW	35.36	27.29	27.19	35.17	13.39	1.000

ZDD:

	DEF	CAN	BST	FNV	DJB	PJW
DEF	1.000	0.992	1.011	0.820	0.682	0.744
CAN	1.008	1.000	1.019	0.826	0.686	0.750
BST	0.990	0.982	1.000	0.811	0.676	0.734
FNV	1.237	1.226	1.249	1.000	0.822	0.908
DJB	1.582	1.566	1.602	1.267	1.000	1.194
PJW	4.708	4.697	4.730	3.758	3.405	1.000

	DEF	CAN	BST	FNV	DJB	PJW
DEF	1.000	1.508	0.943	0.877	0.511	0.441
CAN	0.868	1.000	0.852	0.771	0.410	0.404
BST	1.085	1.842	1.000	0.944	0.578	0.466
FNV	1.166	1.801	1.094	1.000	0.583	0.482
DJB	3.812	4.475	3.731	3.412	1.000	2.254
PJW	436.8	452.1	433.5	296.4	100.5	1.000

CBDD:

	DEF	CAN	BST	FNV	DJB	PJW
DEF	1.000	0.978	1.008	0.886	0.737	0.917
CAN	1.023	1.000	1.030	0.906	0.755	0.937
BST	0.993	0.971	1.000	0.880	0.733	0.910
FNV	1.133	1.108	1.141	1.000	0.826	1.036
DJB	1.454	1.424	1.465	1.274	1.000	1.324
PJW	1.094	1.070	1.101	0.966	0.801	1.000

	DEF	CAN	BST	FNV	DJB	PJW
DEF	1.000	1.003	0.901	1.164	0.605	0.399
CAN	0.997	1.000	0.898	1.162	0.604	0.399
BST	1.216	1.218	1.000	1.588	0.774	0.416
FNV	0.919	0.922	0.866	1.000	0.539	0.395
DJB	2.940	2.945	2.782	3.136	1.000	1.002
PJW	5.350	5.367	3.726	8.257	3.464	1.000

Priemerné pomery časov:

CZDD:

	DEF	CAN	BST	FNV	DJB	PJW
DEF	1.000	0.983	1.102	0.880	0.752	0.950
CAN	1.017	1.000	1.122	0.895	0.766	0.968
BST	0.913	0.898	1.000	0.801	0.682	0.864
FNV	1.140	1.122	1.254	1.000	0.852	1.079
DJB	1.382	1.360	1.513	1.209	1.000	1.310
PJW	1.066	1.049	1.169	0.933	0.799	1.000

TBDD:

	DEF	CAN	BST	FNV	DJB	PJW
DEF	1.000	0.994	1.015	0.896	0.692	0.778
CAN	1.006	1.000	1.021	0.901	0.696	0.782
BST	0.987	0.981	1.000	0.884	0.686	0.768
FNV	1.120	1.113	1.135	1.000	0.767	0.865
DJB	1.571	1.560	1.600	1.393	1.000	1.233
PJW	2.146	2.132	2.174	1.922	1.649	1.000

ESRBDD:

	DEF	CAN	BST	FNV	DJB
DEF	1.000	1.002	1.060	0.801	0.835
CAN	0.998	1.000	1.058	0.800	0.833
BST	0.947	0.949	1.000	0.759	0.790
FNV	1.259	1.262	1.335	1.000	1.043
DJB	1.207	1.210	1.279	0.960	1.000

Priemerné pomery počtov kolízií:

	DEF	CAN	BST	FNV	DJB	PJW
DEF	1.000	0.903	0.748	0.889	0.879	0.352
CAN	1.181	1.000	1.088	0.986	1.219	0.371
BST	1.851	1.777	1.000	1.743	1.313	0.624
FNV	1.199	1.018	1.096	1.000	1.238	0.382
DJB	2.302	2.227	1.296	2.228	1.000	1.195
PJW	5.596	4.638	5.149	4.567	6.107	1.000

	DEF	CAN	BST	FNV	DJB	PJW
DEF	1.000	0.991	0.809	0.767	0.686	0.320
CAN	1.009	1.000	0.816	0.775	0.690	0.321
BST	1.904	1.892	1.000	1.661	1.541	0.361
FNV	1.384	1.372	1.182	1.000	0.876	0.502
DJB	2.606	2.576	2.279	1.719	1.000	0.727
PJW	264.0	253.5	198.0	233.8	103.7	1.000

	DEF	CAN	BST	FNV	DJB
DEF	1.000	0.912	0.683	0.872	0.726
CAN	1.171	1.000	0.684	0.918	0.754
BST	29.21	15.21	1.000	8.244	5.278
FNV	1.503	1.166	0.689	1.000	0.806
DJB	2.103	1.554	0.799	1.287	1.000

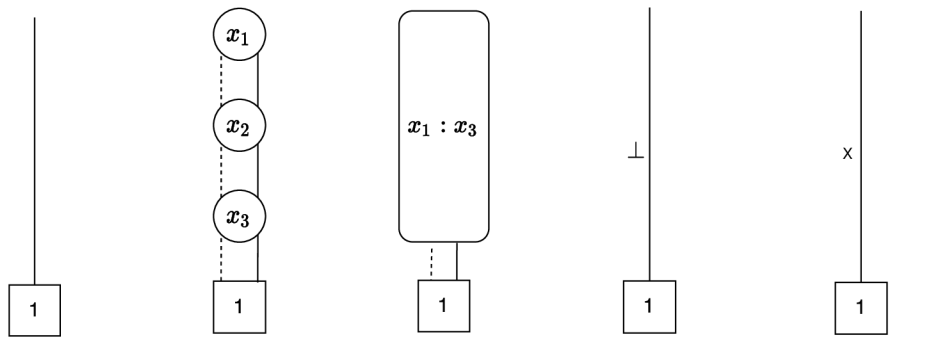
Z vykonaných experimentov vyplýva, že z pohľadu CPU času je najlepšou voľbou pre BDD hashovacia funkcia DEF[2] (hashovacia funkcia na štýl FNV-1a pracujúca s číslami a nie bytmi). Pre ostatné typy diagramov dosahovali experimenty najlepší čas pri použití hashovacej funkcie BST, pričom tesne za ňou boli DEF alebo CAN. Z pohľadu počtu kolízií bola pre BDD, CZDD, TBDD a ESRBDD najlepšia DEF, pre ZDD mala najlepší priemerný výsledok CAN a pre CBDD mala najmenší priemerný počet kolízií klasická FNV-1a funkcia. Z pohľadu času i počtu kolízií najhoršie dopadla PJW funkcia. Pri ESRBDD táto funkcia nie je uvedená pre obzvlášť slabé výsledky, žiaden z testov totiž neskončil v čase do pol hodiny. Podstatne horšie z pohľadu kolízií dopadla pri ESRBDD oproti ostatným typom funkcia BST. To môže byť z dôvodu, že dva z troch prvkov zloženého kľúča sú 64 bitové čísla, v ktorých z vrchných 32 bitov sú využité len 2. Pri ostatných funkciách, obzvlášť DEF a CAN to nie je problém, zrejme preto, že sú multiplikatívneho charakteru, pričom BST je implementovaná len ako bitové posuvy a sčítanie. To je výhodné z pohľadu rýchlosti, no nie z pohľadu kolízií. Hashovacie funkcie multiplikatívneho charakteru v spojení s tabuľkami, ktorých veľkosti sú zarovnané na mocniny 2 sú odporúčané aj v [15]. Všetky funkcie sú v implementácii knižnice a je ich možné meniť definovaním príslušných makier pri preklade, no predvolená je DEF.

[2]Pre význam skratiek viď 3.2.2.

3.2.3 Vytváranie uzlov

Kľúčom, a teda aj vstupom pre hashovaciu funkciu, ktorý jednoznačne identifikuje uzol rozhodovacieho diagramu je trojica: **index premennej**, ktorú uzol zaobaľuje a **low** a **high** potomkovia daného uzla, ktorí sú reprezentovaní indexom do hashovacej tabuľky. Pred každým vytvorením nového uzla sa najprv skontroluje či sa daný uzol už v tabuľke nenachádza. Ak áno, nič sa nevytvára a použije sa existujúci uzol. Ak daný uzol ešte neexistuje, tak sa vytvorí a uloží do tabuľky na miesto určené hashovacím algoritmom na základe spomínaného kľúča.

Tým, že užívateľ nemá cez API knižnice priamo prístup k tabuľke uzlov, a ani k funkcii vytvárajúcej uzly, je stále zabezpečené, že **low** a **high** potomkovia uzla, ktorý má byť vytvorený už v tabuľke uzlov sú. Booleovská konštanta **0** má vždy fixný identifikátor a teda aj pozíciu v hashovacej tabuľke – nachádza sa na indexe 0 a to u všetkých implementovaných rozhodovacích diagramov. Pre konštantu **1** je situácia odlišná. Čo sa týka BDD a CBDD tam rovnako ako **0** aj **1** má fixný identifikátor a pozíciu v tabuľke uzlov – na indexe 1. No v ZDD a CZDD nie je jednoducho možné vyjadriť booleovskú konštantu **1** ako jeden listový uzol s hodnotou 1 ako to je možné u BDD alebo CBDD a s malou modifikáciou i u TBDD a ESRBDD, kvôli tomu ako ZDD a CZDD interpretujú hrany presahujúce jeden alebo viac uzlov. Jeden listový uzol s hodnotou 1, ako to je na obrázku 3.3a totiž v týchto dvoch typoch diagramov znamená funkciu, v ktorej sa všetky premenné danej domény nachádzajú v negovanej podobe. Pri ZDD je treba na vyjadrenie tejto konštanty jeden listový uzol a toľko neterminálnych uzlov koľko je kardinalita domény. Pri CZDD na to stačí jeden listový uzol a jeden neterminálny uzol, ktorý bude siahať od úrovne prvej premennej až po poslednú. Z toho dôvodu nie je v knižnici konštanta **1** pre tieto dva typy diagramov reprezentovaná premennou ako je to u ostatných verzií diagramov, ale funkciou. Čo sa týka TBDD a ESRBDD tam na to postačuje jeden listový uzol so značkou $\perp$ na vstupnej hrane u TBDD a s pravidlom X na vstupnej hrane u ESRBDD. Na základe, ktorý tvoria tieto dve konštanty sú potom vytvárané funkcie reprezentujúce jednoduché premenné a z nich ďalšie komplexnejšie funkcie. Na Obrázku 3.3 sú znázornené vyjadrenia konštanty **1** pre jednotlivé druhy rozhodovacích diagramov.



(a) BDD, CBDD (b) ZDD (c) CZDD (d) TBDD (e) ESRBDD

Obr. 3.3: Vyjadrenie booleovskej konštanty 1 v rôznych typoch rozhodovacích diagramov pre doménu (x_1, x_2, x_3).

Funkcia vytvárajúca jednotlivé uzly má zároveň na starosti kontrolu, či nie je potreba celú tabuľku uzlov re-hashovať do väčšej tabuľky. Stále pre pridaním nového uzla sa skontroluje, či by tento prírastok nespôsobil prekročenie maximálneho povoleného faktoru naplnenia. Ak áno tabuľka sa re-hashuje do novej tabuľky, ktorá je dvakrát väčšia ako tá pôvodná.

Prednastavený limit pre faktor naplnenia je na základe [17] stanovený na **0.68**, no túto hodnotu je možné zmeniť pomocou podmieneného prekladu: `-D MAX_LOAD_FACTOR=XY`, kde XY je daný limit.

3.3 Uzly diagramov a ich štruktúra

Interne sú jednotlivé uzly diagramov BDD, ZDD, CBDD a CZDD reprezentované 32 bitovým číslom (`uint32_t`), ktoré vyjadruje index do hashovacej tabuľky obsahujúcej uzly. Na tomto indexe v tabuľke je potom uložený uzol so všetkými potrebnými informáciami. Štruktúra uzlov BDD a ZDD, a teda položka ich tabuliek uzlov vyzerá nasledovne:

```
struct bdd_zdd_s {
    uint32_t lo, hi;
    uint32_t next;
    uint32_t var_idx : 19;
    uint32_t ref_count : 12;
    uint32_t mark : 1;
} bdd_zdd_t
```

Celkovo uzly BDD a ZDD zaberajú v pamäti 16 Bytov. `lo` a `hi` sú indexy do hashovacej tabuľky pre **low** a **high** potomkov daného uzla. `next` je index do hashovacej tabuľky, na ktorom sa nachádza ďalší uzol, ktorý pri hashovaní vytvoril kolíziu. `var_idx` je index premennej, ktorú uzol zaobaľuje. Na to je použitých 19 bitov, čo umožňuje, aby funkcie reprezentované týmito typmi diagramov mali maximálne 524287 premenných. `ref_count` vyjadruje aktuálny počet referencií. Na to je vyhradených 12 bitov. Posledný jeden bit slúži ako značka, ktorá sa používa predovšetkým pri garbage collectingu, re-hashovaní a v algoritmoch, ktoré prechádzajú grafom, napríklad pri počítaní koľko uzlov graf obsahuje. Viac bude spomenuté v sekcii 3.6.

Štruktúra zreťazených verzií (CBDD a CZDD) vyzerá takto:

```
struct chained_s {
    uint32_t lo, hi;
    uint32_t next;
    uint64_t start_idx : 20;
    uint64_t end_idx : 20;
    uint64_t ref_count : 20;
    uint64_t mark : 1;
    uint64_t used : 1;
    uint64_t reserved : 2;
} chained_t
```

Uzly CBDD a CZDD zaberajú v pamäti 24 Bytov. Efektívna veľkosť je síce len 20 Bytov, ale na 64 bitových architektúrach si to prekladač kvôli väčšej efektivite zaokrúhli na 24. To bolo potvrdené aj experimentom, kedy bola veľkosť štruktúry vynútená na 20 Bytov, čo viedlo k zhoršeniu celkového výkonu.

Položky `lo`, `hi`, `next` a `mark` sú rovnaké ako pri BDD alebo ZDD. Vieme, že tieto diagramy asociujú s každým uzlom dvojicu úrovní. Pri bežných hranách, kedy sa nepreskočil žiaden uzol sa tieto dve hodnoty rovnajú. V prípade, že došlo k redukcii, tieto hodnoty vyjadrujú začiatočnú (`start_idx`) a koncovú (`end_idx`) úroveň „don't care“ alebo „or“

reťazca záležiac na verzii diagramu. Veľkosť štruktúry umožňuje, aby pre tieto hodnoty bolo použitých až 20 bitov, čo umožňuje rozšíriť doménu reprezentovaných funkcií až na 1 048 575 premenných. 20 bitov je použitých takisto na počet referencií na daný uzol. Jedno bitová položka `used` vyjadruje či daná položka v tabuľke uzlov je zabratá alebo prázdna. Teda či sa jedná o reálny uzol alebo o „empty bucket“. Pri BDD a ZDD by jeden bit viedol k zníženiu rozsahu pre počet premenných alebo počet referencií o polovicu, preto je využitosť položky tabuľky uzlov testovaná ináč. Keďže BDD nemôžu obsahovať „don't care“ uzly, tak sa porovná `lo` hodnota voči `hi` hodnote. Ak sú rovnaké, miesto v tabuľke je voľné. Pri ZDD sa porovná `hi` s nulou. Ak je výsledok porovnania pravda, miesto v tabuľke je voľné, pretože ZDD nesmú obsahovať „high zero“ uzly, t.j. uzly, ktorých **high** potomok je booleovská konštanta 0. Posledné 2 bity označené ako `reserved` sú momentálne nevyužité.

Pri zvyšných dvoch typoch diagramov je situácia odlišná. Na to aby sme jednoznačne identifikovali TBDD je potrebná aj vstupná hrana do uzla obsahujúca značku (tag). Preto sú interne jednotlivé uzly TBDD identifikované 64 bitovým číslom (`uint64_t`). Spodných 32 bitov vyjadruje index do hashovacej tabuľky obsahujúcej uzly. Najvrchnejší 64. bit je nastavený na 0. Tým sa vie, že sa jedná o internú reprezentáciu uzla a nie o užívateľovu referenciu na uzol. Viac o týchto referenciách bude v sekcii 3.6. Zo zvyšných 31 bitov je 16 bitov použitých na značku hrany. Štruktúra TBDD uzla je nasledovná:

```
struct tbdd_s {
    uint32_t lo, hi;
    uint32_t next;
    uint16_t var_idx;
    uint16_t lo_tag;
    uint16_t hi_tag;
    uint16_t ref_count : 14;
    uint16_t mark : 1;
    uint16_t used : 1;
} tbdd_t
```

Položky `lo`, `hi`, `next`, `var_idx` a `mark` sú rovnaké ako pri BDD alebo ZDD s tým, že pre index premenných je použítých 16 bitov. To umožňuje reprezentovať funkcie v doméne $2^{16} - 1$ premenných. Najvyššia hodnota z tohoto rozsahu je 65535 a tá je použitá na špeciálnu značku $\perp$. Položky `lo_tag` a `hi_tag` určujú značky na hranách do **low** a **high** potomkov. Ako bolo spomenuté v sekcii 2.5 značka hrany môže byť buď index premennej alebo špeciálna značka $\perp$ vyjadrujúca „don't care“ redukciu do listového uzla. Z toho dôvodu musia mať tieto položky rovnakú veľkosť ako položka vyjadrujúca index premennej a takisto preto je 16 bitov zo 64 bitovej reprezentácie uzla použitých na značku vstupnej hrany. Pochopiteľne do jedného uzla môže viesť viacero vstupných hrán a práve preto nestačí na jednoznačnú identifikáciu TBDD len index do tabuľky uzlov. Napríklad **lo** potomka je takto možné z daného rodičovského uzla získať operáciou `(((uint64_t) tbdd.lo_tag « 32) | tbdd.lo)`, pričom `tbdd` je štruktúra pre rodičovský uzol. Jeden bit pre `used` je významovo rovnaký ako u CBDD alebo CZDD a to isté platí aj pre `ref_count`, na ktorých ostalo zvyšných 14 bitov, čo umožňuje 16 383 referencií. Celkovo zaberá TBDD uzol v pamäti 20 Bytov.

Čo sa týka ESRBDD, situácia je veľmi podobná ako u TBDD. Tam je pre jednoznačnú identifikáciu ESRBDD potrebné aj pravidlo vstupnej hrany. No na vyjadrenie 4 redukčných pravidiel stačia len 2 bity oproti 16 bitom potrebným na značku hrany v mojej implementácii TBDD. Avšak ubrať 2 bity z 32 bitového čísla by znamenalo znížiť maximálny počet uzlov

na štvrtinu, preto je pre internú reprezentáciu ESRBDD hrany s uzlom použitých 64 bitov. Spodných 32 bitov je použitých ako index do tabuľky uzlov. Najvyšší 64. bit takisto ako u TBDD odlišuje internú reprezentáciu uzla od užívateľom definovanej referencie na uzol. Zo zvyšných 31 bitov sú len 2 bity použité na označenie pravidla vstupnej hrany. Štruktúra uzla vyzerá nasledovne:

```
struct esrbdd_s {
    uint32_t lo, hi;
    uint32_t next;
    uint32_t var_idx : 15;
    uint32_t lo_edge_rule : 2;
    uint32_t hi_edge_rule : 2;
    uint32_t ref_count : 11;
    uint32_t mark : 1;
    uint32_t used : 1;
    uint64_t reduce_cache;
} esrbdd_t
```

Položky `lo`, `hi`, `next`, `var_idx`, `ref_count`, `used` a `mark` sú významovo rovnaké ako u TBDD len s trochu odlišnými bitovými šírkami. Pre počet referencií je použitých 11 bitov a pre počet premenných 15 bitov čo umožňuje funkcie s maximálne 32 767 premennými. Pri implementácii som sa snažil udržať veľkosť uzla čo najmenšiu, no v prípade potreby je jednoducho možné navýšiť veľkosť štruktúry o 4 Byty a tým dramaticky navýšiť maximálny počet premenných a referencií. `lo_edge_rule` a `hi_edge_rule` vyjadrujú redukčné pravidlá na hranách vedúcich do **low** a **high** potomkov, preto využívajú po 2 bity. Položka `reduce_cache` slúži ako cache pre výsledok unárnej operácie, ktorá redukuje ESRBDD uzol. Celkovo zaberá uzol ESRBDD v pamäti 24 Bytov.

Všetkých šesť verzií používa pre index do tabuľky uzlov 32 bitov. To umožňuje 2^{32} rôznych hodnôt. Pri prednastavenom maximálnom **load factore 0.68**, ktorý je v prípade potreby možné zmeniť, to znamená, že je možné mať naraz vytvorených maximálne 2 920 577 761 uzlov. Pre BDD a ZDD to znamená 43.5 GB, ktoré by boli použité len na tabuľku uzlov. Pre CBDD, CZDD a ESRBDD to je 65.27 GB a pre TBDD 54.4 GB pamäte použitej len na tabuľku uzlov.

Teoreticky by bolo možné použiť viac bitov na index do tabuľky uzlov, obzvlášť pri TBDD a ESRBDD, kde je zo 64 bitového čísla identifikujúceho uzol nevyužitých 15 bitov pre TBDD a 29 bitov pre ESRBDD. No to by znamenalo, že by bolo potrebné zväčšiť veľkosti pre položky `lo`, `hi` a `next`, čím by sa dosť zväčšila celková veľkosť štruktúr. To by v konečnom dôsledku viedlo k ešte menšiemu maximálnemu počtu uzlov pre stroje s pamäťou do 64 GB a takisto k zhoršeniu celkového výkonu pre použitú hashovaciu techniku, pretože do cacheliny by sa zmestil menší počet uzlov. Na druhej strane navyšovať len o pár bitov pri nezarovnanej veľkosti štruktúr už vôbec nemá zmysel, lebo to by viedlo k ešte väčšiemu zhoršeniu výkonu.

3.4 Užívateľove referencie na uzly diagramov

V predchádzajúcej sekcii sme spomínali, že interne sú jednotlivé uzly rozhodovacích diagramov reprezentované ako 32 bitové (pre BDD, ZDD, CBDD a CZDD) alebo 64 bitové (pre TBDD a ESRBDD) čísla bez znamienka. Z pohľadu užívateľa sú však všetky uzly re-

prezentované 64 bitovým číslom typu `uint64_t` a to platí pre všetky implementované typy diagramov. To je z nasledujúcich dôvodov.

Interná reprezentácia uzla udáva index do tabuľky uzlov, na ktorom sa daný uzol nachádza. Avšak v prípade re-hashovania tabuľky, ktoré sa spúšťa automaticky na základe aktuálneho faktoru naplnenia sa tieto indexy pre jednotlivé uzly zmenia. Preto je potrebné aby užívateľove referencie na uzol boli nezávislé na tabuľke uzlov. Zároveň som nechcel aby musel užívateľ stále na každý uzol, ktorý je výsledkom nejakej operácie pridávať referenciu. Niekedy je totiž potrebné nejaký diagram vytvoriť len ako medzi-výsledok, ktorý sa hneď použije ako operand v nasledujúcej operácii. V takomto prípade by bolo referencovanie takéhoto diagramu zbytočná operácia a navyše by potom bolo potrebné túto referenciu rušiť, aby daný diagram, ktorý už nie je potrebný nezaberal miesto. Preto funkcie implementujúce jednotlivé operácie s rozhodovacími diagramami vracajú internú reprezentáciu, teda index do hashovacej tabuľky a záleží na užívateľovi, či si na daný výsledný diagram pridá referenciu, čím bude zaručené, že pri automatickom uvoľňovaní pamäti o neho nepríde, alebo ho použije v nasledujúcej operácii. No v prípade, že neurobí ani jedno z toho, existuje riziko, že takýto diagram bude odstránený, nakoľko nasledujúca operácia vytvárajúca diagram môže spustiť automatické uvoľňovanie pamäti (viac bude naznačené v 3.6). Z toho vyplýva potreba, aby bolo možné ako parametre funkcií implementujúcich operácie s rozhodovacími diagramami používať aj užívateľove referencie, aj interné indexy. Z toho ďalej vyplýva, že tieto dva rôzne reprezentácie diagramu – internú a užívateľovu, treba nejakým spôsobom rozlíšiť.

Referencovanie sa deje nasledujúcim spôsobom. V knižnici je implementovaný kontajner, v ktorom sú uložené všetky užívateľom priamo referencované uzly diagramov. Tento kontajner je interne implementovaný ako pole 32 bitových čísel. V prípade pridania referencie na nejaký uzol, ktorý je reprezentovaný indexom do tabuľky uzlov – ide teda o internú reprezentáciu, sa v tomto kontajneri na najbližšiu voľnú pozíciu táto interná reprezentácia uzla uloží a vráti sa index, na ktorý bola uložená. K tomuto indexu sa potom v prípade BDD, ZDD, CBDD a CZDD pripočíta 2^{32} a pretože tieto typy diagramov sú interne 32 bitové čísla, takto vzniknutá referencia bude stále odlišná. V prípade TBDD a ESRBDD, čo sú aj interne 64 bitové čísla, no s najvyšším bitom nastaveným stále na 0, sa v indexe do poľa referencovaných uzlov najvyšší bit nastaví na 1.

Tým je vyriešený problém s re-hashovaním, pretože ak k nemu dôjde nové hashe sa uložia v tomto poli referencovaných uzlov presne na tie isté miesta, kde boli staré hashe. Zároveň funkcie implementujúce operácie s diagramami vedia či sa jedná rovno o index do tabuľky uzlov alebo o referenciu, ktorú treba najprv dereferencovať. To sa deje opačným spôsobom. V prípade napríklad BDD sa vezme referencia, odpočíta sa od nej 2^{32}, čím získame index do poľa referencií, na ktorom je stále aktuálny index do tabuľky uzlov.

3.5 Tabuľka výsledkov – Computed cache

Computed cache funguje ako softwarová cache pre výsledky rekurzívnych operácií rozhodovacích diagramov a z hľadiska výkonu je najdôležitejšou súčasťou implementácie. Stále na začiatku každej netriviálnej operácie na diagramoch sa najprv skontroluje či daná operácia s danými operandmi už nebola niekedy počítaná a nie je uložená v tejto cache. Ak áno hneď sa vráti jej výsledok, čím môže odpadnúť veľké množstvo rekurzívnych operácií. Ak tam ten výsledok nie je, operácia normálne pokračuje pokým sa nenarazí na medzi-výsledok, ktorý v cache je alebo dôjde na problém, ktorý už je triviálny, to znamená taký, ktorý ukončuje rekurziu. Obvykle, ale nie výlučne to znamená, že oba operandy sú rovnaké alebo

aspoň jeden z nich je konštanta. Na konci každej netriviálnej operácie sa potom operandy, operátor a výsledok uložia do tejto cache s nádejou, že budú využité neskôr.

Táto tabuľka je implementovaná na štýl jedno-cestnej (priamo mapovanej) cache. V podstate ide o hashovaciu tabuľku, kde sa ignorujú kolízie a nový prvok jednoducho prepíše pôvodný. Niektoré implementácie používajú oddelené tabuľky pre jednotlivé typy operácií (myslí sa napr. jedna tabuľka pre operácie AND, jedna pre operácie OR a podobne), iné používajú jednu spoločnú tabuľku pre všetky operácie. Vyskytujú sa aj hybridné riešenia. Takéto riešenie som použil aj ja – jedna tabuľka výsledkov je pre operácie s dvoma operandmi ako napríklad `APPLY` a jedna tabuľka pre operácie s troma operandmi ako napríklad `ITE`. V prípade používania operácií len s dvoma operandmi je možné knižnicu preložiť s makrom `ALGEBRAIC_ONLY`[3], kedy sa alokuje len jedna tabuľka, čím je možné ušetriť pamäť.

Položka tejto tabuľky má pre BDD, ZDD, CBDD a CZDD nasledujúcu štruktúru:

```
struct cache_entry {
    uint32_t u;
    uint32_t v;
    uint32_t res;
    int operation;
}
```

`u` a `v` sú operandy a `res` je výsledok. Všetky tri sú 32 bitové čísla a udávajú index do tabuľky uzlov („unique table“), na ktorom sa nachádza koreň grafu reprezentujúci funkciu, ktorá je operandom alebo výsledkom operácie. `operation` udáva interný operačný kód pre daný typ operácie. Pre TBDD a ESRBDD vyzerá štruktúra rovnako s jediným rozdielom, že operandy a výsledok sú 64 bitové čísla. Analogicky vyzerá štruktúra pre operácie s troma operandmi.

Pri ESRBDD sú implementované len operácie s dvoma operandmi, preto tam cache pre operácie s troma operandmi vôbec nie je. Avšak ESRBDD v algoritme pre operáciu `APPLY` vykonáva rekurzívnu redukciu výsledného uzla. Z toho dôvodu som pridal istú formu cache aj pre túto redukciu uzlov. Vzhľadom na to, že to je unárna operácia a kvôli lepšej lokalite odkazov som túto cache umiestnil priamo do uzla ESRBDD (viď sekciu 3.3).

Ako už bolo povedané, computed cache je z hľadiska výkonu rozhodujúcim prvkom rozhodovacích diagramov, a to aj napriek tomu, že funkcie, ktoré s ňou pracujú, na nej strávia viac ako polovicu celkového času. Práca s computed cache zaberá BDD, ZDD, CBDD, CZDD a TBDD okolo 65% času z celkového času funkcií implementujúcich operácie s rozhodovacími diagramami, ako je napríklad `APPLY`. Pre ESRBDD je to kvôli komplexnejším algoritmom približne 50%. Príklad dopadu computed cache na výkon diagramov ukazujú nasledujúce dve tabuľky, ktoré udávajú celkový čas potrebný na vytvorenie diagramu pre obvod ISCAS85 c3540 a diagramu pre všetky riešenia problému 8 kráľovien s využitím computed cache a bez nej pre rôzne typy rozhodovacích diagramov. ZDD verzia nemá bez cache hodnotu, pretože sa nezmestila pod vrchný limit 7 minút, pričom s cache to bolo do pár sekúnd. To je z toho dôvodu, že pre ZDD pri manipulácii s booleovskými funkciami má reprezentácia jednej premennej toľko uzlov, aká je veľkosť domény plus dva listové uzly navyše. Tým sa oproti ostatným typom diagramov značne navyšuje počet rekurzívnych operácií. Je to podobný prípad ako s reprezentáciou konštantnej hodnoty **1**, ktorý je na Obrázku 3.3. Počet týchto rekurzívnych operácií pre dva vyššie spomínané prípady je v tabuľkách a grafoch prílohy C.

[3]Rozdelenie oprácií na algebraické a nealgebraické je podľa [7].

c3540		
typ	s cache	bez cache
BDD	2.003 s	17.202 s
ZDD	3.659 s	N/A
CBDD	2.188 s	21.031 s
CZDD	2.766 s	57.925 s
TBDD	2.247 s	20.404 s
ESRBDD	3.750 s	67.547 s

8 kráľovien		
typ	s cache	bez cache
BDD	3.294 s	30.361 s
ZDD	1.917 s	N/A
CBDD	3.087 s	32.446 s
CZDD	3.062 s	213.935 s
TBDD	3.036 s	13.677 s
ESRBDD	3.670 s	335.611 s

3.6 Garbage collector a referencovanie uzlov

Pri operáciách s rozhodovacími diagramami často vzniká veľa prechodných diagramov. Z toho dôvodu je potrebný nejaký mechanizmus, ktorý by uvoľňoval pamäť zabranú diagramami, ktoré už nie sú potrebné.

Na tento účel som v knižnici implementoval jednoduchý „garbage collector“ založený na princípe „mark and sweep“ algoritmu. Z toho dôvodu sa takisto v štruktúrach uzlov nachádza značka – `mark` a položka na počítanie referencií – `ref_count`. Garbage collector má takisto vlastný zásobník (stack), na ktorý sa ukladajú interné referencie na uzly, ktoré figurujú ako operandy alebo medzi-výsledky rozpracovanej operácie. Operácie nad rozhodovacími diagramami, ktorých výsledkom je rozhodovací diagram ako napríklad `APPLY` totiž volajú funkciu, ktorá vytvára uzly a táto funkcia môže spustiť garbage collecting. Z toho dôvodu je treba zaistiť aby tieto operandy a takisto stále potrebné medzi-výsledky neboli pri prípadnom garbage collectingu odstránené a na to slúži spomínaný zásobník.

Implementovaný algoritmus Garbage collectingu vyzerá nasledovne. Najprv sa postupne pre všetky uzly v zásobníku interných referencií a takisto rekurzívne pre ich potomkov nastaví značka (`mark`), znamenajúca, že uzol je z niekade dosiahnuteľný. Potom sa lineárne prechádzajú všetky uzly v tabuľke uzlov. Ak daný uzol má nastavenú značku alebo ak má nejaký počet referencií v položke `ref_count`, nechá sa tak, ak uzol nemal ani značku ani externú referenciu zmaže sa. Následne sa zmažú všetky záznamy v computed cache, ktoré mali ako operand alebo výsledok nejaký mŕtvy uzol. Nakoniec sa odstránia značky (`mark`) zo všetkých uzlov, aby mohli byť použité v ďalšom behu alebo v inej operácii (napríklad počítanie uzlov).

V počítadle referencií (`ref_count`) sú započítané jednak priame referencie na daný uzol vytvorené užívateľom zavolaním príslušnej funkcie a jednak referencie rodičovských uzlov, ktorý majú daný uzol ako **low** alebo **high** potomka. No toto počítanie sa deje len ak je iniciované prvým spôsobom (referencovanie užívateľom zavolaním príslušnej funkcie) ma koreni nejakého grafu. V takom prípade sa rekurzívne zvýši počet referencií na jeho potomkoch a ďalej ich potomkoch až po listové uzly. Analogicky to platí pri odstránení referencie. Počet referencií sa neudržuje stále aktuálny. Teraz máme na mysli to, že pre nejaký uzol sa nezvyšuje/neznižuje zakaždým ak ten uzol je pridaný/odobraný ako potomok iného uzla z toho dôvodu, že veľa grafov vzniká len ako medzi-výsledky, ktoré sa ďalej nepoužijú. Takto by musel garbage collector všetkým týmto uzlom znižovať rekurzívne

referencie, čo nemusí, ale môže zabrať značnú časť CPU času. [15] Zároveň takto pre počet referencií stačia položky s menšími bitovými šírkami. No aj napriek tomu sa z dôvodu obmedzeného rozsahu týchto počítadiel používajú saturovaný inkrement a dekrement. To znamená, že ak počet referencií dosiahne maximálnu hodnotu danú rozsahom počítadla, ďalej sa už nezvyšuje, ale ani neznižuje, lebo už nie je jasné koľkokrát presiahol maximálnu hodnotu.

Garbage collector by mal byť spustený vtedy, ak to oprávňuje dostatočný počet mŕtvych uzlov. Z pohľadu využitia pamäte, by tento počet mal byť malý, no z pohľadu výkonu je lepšie počkať kým sa naakumuluje väčší počet mŕtvych uzlov. Tým sa jednak amortizujú režijne náklady garbage collectingu, no hlavne sa tým zvýši výkonnosť computed cache a to niekedy dramaticky. Viac záznamov znamená väčšiu pravdepodobnosť, že sa v nej bude nachádzať požadovaný výsledok. Vzhľadom na to, že na základe spôsobu referecovania uzlov, ktorý používa aj táto knižnica nie je presne známy počet mŕtvych uzlov, garbage collector sa spúšťa ak pomer počtu vytvorených uzlov od posledného behu k celkovej kapacite tabuľky uzlov presiahne stanovený limit. Musí teda platiť vzťah:

$$\frac{nodes\ created\ after\ last\ GBC}{unique\ table\ size} > GBC^{4}\ trigger\ threshold$$

Tento limit sa dá nastavovať pri preklade definovaním makra `-D GBC_TRIGGER=X`, kde `X` je číslo vyjadrujúce limit.

Garbage collector má v súvislosti so zatvoreným hashovaním ešte jeden nežiadúci efekt. On totiž v podstate vymazuje položky z hashovacej tabuľky. No pri zatvorenom hashovaní vyhľadávacie algoritmy končia ak sa v tabuľke narazí na voľnú pozíciu. Vymazanie položky takto môže vytvoriť dieru vo vyhľadávacej sekvencii, čo ďalej môže zapríčiniť priskoré ukončenie vyhľadávacieho algoritmu. Z toho dôvodu sa typicky v technikách zatvoreného hashovania položky nemažú, ale len označujú ako „vymazané“. Takéto položky sa nazývajú „náhrobky“ („tombstones“). V implementácii knižnice sú tieto položky identifikované rôzne v závislosti na použitej hashovacej technike a type rozhodovacieho diagramu. Pri vkladaní sa na ich miesto môže vložiť vkladaný prvok, no najprv sa musí prezrieť celý zvyšok „kolízneho reťazca“ až pokým sa nenarazí na prvú normálnu voľnú pozíciu alebo hľadaný prvok, pretože daný prvok by sa mohol nachádzať v tomto „reťazci“. Takto tieto náhrobky neblokujú miesto a nekontaminujú hashovaciu tabuľku, no zbytočne predlžujú vyhľadávacie sekvencie. Z toho dôvodu sa po určitom počte garbage collectingov celá tabuľka re-hashuje do rovnako veľkej tabuľky, aby sa tým skrátili vyhľadávacie sekvencie. Počet po koľkých behoch garbage collectora sa to stane sa dá nastaviť pri preklade definovaním makra `-D REORGANIZE=X`, kde `X` je celé kladné číslo. Ak je `X` rovné 0, táto funkcia je vypnutá. V súvislosti s touto reorganizáciou a vyššie spomínaným limitom pre spúšťanie garbage collectora boli vykonané experimenty, ktorých výsledky sú zhrnuté v prílohe B.

[4] skratka pre Garbage collection

Kapitola 4

Experimenty

V tejto kapitole sú popísané benchmarky, ktoré poukazujú na niektoré špecifiká jednotlivých typov rozhodovacích diagramov a ich implementácie. Bechmarky sú zamerané na riešenie kombinatorických problémov, reprezentovanie informácií v kompaktnej forme a reprezentovanie typických digitálnych obvodov.

Pre každý z experimentov boli všetky testy pre každý diagram vykonané štyrikrát: dva s hashovacou technikou linear probing a dva s technikou quadratic probing s rôznymi počiatočnými veľkosťami tabuliek uzlov a tabuliek výsledkov, aby sa aspoň čiastočne znížila pravdepodobnosť, že nejaký z diagramov mal šťastný zásah do tabuľky výsledkov. Z týchto štyroch testov bol pre každý diagram vybraný najlepší výsledok.

Všetky experimenty boli vykonané na procesore 2.6 GHz Intel Xeon s 32 GB pamäte a operačným systémom Debian 10.

4.1 Problém N kráľovien

Tento experiment sa týka reprezentovania všetkých možných riešení problému N kráľovien vo forme booleovskej funkcie. Problém N kráľovien spočíva v tom, ako rozložiť N kráľovien na šachovnicu o veľkosti $N \times N$, tak aby žiadna kráľovná nemohla podľa pravidiel šachu ohroziť inú kráľovnú.

Vstup experimentu bol vo formáte *DIMACS CNF*, to znamená že problém bol vyjadrený ako súbor obmedzujúcich podmienok v konjuktívnej normálnej forme. Jedno políčko šachovnice znamenalo jednu premennú. Teda napríklad pre problém 10 kráľovien bolo potrebných 100 premenných. Experiment bol vykonaný pre $N = 7, \ldots, 14$.

Tabuľka 4.1 zobrazuje výsledky experimentu pre $N = 14$ v kategóriach CPU čas v sekundách, počet celkovo vytvorených uzlov za beh programu, počet uzlov výsledného grafu, počet netriviálnych rekurzívnych operácií a počet CPU inštrukcií.

Keď sa na výsledky pozrieme z pohľadu veľkosti výsledného grafu, už z povahy problému N kráľovien a vstupného formátu — *DIMACS CNF* môžeme usudzovať, že sa tam bude vyskytovať veľa negovaných premenných, čo je výhoda pre všetky typy zúčastnených diagramov, okrem BDD, pretože BDD neredukujú „high zero“ uzly. Takisto vidíme, že veľkosti výsledných grafov sú pre ZDD, CZDD a TBDD identické, z čoho môžeme usudzovať, že výsledné grafy neobsahujú žiadne „don't care“ uzly. Z toho dôvodu BDD nemajú čo redukovať a veľkosť výsledného grafu je pre ne najväčšia. Z toho istého dôvodu CZDD a TBDD neušetria oproti ZDD ani jeden uzol, no na druhej strane to znamená výhodu pre CBDD, ktoré sú kvôli redukcii „OR-chains“ až 4.8 krát kompaktnejšie ako obyčajné BDD. Najlepšie

Tabuľka 4.1: Výsledky experimentu na probléme 14 kráľovien v kategóriach CPU čas v sekundách, počet celkovo vytvorených uzlov za beh programu, počet uzlov výsledného grafu, počet netriviálnych rekurzívnych operácií a počet CPU inštrukcií.

typ	CPU čas	vyt. uzl. celk.	poč. uzlov	počet rek. op.	počet inštr.
BDD	4047.01	2,491,955,908	9,572,420	21,934,758,195	4,919,986,455,492
ZDD	1536.25	515,418,070	911,422	24,022,730,093	2,094,022,543,353
CBDD	1517.02	1,133,503,766	1,992,396	8,744,432,499	2,452,369,193,557
CZDD	3151.93	515,419,259	911,422	24,068,428,916	7,288,773,384,468
TBDD	904.92	517,096,313	911,422	8,318,248,736	1,707,223,665,025
ESRBDD	12141.44	6,082,865,652	909,616	43,894,543,252	14,327,749,157,480

sú na tom ESRBDD, ktoré dokážu navyše redukovať aj „low zero“ uzly, teda uzly, ktoré majú **0** ako **low** potomka.

Ak sa na výsledky pozrieme z pohľadu CPU času, jednoznačným víťazom sú TBDD. Ak ich porovnáme so ZDD, tak majú rovnako veľký výsledný graf a približne rovnaký počet celkovo vytvorených uzlov v priebehu programu. No ZDD sú pri manipulácii s booleovskými funkciami v značnej nevýhode kvôli tomu ako vyzerajú funkcie reprezentujúce jednotlivé premenné. Vyjadrenie premennej v TBDD je nezávislé na veľkosti domény problému. Stále tam pre akúkoľvek premennú stačí jeden neterminálny uzol a dva listové uzly, no pre ZDD to neplatí. Tie musia explicitne vyjadriť každý „don't care“ uzol. Pri tomto konkrétnom prípade, kde je 196 premenných to znamená, že každá jednotlivá premenná musí mať 196 interných uzlov a dva listové. To je našťastie zmiernené tým, že ZDD majú zo všetkých šiestich typov diagramov najjednoduchšie algoritmy implementujúce binárne operácie. Jadrom týchto algoritmov je v podstate jedno `if - else if - else` vetvenie, kde v dvoch vetvách sa volá funkcia rekurzívne raz a v tretej vetve dvakrát.

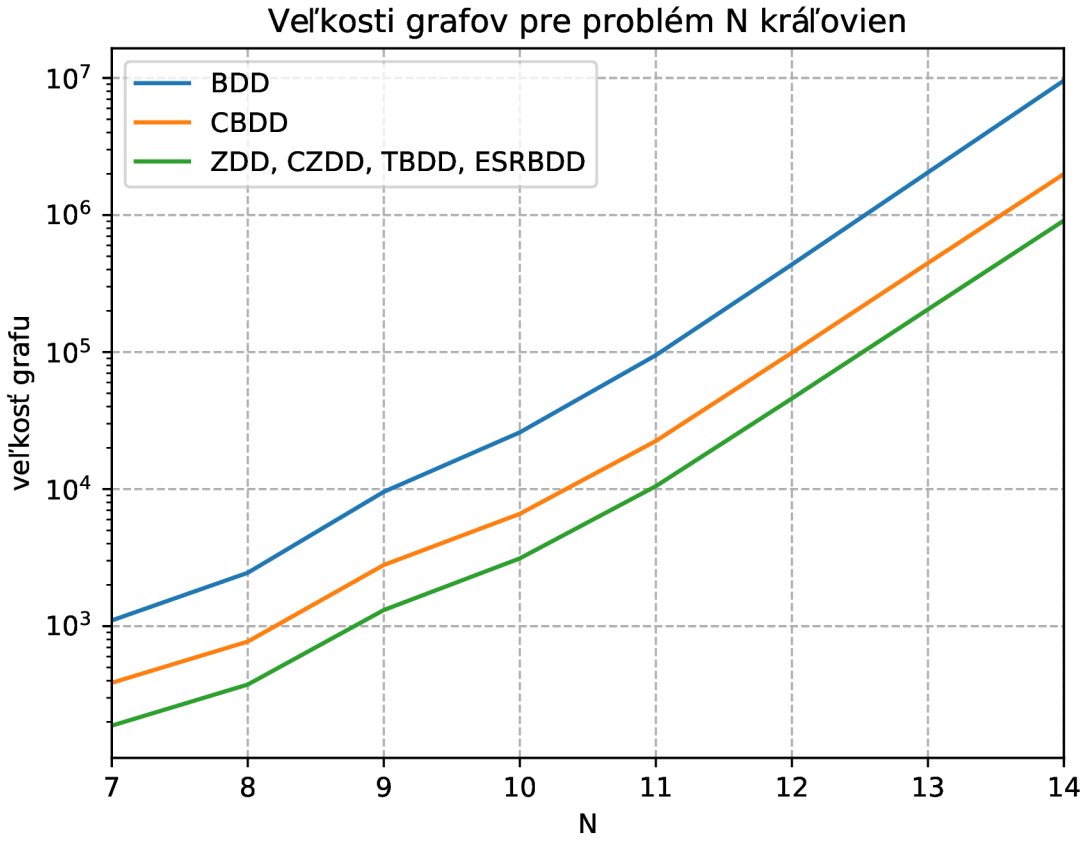
Rovnako veľký výsledný graf a približne rovnaký počet celkovo vytvorených uzlov majú aj CZDD. Tie tým že redukujú „don't care chains“ majú oproti ZDD výhodu pri reprezentácii jednotlivých premenných. Na vyjadrenie premennej stačia dva neterminálne uzly, v prípade poslednej premennej jeden, a dva listové uzly. Avšak vzhľadom na to, že vo výslednom grafe v tomto konkrétnom probléme nie sú žiadne „don't care“ uzly, neposkytujú tu oproti klasickým ZDD žiadnu výhodu. Takisto z trojice ZDD, CZDD a TBDD majú najzložitejšie algoritmy a preto boli pomalšie ako ZDD.

Prekvapivý výsledok bol v súvislosti s CBDD. Tie napriek tomu, že mali dvakrát väčší výsledný graf ako i počet celkovo vytvorených uzlov, boli zároveň dvakrát rýchlejšie ako CZDD a trochu rýchlejšie ako ZDD. CBDD stačí na vyjadrenie premennej 1 neterminálny uzol a dva listové, čo ich stavia do výhody oproti ZDD. Čo sa týka zložitosti algoritmov, sú na tom veľmi podobne ako CZDD. Hlavnou príčinou rozdielu medzi CBDD a CZDD vo výkone bol počet rekurzívnych operácií. Z nejakého dôvodu mali CZDD v mojej implementácii zhruba dvakrát viac kolízií v tabuľke uzlov i tabuľke výsledkov. V súvislosti s tým som skúšal viacero hashovacích funkcií ako i rôzne usporiadania položiek zloženého kľúča, aby som znížil počet kolízií v CZDD, no nedošiel som k výraznejšiemu pokroku. Na druhej strane očakávaný bol rozdiel v porovnaní s klasickými BDD. CBDD tu mohli naplno využiť redukčné pravidlá ZDD, a preto boli 2.67 krát rýchlejšie ako klasické BDD.

Paradoxne najpomalšie boli v mojej implementácii ESRBDD, ktoré majú najmenší výsledný graf. Tie však majú zo všetkých šiestich typov diagramov najkomplexnejšie algoritmy. Takisto vytvárajú v priebehu konštrukcie grafu najväčší počet uzlov, o čo sa pričiňuje strie-

danie redukčných pravidiel a ich spájanie. Ak ich porovnáme s TBDD, ktoré na vyjadrenie redukcií používajú ako značky indexy premenných, tak redukčné pravidlá ESRBDD možno sú konceptuálne jednoduchšie, avšak značka hrany v podobe indexu umožňuje jednoduchšie algoritmy a navyše TBDD na prechod z „don't care“ uzla na „high zero“ uzol nepotrebujú prechodný uzol navyše.

Výsledky experimentu pre $N = 7, \ldots, 14$ demonštruje graf 4.1 a tabuľky D.1, D.2 a D.3 v prílohe D.



Obr. 4.1: Veľkosti výsledných grafov problému N kráľovien pre $N = 7, .., 14$.

4.2 Slovník

Ďalší experiment je zameraný na schopnosť kompaktne reprezentovať informácie. Vstupom pre tento experiment bol slovník, ktorý sa nachádza na Macintosh systémoch v súbore `/usr/share/dict/words`. Súbor slov môže byť zakódovaný ako funkcia mapujúca slová na 1, ak slovo v danom zozname je alebo 0 ak tam nie je. Slová môžu byť ďalej zakódované ako sekvencia bitov. To umožňuje slová reprezentovať ako booleovskú funkciu, kde jednotlivé premenné tejto funkcie sú bity, ktorými sú zakódované slová.

Slovník bol zakódovaný binárnym kódom, ktorý vyžaduje $\log_2 r$ bitov na jeden znak, kde r je počet všetkých možných znakov. Každý znak je takto reprezentovaný unikátnou sekvenciou bitov, ktoré sú pospájané do slov. Spomínaný vstupný slovník obsahuje 235886 slov s maximálnou dĺžkou 24 znakov. Možné znaky sú malé a veľké písmená anglickej abecedy, pomlčka a `NULL` symbol, ktorý znamená koniec slova. Teda 54 rôznych znakov. Z toho

vyplýva, že funkcia, ktorá reprezentuje tento slovník má 144 premenných ($\lceil log_2\ 54 \rceil * 24$). Slová ktoré sú kratšie ako 24 znakov boli doplnené potrebným počtom `NULL` symbolov. Slovo je tvorené konjunkciou znakov – premenných a slovník disjunkciou slov, čím vlastne dostávame funkciu v disjunktívnej normálnej forme, ktorá je základom pre vytvorenie grafov.

Tabuľka 4.2: Výsledky experimentu so slovníkom na Macintosh systémoch v kategóriach CPU čas v sekundách, počet celkovo vytvorených uzlov za beh programu, počet uzlov výsledného grafu, počet netriviálnych rekurzívnych operácií a počet CPU inštrukcií.

typ	CPU čas	vyt. uzl. celk.	poč. uzlov	poč. rek. op	poč. inštr.
BDD	150.504	778,993,968	1,104,755	817,182,079	270,317,569,649
ZDD	91.241	369,049,997	651,995	745,942,751	137,773,270,145
CBDD	152.828	646,112,334	949,177	684,172,993	271,707,730,148
CZDD	158.825	390,609,486	651,994	791,245,307	306,257,856,018
TBDD	91.124	390,470,215	651,906	465,358,011	150,142,678,623
ESRBDD	258.304	822,057,267	461,156	1,711,377,133	665,413,448,556

Výsledky sú zhrnuté v Tabuľke 4.2. Ak sa pozrieme na veľkosti výsledných grafov, tak najviac uzlov má BDD, pretože vo výsledku sa nachádza len minimálne množstvo uzlov, na ktorých je funkcia nezávislá. To je zrejmé z minimálneho rozdielu počtu uzlov ZDD, CZDD a TBDD. O niečo lepšie je na tom CBDD vďaka redukcii „OR-chains“. No narozdiel od predošlého experimentu v sekcii 4.1, kde bol pomer medzi veľkosťami BDD a CBDD skoro päťnásobný v prospech CBDD, tu je tento pomer len 1.164. CBDD totiž, aby mohli redukovať tieto „OR-chains“ potrebujú z nich ponechať jeden uzol. To znamená, čím sú tieto reťazce dlhšie, tým viac sú CBDD výhodné. Naopak ak je týchto reťazcov veľa a sú krátke, pomer medzi CBDD a BDD bude malý ako je to v tomto prípade.

Ako bolo spomínané vyššie, najdlhšie slová v slovníku mali 24 znakov a tie kratšie boli doplnené do konca symbolom `NULL`, ktorý je zakódovaný ako `000000`, čo znamená šesť za sebou idúcich negovaných premenných. To spolu s priemernou dĺžkou slova, ktorá je v tomto slovníku 10.57 znamená veľké množstvo „high zero“ uzlov. Vďaka tomu majú v tomto experimente ZDD skoro polovične veľký graf oproti BDD. O jeden uzol menší graf ako ZDD majú CZDD. Tie analogicky k CBDD, ak chcú redukovať „don’t care“ reťazce, potrebujú z nich jeden uzol ponechať. Z toho dôvodu v tomto prípade neprinášajú skoro žiadnu úsporu. O niečo lepšie sú na tom TBDD, keďže tie na vyjadrenie „don’t care“ uzlov nepotrebujú uzol navyše. Jedinou výnimkou je prípad keď „don’t care“ uzol nasleduje hneď za „high zero“ uzlom. Najmenší výsledný graf vďaka redukčnému pravidlu navyše majú ESRBDD. Tie sú v tomto prípade až 1.41 krát menšie ako TBDD a 2.4 krát menšie ako BDD.

Čo sa týka CPU času potrebného na vytvorenie diagramov, tak ten nezávisí na veľkosti výsledku, ale na počte inštrukcií, ktorý sa odvíja hlavne od počtu rekurzívnych operácií. Tie sú zasa do veľkej miery závislé na stratégii použitej pri computed cache a garbage collectore. Najlepší čas mali TBDD, pretože mali najmenší počet rekurzívnych operácií. Za to môžu ich redukčné schopnosti, ale aj dobré zásahy do computed cache. Tesne za nimi skončili ZDD, ktoré síce mali 1.6 krát viac operácií, ale vynahradili to jednoduchými algoritmami, čo môžeme vidieť na celkovom počte CPU inštrukcií, ktorý majú najmenší. Na treťom mieste skončili klasické BDD, ktoré v tomto prípade mali najmenšie možnosti redukovať. Hneď za nimi boli CBDD, u ktorých príležitosti redukovať „OR-chains“ neboli dostatočné na to, aby vykompenzovali väčšiu zložitosť algoritmov. Napriek tomu, že mali menej rekurzívnych

APPLy operácií ako i celkový počet vytvorených uzlov, ich celkový počet CPU inštrukcií bol o 0.5% vyšší ako u klasických BDD. CZDD kvôli malému počtu „don't care“ uzlov, nemali takisto veľkú príležitosť na zlepšenie oproti ZDD. Zároveň ako bolo spomenuté v sekcii 4.1 sa na nich prejavil nedostatok spojený s tým, že v mojej implementácii pri nich vzniká podstatne väčšie množstvo kolízií ako u ostatných typoch diagramov. Napriek tomu, že algoritmy sú podobné ako u CBDD, v tomto probléme je množstvo „high zero“ uzlov a dokonca celkový počet vytvorených uzlov bol podstatne menší ako u CBDD, majú väčší počet rekurzívnych operácií ako CBDD a horší čas. Problém s kolíziami sa v ešte väčšej miere prejavil aj na ESRBDD. To spolu so zložitosťou ich algoritmov spôsobilo, že napriek tomu, že majú naviac možností redukovať, vytvorili počas chodu programu najviac uzlov, no predovšetkým mali najväčší počet rekurzívnych APPLY operácií.

Tabuľka 4.3: Postupný nárast výsledného grafu pri budovaní grafovej reprezentácie slovníka.

časť slov.	BDD	ZDD	CBDD	CZDD	TBDD	ESRBDD
1	1104755	651995	949177	651994	651906	461156
9/10	1049802	617792	900857	617792	617706	435841
8/10	988218	579470	846700	579470	579391	407844
7/10	915909	535292	783640	535292	535214	375783
6/10	837072	487489	715258	487489	487419	341280
5/10	751003	435436	640594	435436	435370	303969
4/10	654791	377820	557498	377820	377761	262692
3/10	544312	311949	462110	311949	311908	216267
2/10	417374	237214	353207	237214	237189	163609
1/10	264226	147934	222209	147934	147924	101398

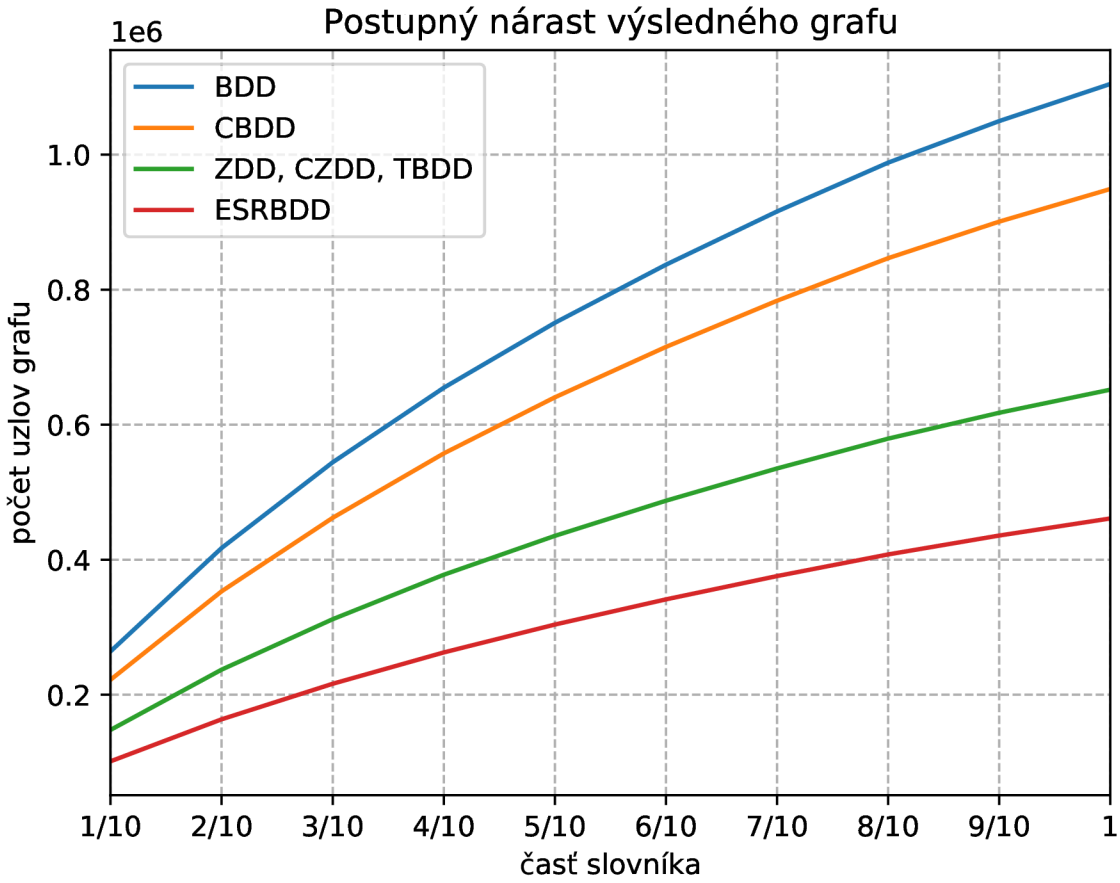
Tabuľka 4.3 a graf na Obrázku 4.2 ukazujú ako sa postupne vyvíjala veľkosť diagramov pri budovaní výslednej grafovej reprezentácie slovníka.

4.3 Digitálne obvody

BDD sa zvyknú využívať pri syntéze a verifikácii digitálnych obvodov[2], a preto ďalším benchmarkom je skupina vybraných digitálnych obvodov z benchmarkovej sady *ISCAS '85*. Táto sada bola pôvodne vyvinutá ako benchmark na generovanie testov, ale uchytila sa aj pri testovaní rozhodovacích diagramov. Oproti predošlým experimentom, tento je zameraný na schopnosť redukovať redundantné uzly. Tabuľka 4.4 prezentuje veľkosti rozhodovacích diagramov reprezentujúcich všetky výstupy jednotlivých obvodov. Všetky obvody mali viacero výstupov, preto číslo v tabuľke je súčtom všetkých výstupov daného obvodu.

Vzhľadom na to, že test je zameraný na redundantné uzly, vo všeobecnosti najlepšie dopadli ESRBDD, TBDD, CBDD a BDD s malými rozdielmi v počte uzlov. CZDD boli vďaka reťazeniu „don't care“ uzlov v najlepšom prípade 1.5 krát lepšie ako ZDD, ktoré mali s výnimkou obvodov c499 a c1908 vždy najväčšie grafy. V spomínaných dvoch obvodoch skončili na treťom mieste spolu s CZDD.

Čo sa týka časov, všetky obvody okrem c3540 a c6288 prebehli do jednej sekundy. Vybudovanie diagramu pre c3540, čo je 8 bitová ALU, trvalo od 2.6 do 5.75 sekúnd s najlepším časom pre BDD a najhorším pre ZDD, nakoľko malý počet „high zero“ a „low zero“ uzlov nestačil na vykompenzovanie zložitejších algoritmov TBDD, CBDD, ESRBDD a CZDD.



Obr. 4.2: Postupný nárast výsledného grafu pri budovaní grafovej reprezentácie slovníka.

Jednoduchosť algoritmov a s tým spojený najmenší počet CPU inštrukcií bol rozhodujúci aj pri obvode c6288. c6288 je 32 bitová násobička, pre ktorú bolo dokázané, že rozhodovacie diagramy majú pri nej stále exponenciálnu zložitosť a to aj pri ideálnom usporiadaní [3]. Konkrétne výsledky pre tento obvod sú v Tabuľke 4.5.

Tabuľka 4.4: Veľkosti diagramov pre jednotlivé obvody.

obvod	BDD	ZDD	CBDD	CZDD	TBDD	ESRBDD
c432	1,850	2,943	1,850	2,616	1,827	1,789
c499	50,684	50,451	50,672	50,451	50,429	50,345
c880	346,690	516,741	345,635	380,143	346,689	346,216
c1355	50,684	50,451	50,672	50,451	50,429	50,345
c1908	49,325	49,651	49,198	49,421	48,196	48,179
c3540	672,437	1,088,275	670,107	725,509	659,318	653,926
c6288	48,181,908	48,331,495	48,177,349	48,329,117	48,181,886	48,141,015

Tabuľka 4.5: Výsledky benchmarku pre obvod c6288 – 32 bitová násobička.

typ	CPU čas	vyt. uzl. celk.	poč. uzlov	poč. rek. op	poč. inštr.
BDD	261.412	543,992,753	48,181,908	1,660,276,435	249,057,571,370
ZDD	263.913	545,755,885	48,331,495	1,672,090,135	257,190,432,781
CBDD	352.119	543,986,618	48,177,349	1,660,211,012	333,936,083,754
CZDD	361.467	545,716,661	48,329,117	1,671,936,616	417,175,679,967
TBDD	409.072	543,911,169	48,181,886	2,072,809,750	347,606,942,094
ESRBDD	544.662	583,773,953	48,141,015	2,012,085,302	742,026,801,271

Kapitola 5

Záver

Výsledkom tejto práce je knižnica, ktorá podporuje šesť variánt binárnych rozhodovacích diagramov — BDD, ZDD, CBDD, CZDD, TBDD a ESRBDD. Táto knižnica je implementovaná v jazyku `ISO C`, konštruovanie diagramov je založené na „depth-first“ prechode, používa garbage collector na princípe „mark and sweep“ algoritmu, uzly diagramov sú integrované do tabuľky uzlov, ktorá používa „otvorené adresovanie“ a referencovanie uzlov diagramov je založené na indexoch a nie ukazateľoch. Tieto techniky umožnili znížiť pamäťové nároky na jeden uzol v niektorých prípadoch až na 12 bytov. Hashovacie kolízie v tabuľke uzlov sú vyriešené implementáciou troch základných techník zatvoreného hashovania. Konkrétnu implementáciu je potom možné vybrať podmieneným prekladom.

Experimenty s knižnicou potvrdili, že výber optimálnej varianty rozhodovacieho diagramu závisí na riešenom probléme. V prípadoch, kedy je možné využiť len jedno z redukčných pravidiel je výhodnejšie použiť BDD alebo ZDD záležiac, o aké pravidlo sa jedná. Podobne je to v prípade, keď je známe, že problém bude mať stále exponenciálnu zložitosť ako je to napríklad v prípade násobičiek. V takých prípadoch sa na čase potrebnom na zostrojenie výsledného diagramu prejavia vyššie režijne náklady ostatných štyroch variánt. No vo všeobecnosti sa použitie sofistikovanejších variánt oplatí. Ako najlepšia voľba sa v takom prípade ukázali TBDD. Tie mali stále menej uzlov ako CBDD a CZDD a vo väčšine prípadov aj lepší čas. V porovnaní s ESRBDD mali síce viac uzlov, no vždy mali podstatne lepší čas.

Existuje viacero možností ako v budúcnosti knižnicu vylepšiť a rozšíriť. Patrí medzi ne napríklad implementovanie dynamického preusporiadania premenných, podpora hrán s atribútmi alebo implementovanie samostatných špecializovaných funkcií pre komplexnejšie operácie. Možné je takisto implementovať iné stratégie garbage collectora a počítania referencií alebo experimentovať s ďalšími hashovacími funkciami a technikami riešenia kolízií. To môže byť obzvlášť prospešné pre CZDD a ESRBDD, pretože pri nich v súčasnej implementácii vzniká viac hashovacích kolízií ako pri ostatných variantách diagramov. Zaujímavá je takisto otázka paralelizácie knižnice.

Literatúra

[1] Akers, S. B. Binary Decision Diagrams. *IEEE Trans. Comput.* Washington, DC, USA: IEEE Computer Society. jún 1978, zv. 27, č. 6, s. 509–516. ISSN 0018-9340.

[2] Babar, J., Jiang, C., Ciardo, G. a Miner, A. Binary Decision Diagrams with Edge-Specified Reductions. In: Vojnar, T. a Zhang, L., ed. *Tools and Algorithms for the Construction and Analysis of Systems.* Cham: Springer International Publishing, 2019, s. 303–318. ISBN 978-3-030-17465-1.

[3] Brace, K. S., Rudell, R. L. a Bryant, R. E. Efficient Implementation of a BDD Package. In: *Proceedings of the 27th ACM/IEEE Design Automation Conference.* New York, NY, USA: Association for Computing Machinery, 1991, s. 40–45. DAC '90. ISBN 0897913639.

[4] Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers.* Aug 1986, C-35, č. 8, s. 677–691. ISSN 2326-3814.

[5] Bryant, R. E. Symbolic Boolean Manipulation with Ordered Binary-decision Diagrams. *ACM Comput. Surv.* New York, NY, USA: ACM. september 1992, zv. 24, č. 3, s. 293–318. ISSN 0360-0300.

[6] Bryant, R. E. Binary decision diagrams and beyond: Enabling technologies for formal verification. *Proc. of the Intl. Conf. on Computer-Aided Design (ICCAD).* IEEE,Piscataway. 1995, s. 236–243.

[7] Bryant, R. E. Binary Decision Diagrams. In: Clarke, E. M., Henzinger, T. A., Veith, H. a Bloem, R., ed. *Handbook of Model Checking.* Cham: Springer International Publishing, 2018, s. 191–217. ISBN 978-3-319-10575-8.

[8] Bryant, R. E. Chain Reduction for Binary and Zero-Suppressed Decision Diagrams. In: Beyer, D. a Huisman, M., ed. *Tools and Algorithms for the Construction and Analysis of Systems.* Cham: Springer International Publishing, 2018, s. 81–98. ISBN 978-3-319-89960-2.

[9] Drechsler, R. a Sieling, D. Binary decision diagrams in theory and practice. *International Journal on Software Tools for Technology Transfer.* May 2001, zv. 3, č. 2, s. 112–136. ISSN 1433-2779.

[10] Knuth, D. E. *Fun With Binary Decision Diagrams [online].* Jún 2008. https://www.youtube.com/watch?v=SQE21efsf7Y&list=PL94E35692EB9D36F3&index=48&t=1920s.

[11] Knuth, D. E. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*. 12th. Addison-Wesley Professional, 2009. ISBN 0321580508.

[12] Lee, C. Y. Representation of switching circuits by binary-decision programs. *The Bell System Technical Journal*. July 1959, zv. 38, č. 4, s. 985–999. ISSN 0005-8580.

[13] Minato, S.-i. Zero-suppressed BDDs for Set Manipulation in Combinatorial Problems. In: *Proceedings of the 30th International Design Automation Conference*. New York, NY, USA: ACM, 1993, s. 272–277. DAC '93. ISBN 0-89791-577-1.

[14] Sieling, D. a Wegener, I. Reduction of OBDDs in Linear Time. *Inf. Process. Lett.* USA: Elsevier North-Holland, Inc. november 1993, zv. 48, č. 3, s. 139–144. ISSN 0020-0190.

[15] Somenzi, F. Efficient manipulation of decision diagrams. *STTT*. Január 2001, zv. 3, s. 171–181. DOI: 10.1007/s100090100042.

[16] van Dijk, T., Wille, R. a Meolic, R. Tagged BDDs: Combining Reduction Rules from Different Decision Diagram Types. In: *Proceedings of the 17th Conference on Formal Methods in Computer-Aided Design*. Austin, Texas: FMCAD Inc, Oct 2017, s. 108–115. FMCAD '17. ISBN 9780983567875.

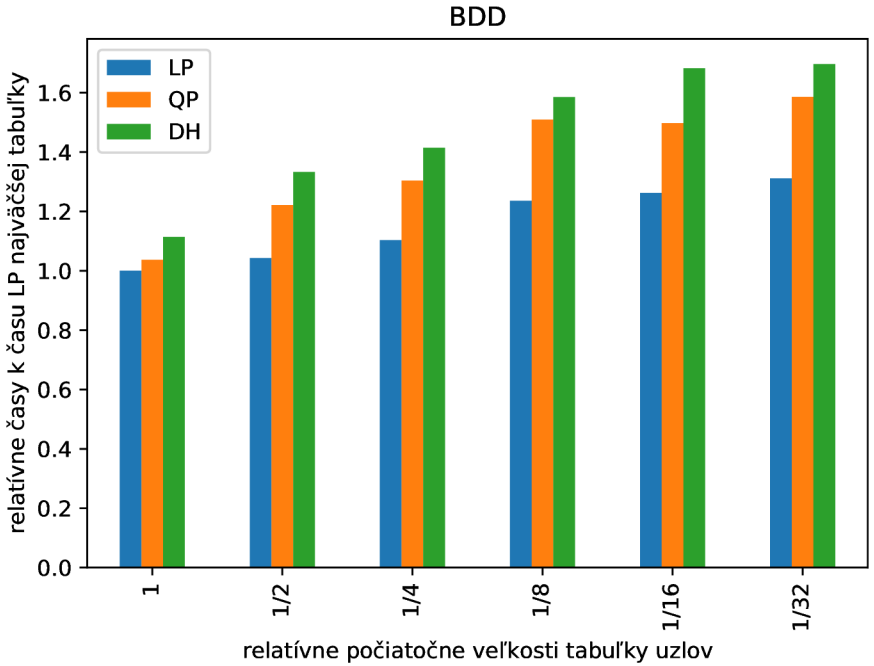
[17] Vitter, J. S. Implementations for Coalesced Hashing. *Commun. ACM*. New York, NY, USA: Association for Computing Machinery. december 1982, zv. 25, č. 12, s. 911–926. ISSN 0001-0782.

Príloha A

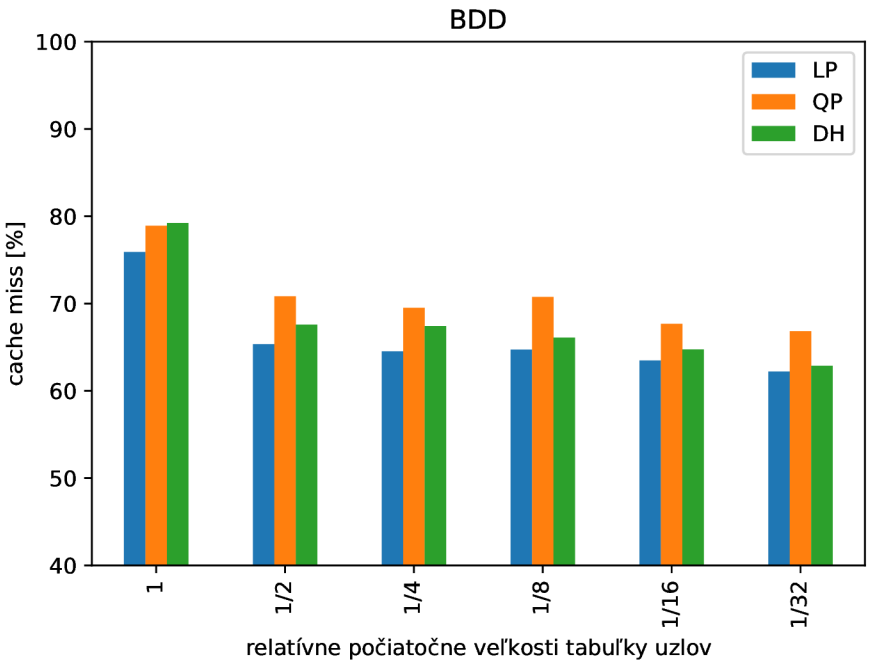
HW cache-miss a CPU čas

Táto príloha obsahuje na nasledujúcich stranách grafy k tomu ako sa mení percentuálna hodnota cache-miss hardwarovej cache PC a CPU čas pre jednotlivé typy diagramov vzhľadom na zmenšujúcu sa počiatočnú veľkosť tabuľky uzlov. Cache-miss sú v percentách a CPU čas je vyjadrený ako relatívna hodnota vzhľadom na CPU čas techniky linear probing pre tabuľku s najväčšou veľkosťou. Teda CPU čas pre linear probing najväčšej tabuľky je vždy hodnota 1. Význam skratiek:

1. LP - linear probing,
2. QP - quadratic probing,
3. DH - double hashing.

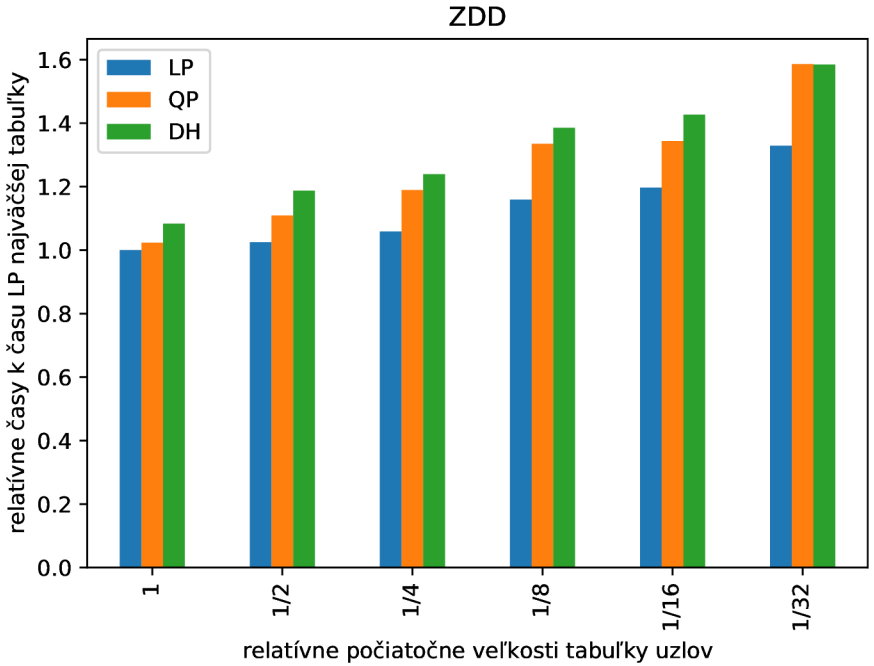


(a) CPU čas

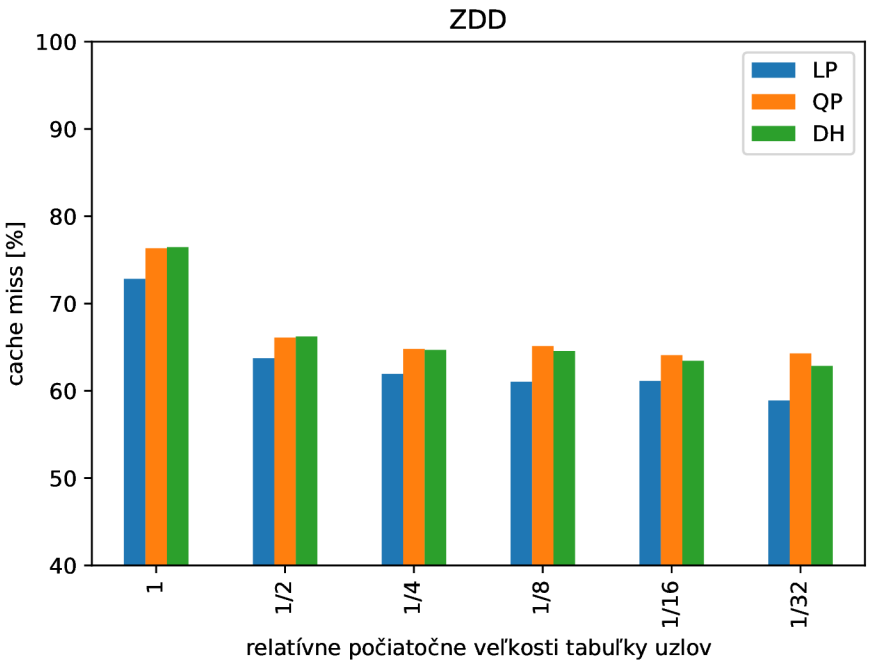


(b) cache-miss

Obr. A.1: CPU čas a cache-miss pre BDD

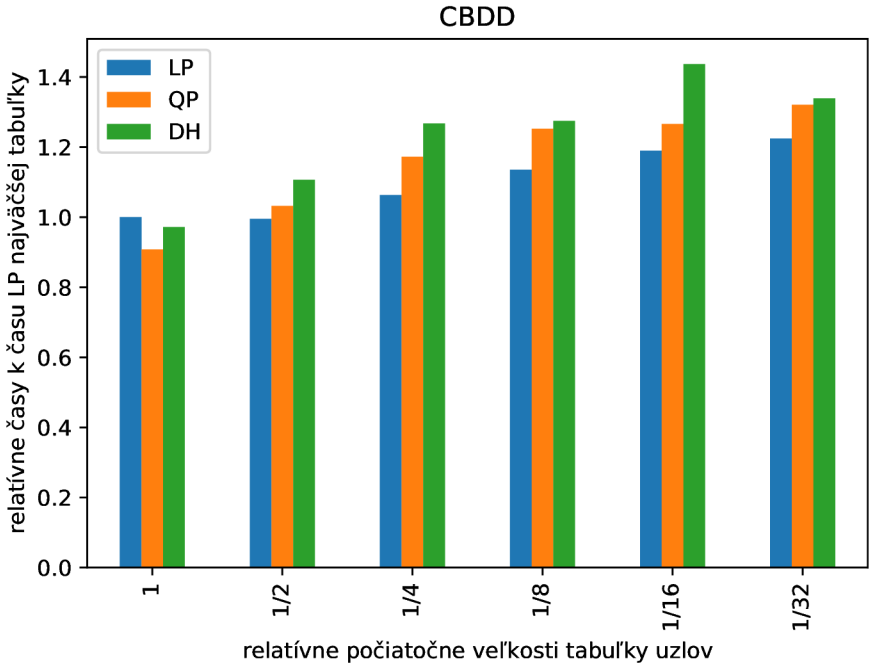


(a) CPU čas

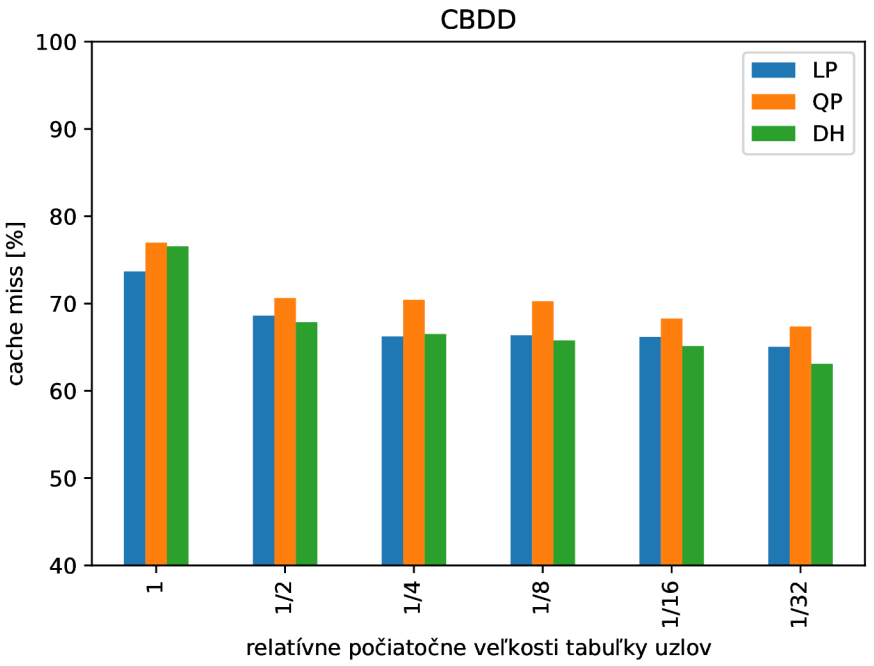


(b) cache-miss

Obr. A.2: CPU čas a cache-miss pre ZDD

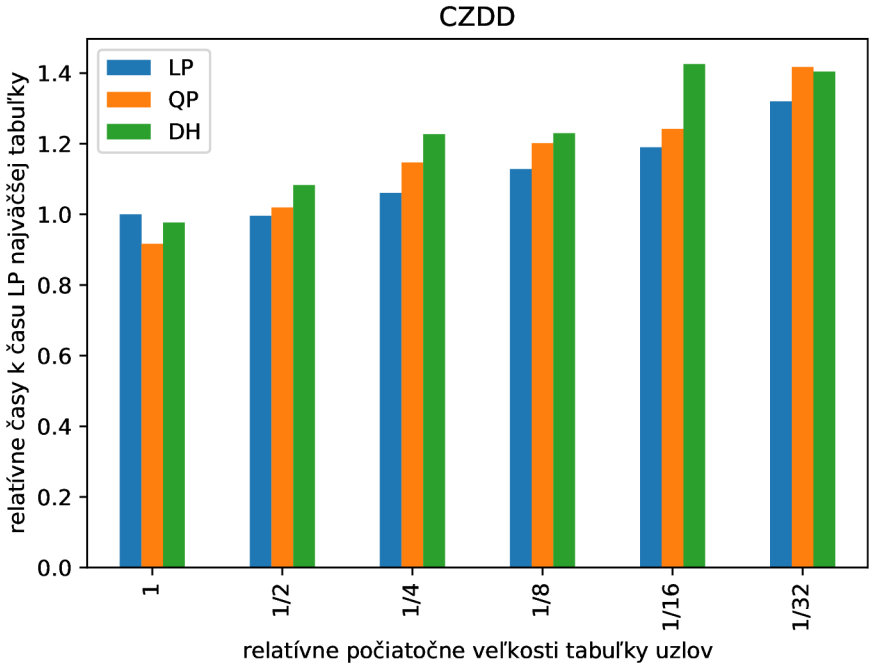


(a) CPU čas

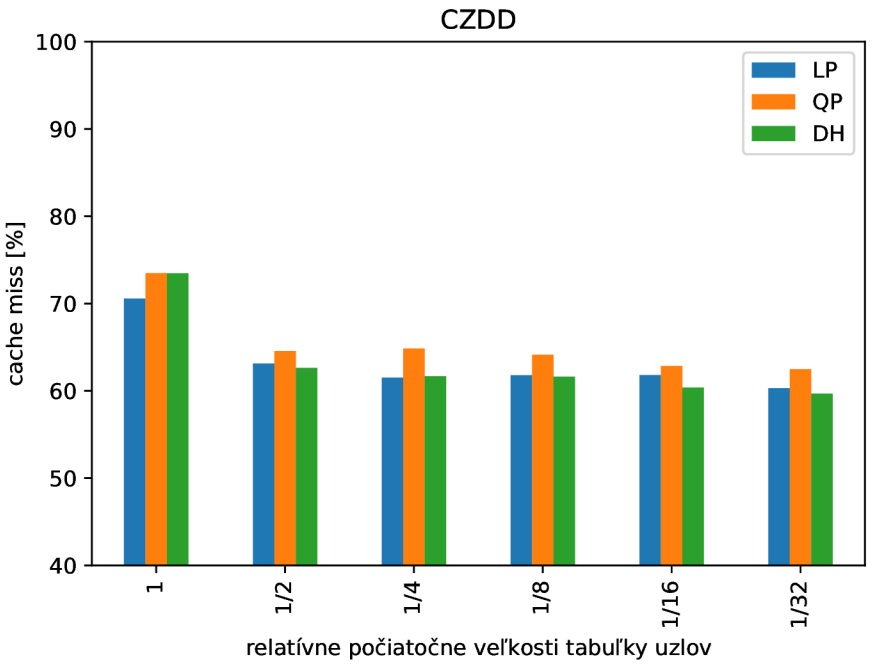


(b) cache-miss

Obr. A.3: CPU čas a cache-miss pre CBDD

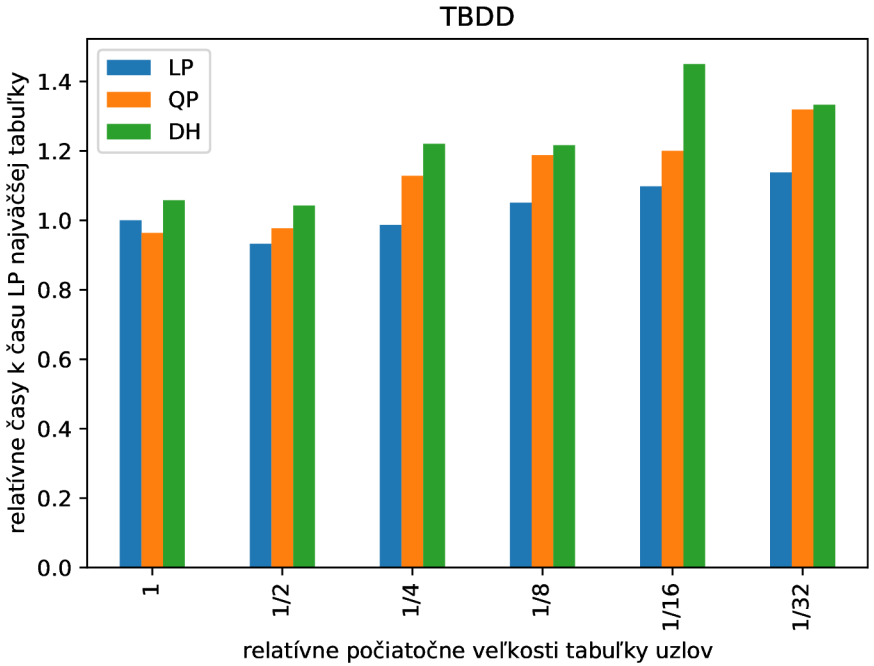


(a) CPU čas

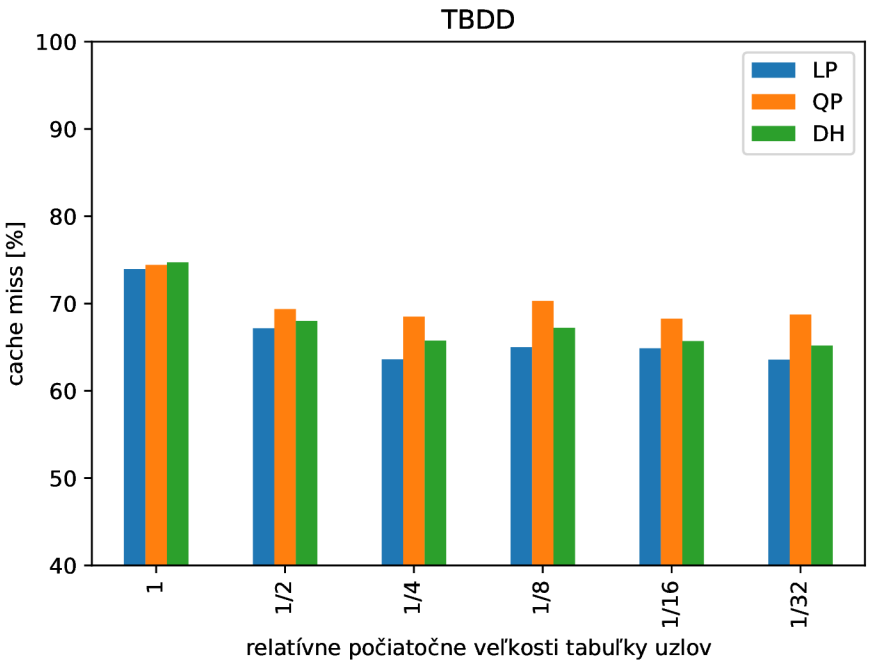


(b) cache-miss

Obr. A.4: CPU čas a cache-miss pre CZDD

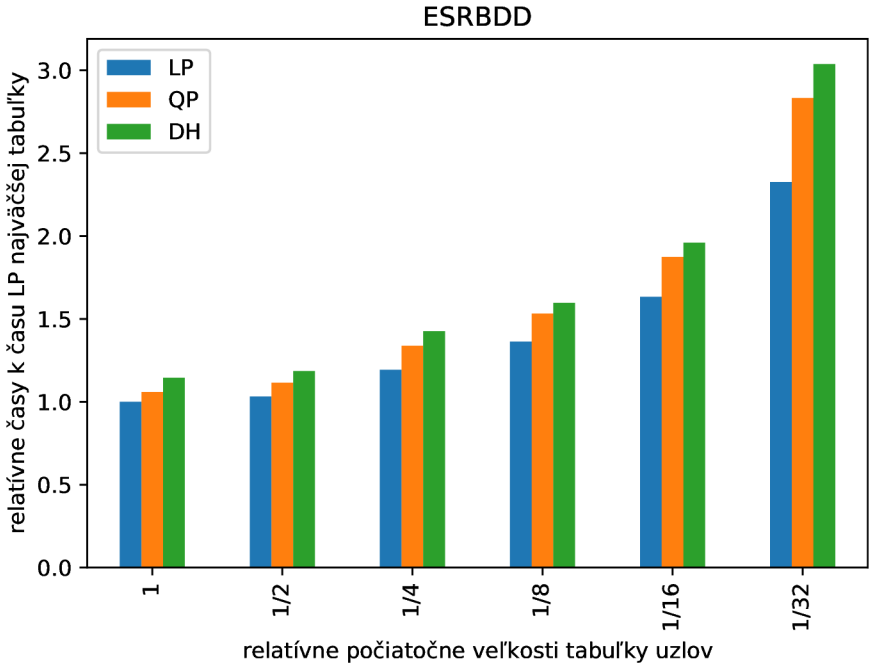


(a) CPU čas

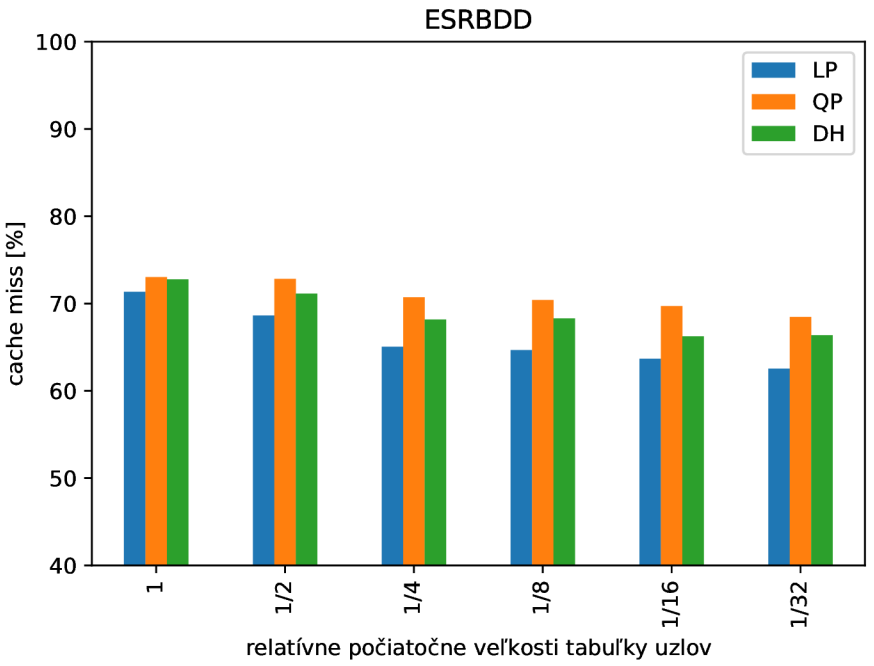


(b) cache-miss

Obr. A.5: CPU čas a cache-miss pre TBDD



(a) CPU čas



(b) cache-miss

Obr. A.6: CPU čas a cache-miss pre ESRBDD

Príloha B

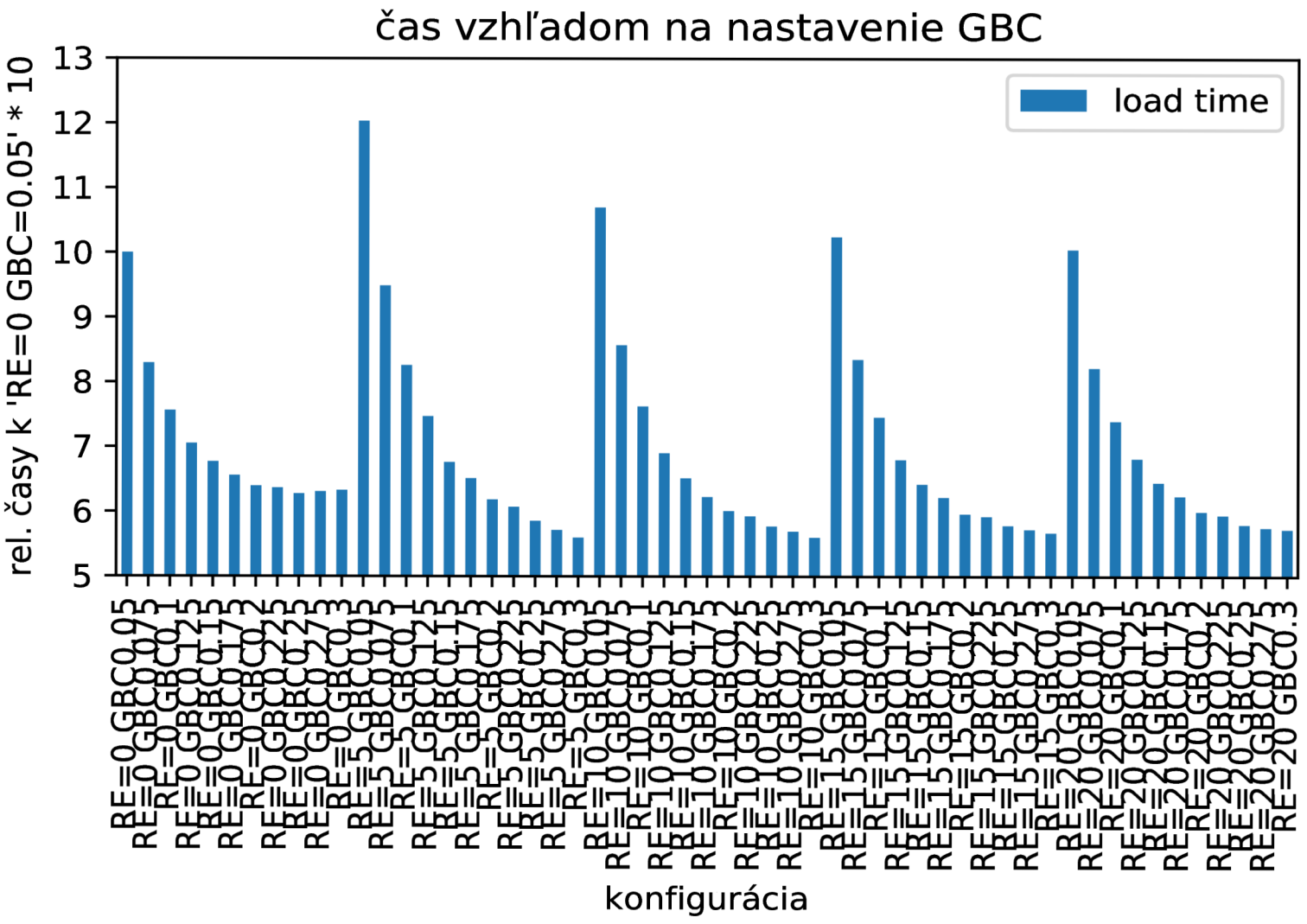
Garbage collector

V tejto prílohe sú grafy a tabuľka, ktoré popisujú ako vplýva nastavenie garbage collectora na celkový výkon diagramov meraný ako CPU čas a veľkosť tabuľky uzlov meraný v uzloch. `RE=X`, kde `X` je číslo, znamená po koľkých behoch garbage collectora sa spustí reorganizácia tabuľky uzlov, kvôli skráteniu kolíznych reťazcov. Hodnota `0` znamená, že táto funkcia je vypnutá. `GBC X`, kde `X` je číslo, znamená limit kedy sa spúšťa garbage collector. Tento limit vyjadruje podiel počtu vytvorených uzlov od posledného behu garbage collectora ku celkovej kapacite tabuľky uzlov. `RE=15 GBC0.2` napríklad znamená, že na spustenie garbage collectora je potrebné aby sa vytvorilo 20% uzlov z celkovej kapacity tabuľky a po 15 behoch sa reorganizuje tabuľka uzlov.

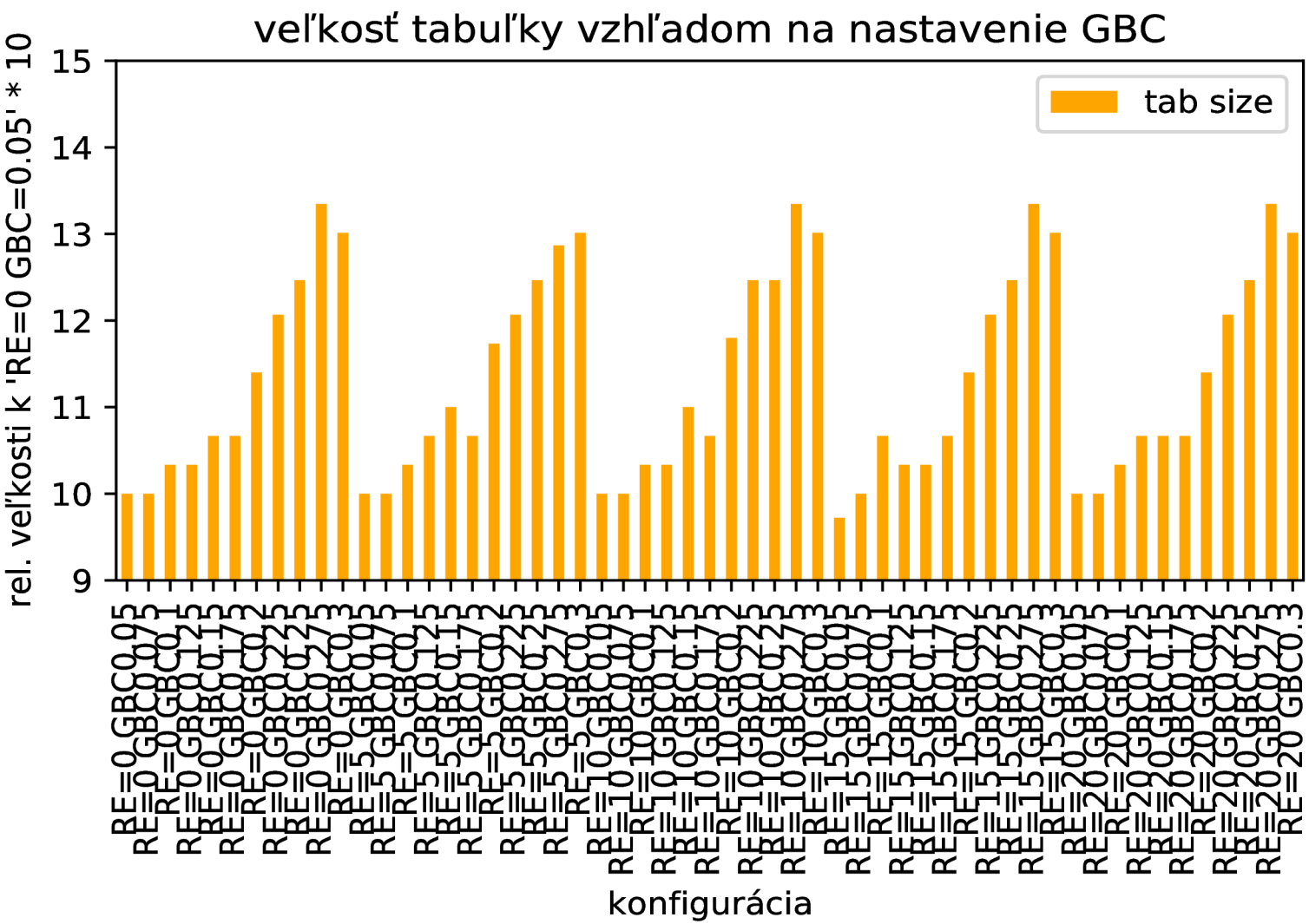
Z experimentov vyplynulo, že vyšší limit pre garbage collector znamená lepší celkový čas a to jednak z dôvodu amortizácie režijnych nákladov garbage collecotra, no hlavne pre lepšiu výkonnosť computed cache. No na druhej strane s menej častým garbage collectingom sa zvyšujú pamäťové nároky – sú potrebné väčšie tabuľky uzlov. Z experimentov takisto vidno, že občasná reorganizácia tabuľky uzlov takisto zlepšuje celkový čas. Hodnoty v tabuľke i grafoch sú relatívne hodnoty času a kapacity uzlov vztiahnuté k referenčnému nastaveniu `RE=0 GBC0.05`.

	setup	load time	tab size
0	RE=0 GBC0.05	10.000000	10.000000
1	RE=0 GBC0.075	8.293335	10.000000
2	RE=0 GBC0.1	7.559884	10.333333
3	RE=0 GBC0.125	7.051538	10.333333
4	RE=0 GBC0.15	6.769421	10.666667
5	RE=0 GBC0.175	6.558213	10.666667
6	RE=0 GBC0.2	6.394480	11.400000
7	RE=0 GBC0.225	6.365315	12.066666
8	RE=0 GBC0.25	6.275633	12.466666
9	RE=0 GBC0.275	6.308120	13.346666
10	RE=0 GBC0.3	6.329855	13.013333
11	RE=5 GBC0.05	12.034057	10.000000
12	RE=5 GBC0.075	9.489685	10.000000
13	RE=5 GBC0.1	8.259659	10.333333
14	RE=5 GBC0.125	7.471164	10.666667

15	RE=5 GBC0.15	6.764107	11.000000
16	RE=5 GBC0.175	6.513278	10.666667
17	RE=5 GBC0.2	6.186196	11.733333
18	RE=5 GBC0.225	6.073618	12.066666
19	RE=5 GBC0.25	5.857440	12.466666
20	RE=5 GBC0.275	5.715176	12.866666
21	RE=5 GBC0.3	5.598527	13.013333
22	RE=10 GBC0.05	10.698677	10.000000
23	RE=10 GBC0.075	8.571326	10.000000
24	RE=10 GBC0.1	7.626525	10.333333
25	RE=10 GBC0.125	6.904015	10.333333
26	RE=10 GBC0.15	6.517574	11.000000
27	RE=10 GBC0.175	6.229540	10.666667
28	RE=10 GBC0.2	6.013407	11.800000
29	RE=10 GBC0.225	5.931504	12.466666
30	RE=10 GBC0.25	5.775904	12.466666
31	RE=10 GBC0.275	5.695527	13.346666
32	RE=10 GBC0.3	5.603151	13.013333
33	RE=15 GBC0.05	10.245507	9.722222
34	RE=15 GBC0.075	8.352404	10.000000
35	RE=15 GBC0.1	7.459858	10.666667
36	RE=15 GBC0.125	6.803107	10.333333
37	RE=15 GBC0.15	6.424505	10.333333
38	RE=15 GBC0.175	6.221224	10.666667
39	RE=15 GBC0.2	5.965512	11.400000
40	RE=15 GBC0.225	5.927903	12.066666
41	RE=15 GBC0.25	5.787659	12.466666
42	RE=15 GBC0.275	5.725272	13.346666
43	RE=15 GBC0.3	5.674859	13.013333
44	RE=20 GBC0.05	10.051096	10.000000
45	RE=20 GBC0.075	8.221005	10.000000
46	RE=20 GBC0.1	7.399641	10.333333
47	RE=20 GBC0.125	6.819961	10.666667
48	RE=20 GBC0.15	6.451238	10.666667
49	RE=20 GBC0.175	6.242437	10.666667
50	RE=20 GBC0.2	6.002929	11.400000
51	RE=20 GBC0.225	5.946282	12.066666
52	RE=20 GBC0.25	5.800393	12.466666
53	RE=20 GBC0.275	5.754082	13.346666
54	RE=20 GBC0.3	5.727811	13.013333



Obr. B.1: Vývoj CPU času diagramov pre rôzne nastavenia garbage collectora.



Obr. B.2: Pamäťové nároky diagramov pre rôzne nastavenia garbage collectora.

Príloha C

Príklad dopadu Computed cache

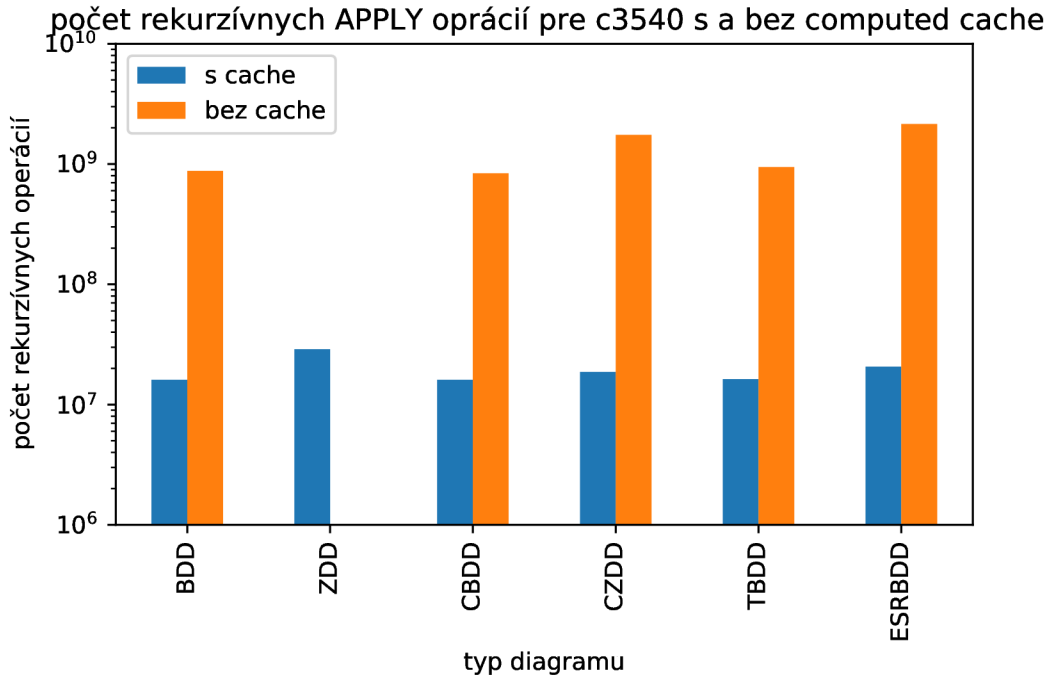
V tejto prílohe je v náväznosti na sekciu 3.5 príklad dopadu computed cache na celkový počet rekurzívnych APPLY operácií. Jeden príklad je zostrojenie diagramu pre obvod ISCAS85 c3540 a druhý pre zostrojenie diagramu reprezentujúceho všetky riešenia problému 8 kráľovien. ZDD nemajú bez využitia cache hodnoty, pretože sa nezmestili do limitu 7 minút. Príčina prečo je to tak je naznačená v sekcii 3.5. Na týchto príkladoch možno vidieť, že computed cache môže znížiť počet rekurzívnych operácií niekedy až niekoľko tisíc násobne a to sa jedná o relatívne maličké diagramy. Nasledujúce dve tabuľky uvádzajú konkrétne čísla udávajúce počet rekurzívnych `APPLY` operácií pre dve vyššie spomínané diagramy.

c3540

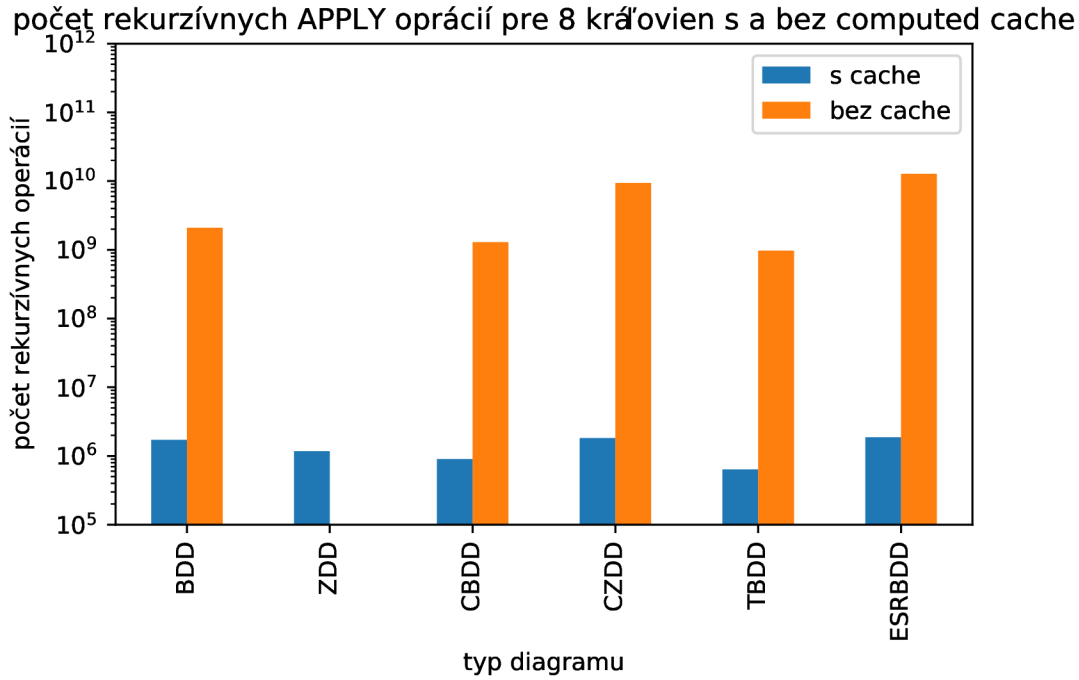
typ	s cache	bez cache
BDD	16062888	877117984
ZDD	28876081	N/A
CBDD	16039412	836807466
CZDD	18679134	1748167012
TBDD	16246227	944681521
ESRBDD	20678128	2152854830

8 kráľovien

typ	s cache	bez cache
BDD	1713700	2095349444
ZDD	1168828	N/A
CBDD	898784	1293581104
CZDD	1822370	9375635682
TBDD	634420	971310090
ESRBDD	1870272	12755260566



Obr. C.1: Počet APPLY operácií pre zostrojenie diagramu pre obvod c3540 s a bez computed cache.



Obr. C.2: Počet APPLY operácií pre zostrojenie diagramu pre všetky riešenia problému 8 kráľovien s a bez computed cache.

Príloha D

Problém N kráľovien

Tabuľka D.1: Počty uzlov výsledných grafov reprezentujúcich všetky riešenia problému N kráľovien pre $N = 7 \ldots, 14$.

	7	8	9	10	11	12	13	14
BDD	1101	2453	9559	25947	94824	435172	2044396	9572420
ZDD	188	375	1311	3122	10505	45835	204783	911422
CBDD	386	772	2795	6601	22455	98900	445755	1992396
CZDD	188	375	1311	3122	10505	45835	204783	911422
TBDD	188	375	1311	3122	10505	45835	204783	911422
ESRBDD	188	373	1306	3113	10477	45706	204345	909616

Tabuľka D.2: CPU časy potrebné na zostrojenie výsledných grafov reprezentujúcich všetky riešenia problému N kráľovien pre $N = 7, \ldots, 14$.

	7	8	9	10	11	12	13	14
BDD	0.021	0.072	0.297	2.449	14.582	84.009	528.889	4047.010
ZDD	0.017	0.047	0.149	1.282	6.850	38.589	230.553	1536.251
CBDD	0.020	0.053	0.212	1.359	7.734	42.050	240.450	1517.024
CZDD	0.023	0.082	0.356	2.807	15.346	86.416	512.167	3151.925
TBDD	0.017	0.035	0.117	0.746	4.095	24.629	150.794	904.920
ESRBDD	0.032	0.127	0.655	4.594	26.295	148.961	1082.823	12141.448

Tabuľka D.3: Počet rekurzívnych operácií `APPLY` potrebných na zostrojenie diagramov reprezentujúcich všetky riešenia problému N kráľovien pre $N = 7, \ldots, 14$.

	7	8	9	10	11
BDD	217,618	956,876	4,784,324	23,277,026	125,576,829
ZDD	274,486	1,136,664	5,495,531	25,613,141	136,533,748
CBDD	136,206	545,408	2,492,882	11,261,111	57,322,238
CZDD	252,141	1,093,628	5,427,243	25,465,997	136,274,198
TBDD	120,430	478,352	2,139,620	9,820,184	50,516,581
ESRBDD	433,794	1,906,952	9,946,280	45,349,365	244,154,695

	12	13	14
BDD	680,640,592	3,867,271,348	21,934,758,195
ZDD	734,333,991	4,256,206,059	24,022,730,093
CBDD	295,365,533	1,613,288,513	8,744,432,499
CZDD	733,880,949	4,269,443,416	24,068,428,916
TBDD	267,178,425	1,492,517,968	8,318,248,736
ESRBDD	1,318,215,143	7,705,698,563	43,894,543,252

Príloha E

Vybrané SATLIB benchmarky

V tejto prílohe sú zobrazené výsledky vybraných SATLIB benchmarkov, ktoré sú dostupné na https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html. Prvý riadok daného benchmarku v tabuľke vyjadruje CPU čas potrebný na vytvorenie diagramu a druhý počet jeho uzlov.

Tabuľka E.1: Vybrané *SATLIB* benchmarky.

bench.	BDD	ZDD	CBDD	CZDD	TBDD	ESRBDD
aim	0.634	**0.304**	0.833	0.780	0.937	1.138
	202	82	124	82	82	**62**
ais	25.628	11.755	12.484	27.006	**5.910**	46.143
	25,618	2,881	5,762	2,881	2,881	**2,857**
d60	81.413	114.138	86.668	98.454	**61.726**	145.631
	1	1	1	1	1	1
d66	493.560	688.063	496.837	527.96	**473.981**	1223.359
	1	1	1	1	1	1
fgc	293.947	177.493	275.074	314.819	**132.361**	571.751
	11,136	3,992	7,668	3,992	3,992	**3,958**
qg-m	245.602	552.349	**208.022**	674.580	273.999	584.928
	308,396,305	430,308,312	245,437,517	422,116,970	308,396,305	**292,930,609**
uf50	**29.670**	46.598	34.400	52.675	30.227	65.662
	64	36	48	36	36	**24**
par	6.408	**5.173**	5.719	8.276	11.082	24.892
	325	81	111	81	81	**48**
pret	17.782	**15.699**	20.731	23.821	29.344	60.265
	1	1	1	1	1	1

Zoznam jednotlivých súborov:

1. aim - AIM : Artificially generated Random-3-SAT (`aim-200-6_0-yes1-4.cnf`)
2. ais - All Intervall Series (`ais10.cnf`)
3. d60 - DUBOIS: Randomly generated SAT instances (`dubois20.cnf`)
4. d66 - DUBOIS: Randomly generated SAT instances (`dubois22.cnf`)
5. fgc - Flat Graph Colouring (`flat50-1.cnf`)

6. qg-m - SAT-encoded Quasigroup (`qg1-07.cnf`), znížený počet klauzulí
7. uf50 - Uniform Random-3-SAT, 50 premenných 218 klauzulí (`uf50-019.cnf`)
8. par - PARITY: Instances for problem in learning the parity function (`par8-1.cnf`)
9. pret - PRET: Encoded 2-colouring forced to be unsatisfiable (`pret60_25.cnf`)

Príloha F

Obsah priloženého CD

Priložené CD obsahuje tieto súbory a adresáre:

- *ddlib/* - priečinok obsahujúci zdrojové kódy vytvorenej knižnice
- *DOC/* - priečinok obsahujúci dokumentáciu k vytvorenej knižnici
- *tex/* - priečinok obsahujúci súbory potrebné ku kompilácii toho to textu
- *LICENSE* - súbor obsahujúci licenciu k projektu
- *README.md* - krátky popis knižnice a jej použitia
- *thesis.pdf* - text tejto práce v elektronickej podobe