ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Bakalářská práce:

Minimalizace neúplně určených logických funkcí pomocí binárních rozhodovacích diagramů

Vypracoval: Martin Blum

Vedoucí práce: Ing. Petr Fišer, Ph.D.

Bakalářský program: Elektrotechnika a informatika, strukturovaný
Obor: Informatika a výpočetní technika
2009

Poděkování

Chtěl bych poděkovat především za trpělivost a pomoc s prací svému vedoucímu, Ing. Petru Fišerovi, Ph.D.

Čestné prohlášení

Prohlašuji, že jsem zadanou bakalářskou práci zpracoval sám samostatně a používal jsem pouze podklady uvedené v přiloženém seznamu.

Dále prohlašuji, že nemám námitek proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne .....

podpis

Abstrakt

Bakalářská práce pojednává o minimalizaci neúplně zadaných logických funkcí pomocí modifikovaných binárních rozhodovacích diagramů. Pro potřeby minimalizace je použit CUDD balík (Colorado University Decision Diagram Package) na operace s binárními stromy, minimalizátor Espresso a algoritmy vytvořené Janem Bílkem a Pavlem Černým v předešlých pracích pojednávajících na dané téma.

Abstract

Bachelor's thesis deals with the minimalization of incompletely specificed boolean functions using modified binary decision diagrams. For the purpose of minimalization, CUUD package (Colorado University Decision Diagram Package) for operations with binary trees, minimizer Espresso and algorithms created by Jan Bílek and Pavel Černý in previous thesis dealing with the same topic are used.

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

Výchozí projekt byl převzat od Pavla Černého[1]. Úkolem bylo urychlit zpracování tohoto programu nahrazením externího voláni minimalizátoru interním (knihovním) a dále provést diskuzi nad úrovní minimalizace a časové náročnosti v závislosti na volbě pořadí proměnných při tvorbě BDD. Snažit se o co nejlepší minimalizaci v co nejkratším čase variací vstupních proměnných a modifikací knihovny Espressa.

1.1 Účel práce

Minimalizace logických funkcí je velmi obtížná úloha. Spočívá v nalezení množiny přímých implikantů a následném výběru minimálního počtu přímých implikantů, které danou funkci definují. Existuje celá řada minimalizátorů, avšak jejich časové nároky i úroveň minimalizace se liší. Zde je realizován další způsob a jeho srovnání s ostatními používanými minimalizátory.

1.2 Popis řešení

Minimalizace spočívá v načtení všech logických funkcí do jediného BDD, jejich následné minimalizaci každé zvlášť a sjednocení výsledku. V BDD reprezentuje každou výstupní funkci vlastní kořen stromu. Podstromy grafu se snažíme co nejvíce sdílet s ostatními funkcemi, čímž dosáhneme rychlejší paralelní minimalizace funkcí. Získaná data jsou spojeny v jediný soubor. V BDD je snaha docílit co nejmenšího množství vnitřních uzlů negovanými hranami. Práce s BDD je reprezentována CUDD balíčkem a minimalizace uzlů je reprezentována Espressem.

1.3 Výchozí stav projektu

Po detailním prozkoumání projektu jsem zjistil zásadní nedostatky v projektu, z kterého vycházím. Dosahuje sice srovnatelných výsledků úrovně minimalizace

v případě minimalizace jednovýstupového pla neobsahujícího DCset, avšak nedosahuje srovnatelných výsledků minimalizace pla obsahujícího více výstupních funkcí. DCset je chybně definován a při minimalizaci je jeho role zcela potlačena. Projekt nedosahuje co se minimalizace i času týče ani zdaleka úrovně srovnatelné s Espressem.

1.4 Úskalí řešení

Minimalizace provedená tímto způsobem skýtá několik problémů. Zde je výčet některých z objevených nedostatků.

Konstrukcí BDD a postupnou minimalizací každé výstupní funkce zvlášť, se ztrácí informace, zdali je funkce pro termy obsazené v jiných výstupních funkcích 0 nebo X. Proto dochází k mnohem horším výsledkům pro skupinovou minimalizaci neúplně zadaných funkcí.

Jedním z nejhorších aspektů je nezohledňování jiných možností průniků množin implikantů výstupních funkcí a z toho plynoucí nárůst počtu termů na výstupu u vícevýstupových funkcí. BDD nabízí pouze jediné možné řešení, proto nelze zkoumat možnost jiného pokrytí ONsetu výstupních funkcí pro dosažení lepšího výsledku.

Dalším z problémů, i když méně závažný, je velká neefektivita na funkce neobsahující DCset. Tento problém by šel vyřešit zakázáním negovaných hran, procházením stromu, ručním sestavením termů a použitím pravidla absorpce negace pro každý uzel, jehož potomkem je 1 (předpokládána neužívání negovaných hran) a následné zavolání externího minimalizátoru na kořen stromu. Dalším z možných řešení by byl jiný minimalizátor navržený přímo za tímto účelem. Všechny tyto návrhy by znamenaly potlačit negované hrany stromu. To lze provést pouze diky vlastnosti CUDD, které se snaží udržovat strom minimální a tím minimalizuje počet termů a množství literálů v termech.

Efekt vzniklý BDD a jejich prolínáním, jenž zaručuje více společných termů je částečně potlačen využitím negovaných hran a taktéž komplikuje problém minimalizace vnitřních uzlů. V některých testech se prokázal nárůst množství termů při využití negovaných hran.

Z testování bylo patrné, že minimalizovaná funkce má u kořene někdy větší složitost nežli původní, a proto externí minimalizátor musí řešit složitější problém než

počáteční, což zajisté není účel práce. Tento efekt přisuzuji negovaným hranám nebo nekorektní manipulací s nimi v předešlých částech práce.

Dalším poznatkem z testování bylo, že aplikace občas u úplně zadaných funkcí dosáhne horšího výsledku, nežli Espresso samotné. Jinými slovy dostane se minimalizací funkce do stavu, který Espresso není schopné řešit. Jde o konflikt minimální počet termů versus větší smyčky.

1.5 Cíl práce

Optimalizovat řešení pro dosažení stávající či lepší úrovně minimalizace za nižší čas. Umožnit volání interních minimalizátorů. Napravit chyby v předešlé implementaci. Upravit Espresso jako dynamickou knihovnu použitelnou jako externí minimalizátor. Porovnat výsledky s předešlou prací a výhodnost tohoto způsobu řešení zadané problematiky.

2 Základní pojmy a definice

Zde jsou vysvětleny základní pojmy vztahující se k problematice. Účel této kapitoly je obeznámit čtenáře s terminologií používanou v dalších kapitolách a nezbytnou pro pochopení práce.

2.1 Booleova algebra

Logické funkce jsou funkce, ve kterých jednotlivé proměnné nabývají hodnot {0, 1} neboli {false,true}. Výsledkem takovéto funkce je opět 1 nebo 0.

Matematicky zapsáno f : $\{0,1\}^n$ -> {0,1}, kde f je logická funkce o n vstupech.
Booleova algebra je algebraická struktura popisující vlastnosti množinových a logických operací.
Skládá se z:

- množiny logických proměnných
- binárních operací + (disjunkce) nebo * (konjunkce)
- unární operace ¯ nebo ¬ (negace)
- konstant 0 a 1

Základní pravidla:

- asociativita: (x + y) + z = x + (y + z), (x * y) * z = x * (y * z)
- komutativita: x + y = y + x, x * y = y * x
- distributivní zákon: x + (y * z) = (x + y) * (x + z), x * (y + z) = x * y + x * z
- absorpce: x + (x * y) = x, x * (x + y) = x, x + x = x, x * x = x
- absorpce negace: x + (¬x * y) = x + y, x * (¬x + y) = x * y
- agresivita nuly a jedničky: x * 0 = 0, x + 1 = 1
- komplementarita: x + ¬x = 1, x * ¬x = 0
- dvojitá negace: ¬(¬x) = x
- De Morganovy zákony: ¬x * ¬y = ¬(x + y), ¬x + ¬y = ¬(x * y)

Z těchto pravidel existuje i řada dalších odvozených, jedním z nich je i Shannonův expanzní teorém popsaný dále.

Pro práci s třístavovými funkcemi vypadají elementární funkce následovně.

+	0	1	X
0	0	1	X
1	1	1	1
X	X	1	X

Tabulka 2.1: Definice třístavového logického součtu

*	0	1	X
0	0	0	0
1	0	1	X
X	0	X	X

Tabulka 2.2: Definice třístavového logického součinu

	¬
1	0
0	1
X	X

Tabulka 2.3: Definice třístavové negace

2.2 Použité pojmy z této kapitoly

Literál – logická proměnná či její negace (x či ¬x)
Konjunkce – logický součin
Disjunkce – logický součet
Součinový term – logický výraz tvořený konjunkcí literálů
Součtový term – logický výraz tvořený disjunkcí literálů
Term – součtový nebo součinový. Zde bude toto označení používáno pro součtový term
Minterm – součinový term obsahující literály všech vstupních proměnných
Maxterm – součtový term obsahující literály všech vstupních proměnných
Disjunktivní normální forma (DNF) – logický výraz tvořený disjunkcí součtových termů
Konjunktivní normální forma (CNF) – logický výraz tvořený konjunkcí součinových termů
Úplná disjunktivní normální forma (UDNF) – logický výraz plně popisující funkci tvořený disjunkcí mintermů
Úplná konjunktivní normální forma (UCNF) – logický výraz plně popisující funkci f tvořený konjunkcí maxtermů
ONset – množina termů, pro které funkce nabývá hodnoty 1
OFFset – množina termů, pro které funkce nabývá hodnoty 0
DCset – množina termů, pro které nezáleží jaké hodnoty funkce nabývá
Implikant logické funkce – term, jenž je prvkem množiny ONset
Přímý implikant – takový implikant logické funkce, který již nelze zjednodušit
Shannonův expanzní teorém – jeho myšlenka spočívá v tom, že každou logickou funkci f lze rozložit $f(x_0, x_1, ...x_n) = x_0.f_{high}(1, x_1, ...x_n) + \neg x_0.f_{low}(0, x_1, ...x_n)$
Úplně určená logická funkce – definována množinami ONset a OFFset, jejichž průnik je nulový a sjednocení odpovídá všem kombinacím vstupních proměnných
Neúplně určená logická funkce – definována množinami ONset, OFFset a DCset (v praxi i pouze dvěma z nich), kde každá z těchto množin má prázdný průnik se zbývajícími a jejich sjednocení odpovídá všem kombinacím vstupních proměnných

2.3 Možnosti vyjádření logických funkcí

Logické funkce nejčastěji popisujeme:
- pravdivostní tabulkou
- logickým výrazem
- mapou
- vícerozměrnou jednotkovou krychlí
- binárním rozhodovacím diagramem

Pravdivostní tabulka – jde o kompletní tabulku obsahující všechny možné kombinace vstupních proměnných a pro ně definovaných výstupů
- velikost je 2^n kde n je počet vstupních proměnných
- jednoduše z ní lze vyčíst úplná konjunktivní nebo disjunktivní normální forma

Mapa – Karnaughova mapa se používá pro minimalizaci logické funkce
- obsahuje 2^n kde počet vstupních proměnných polí
- není vhodná pro minimalizaci funkcí více proměnných pro ztrátu přehlednosti

Vícerozměrná jednotková krychle – další z možných vyjádření logické funkce

Modifikované binární rozhodovací diagramy – vhodné pro strojovou minimalizaci. Je jim věnována samostatná kapitola.

3 Binární rozhodovací diagramy

Jde o další možnou formu reprezentace logické funkce, vycházející z Shannonova expanzního teorému. V této práci jsou použity binární rozhodovací diagramy (Binary Decision Diagrams – BDD), respektive jejich modifikace – uspořádané binární rozhodovací diagramy (Ordered Binary Decision Diagrams – OBDD) a redukované upořádané binární rozhodovací diagramy (Reduced Ordered Binary Decision Diagrams – ROBDD).

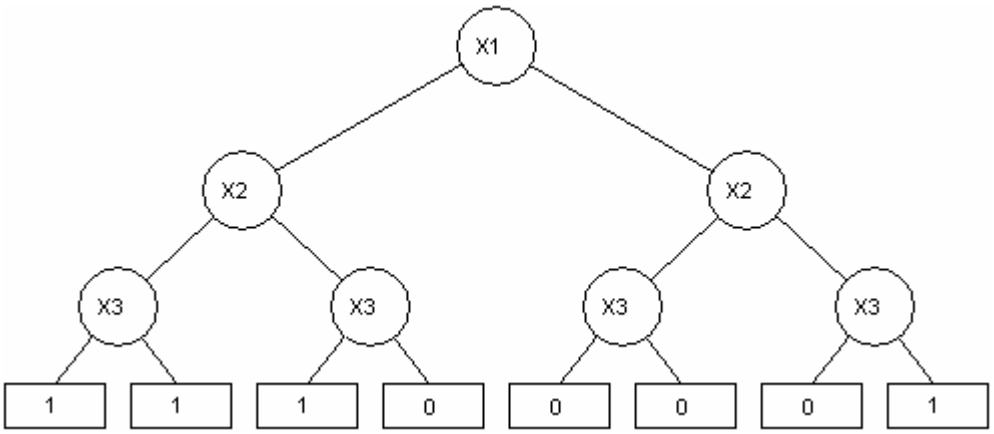
3.1 Historie BDD

Poprvé byla myšlenka binárních rozhodovacích diagramů prezentována C. Y. Leem ve článku "Representation of Switching Circuits by Binary-Decision Programs" [4] v časopise Bell Systems Technical Journal roku 1959 a dále prohloubena S. B. Aversem v publikaci "Binary Decision Diagrams" [5] roku 1978.

Další práce od R. E. Bryanta "Graph-Based Algorithms for Boolean Function Manipulation" [6] roku 1986 a "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams" [7] roku 1992 rozšiřovaly schopnosti BDD například o sdílení podgrafů.

3.2 BDD

BDD je kořenově orientovaný acyklický graf, kde každý vnitřní uzel má právě dva potomky. V této fázi BDD téměř odpovídá úplnému binárnímu stromu. Na každý uzel vede právě jedna reference, což nasvědčuje velkému množství redundantních informací, proto jsou v následujících částech popsány postupné modifikace těchto stromů pro potřeby minimalizace logických funkcí.



Obrázek 3.1: Úplný binární strom

3.3 OBDD

Velikost BDD je znatelně ovlivněna pořadím, v jakém se proměnné zpracovávají. BDD s určeným pořadím nazýváme uspořádané binární rozhodovací diagramy (OBDD).

3.4 ROBDD

Dalším krokem je redukce vnitřních uzlů OBDD. Redukované upořádané binární rozhodovací diagramy – ROBDD.
Případy redukce vnitřních uzlů:

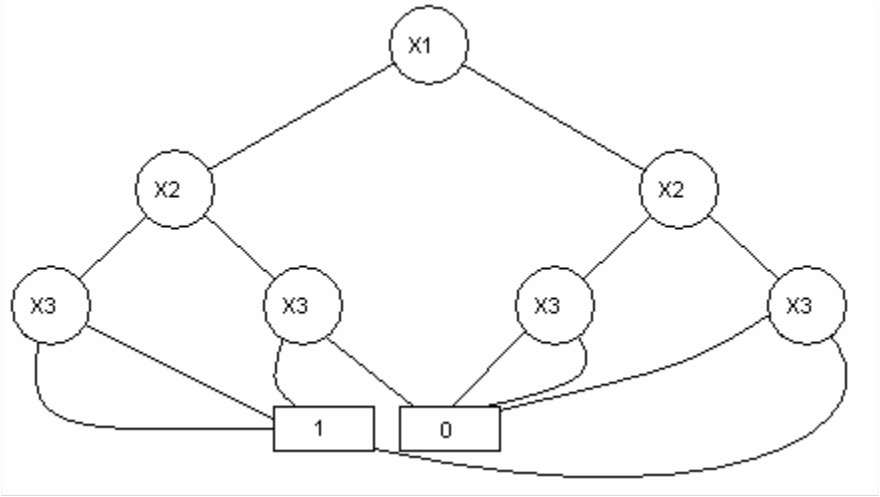
1. Odstranění všech nadbytečných listů grafu a ponechání pouze 1 (prvek ONset) a 0 (prvek OFFset).
2. Odstranění zbytečných uzlů – pokud si oba potomci daného uzlu odpovídají, můžeme referenci na daný uzel nahradit referencí na jeho potomka a daný uzel odstranit.
3. Odstranění stejných podgrafů – pokud si dva uzly odpovídají podmínkou i funkcemi f_{high} a f_{low} (Shannonův expanzní teorém), je zbytečné mít autentický podgraf v BDD vícekrát. Proto nadbytečný podgraf smažeme a reference na něj převedeme na autentický podgraf.

Na obrázku 3.1 je binární rozhodovací diagram, kde přecházíme na pravého potomka v případě, že je proměnná v uzlu 1 a na levého v případě že je 0.

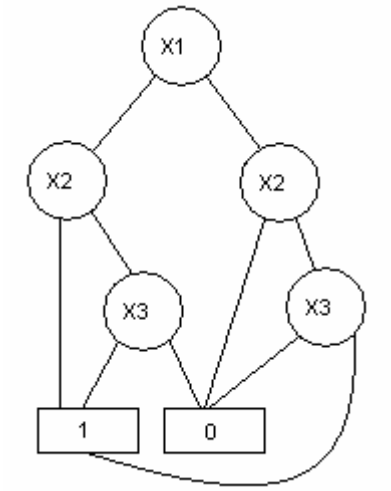
Po aplikaci prvního pravidla dostaneme strom na obrázku 3.2.

Po aplikaci druhého pravidla dostaneme strom na obrázku 3.3.

Třetí pravidlo zde nelze využít.



Obrázek 3.2: Binární strom po redukci listů



Obrázek 3.3: Binární strom po redukci uzlů

3.5 Operace s ROBDD

V této podkapitole jsou popsány operace s ROBDD, na které se budu odkazovat v následujících kapitolách, týkajících se CUDD balíčku.

Pro práci s ROBDD bylo nutné zavést operace s nimi manipulující. Základní dvě operace Apply a Restrict byly popsány v již zmíněné práci "Graph-Based Algorithms for Boolean Function Manipulation."

3.5.1 Operace Apply

Provede operaci mezi dvěma logickými funkcemi reprezentovanými ROBDD. Výsledkem takovéto operace je ROBDD. Je to rychlejší než zjišťovat logické funkce odpovídající ROBDD a provádět operaci nad těmito výsledky a znovu vytvářet ROBDD. Parametr Apply je jedna ze standardních logických funkcí AND, OR, XOR, NAND, NOR…

3.5.2 Operace Restrict

Tato operace slouží k nahrazení všech uzlů konkrétní proměnné v ROBDD jedním z jejich potomků. Jedná se tedy o dosazení za konkrétní proměnnou v ROBDD.

3.5.3 Operace Redukce

Tato operace vrací ROBDD z BDD.

3.6 ROBDD práce s neúplně určenými funkcemi

Pro práci s neúplně určenými logickými funkcemi bylo vyvinuto několik metod:

- dvojce ROBDD reprezentující množiny ONset a DCset
- rozšíření listu ROBDD o výskyt třetího stavu – prvku DCset
- zavedení speciální funkce na vyjádření prvku DCset

3.6.1 Neúplně určené funkce pomocí dvou ROBDD

Výhodou této metody je, že se nemusí nijak měnit způsob práce s ROBDD. Nevýhodou je vysoká redundance ve stromech, jelikož se snažíme minimalizovat dvě části úlohy namísto celku. Pokud ROBDD ONset vrací 1, pak je výsledek 1. Pokud 0, zpracovává se ROBDD DCset, který vrátí zdali je výsledkem neurčený stav či 0.

3.6.2 Neúplně určené funkce pomocí třetího stavu v listech

Pro tyto účely ROBDD rozšíříme o možnost výskytu X v listech. Z toho plyne, že nyní v listech mohou být hodnoty 1 (prvek ONset), 0 (prvek OFFset) a X (prvek DCset). Prvky DCset jsou reálně reprezentovány mnohdy různými symboly. V této práci budou označeny X. Tato metoda je snadná na implementaci a vede k možnosti redukovat celý problém v jednom grafu, proto lze více zredukovat ROBDD.

3.6.3 Neúplně určené funkce pomocí speciální funkce DCset

Zavede se speciální extended funkce vracející hodnotu, zdali daný uzel je prvkem DCset a zároveň není prvkem ONset. Metoda má dvě části. V první se hledá hodnota funkce pro každý uzel a vyjádří se pomocí ROBDD. V druhé se všechny reference přesměrují na nové. Jedná se o složitou metodu a její detailní popis je uveden v práci "Minimizing ROBDD size of incompletely specified multiple output functions" z roku 1994.

4 Colorado University Decision Diagram Package (CUDD package)

Vývoj balíku byl zahájen v roce 1966 byl vyvíjen Fabiem Somenzim a spolupracovníky na Coloradské univerzitě a vyvíjí se dodnes. Jeho poslední verze 2.4.2 je přístupná na stránkách http://vlsi.colorado.edu/~fabio/CUDD/ [8] a datována k 20.2.2009. V základním balíku je přes 400 funkcí a jsou k dispozici různá rozšíření řešící například načítání ze souborů. Tento balík je napsán v jazyce C, ale existuje také jeho C++ verze.

CUDD je jeden z nejpoužívanějších balíčků pro práci s rozhodovacími diagramy. Umožňuje pracovat s třemi druhy diagramů:

1. BDD binární rozhodovací diagramy
2. ADD algebraické rozhodovací diagramy
3. ZDD diagramy s potlačenou nulou

4.1 Správa paměti

Jednou z nesporných výhod balíčku je vlastní správa paměti. Po zavolání funkce Cudd_init() je inicializován DdManagment. DdManagment udržuje vlastní hashovaní tabulku s informacemi o každém uzlu. Při ukončení práce je nutné zavolat funkci Cudd_Quit(), tím DdManagment ukončíme.

4.2 Struktura uzlů

Všechny podporované diagramy používají strukturu DdNode. DdNode obsahuje informace o jméně proměnné reprezentovaného uzlu, počet referencí na uzel, ukazatel na další uzel v hashovací tabulce a pro případ vnitřního uzlu reference na potomky.

4.3 Načítání souboru typu pla

Tento formát souboru vyvinuli na universitě Berkeley. Soubor typu pla obsahuje dvouúrovňový popis logické funkce. Na počátku souboru je definován počet vstupů, počet výstupů, počet termů obsažených v souboru, typ určující jaké množiny jsou v souboru vyčteny. Za touto hlavičkou následuje seznam, jehož řádky odpovídají

vždy součinovým termům a odpovídajícím výstupům. Soubor má i vlastní ukončování.

Jelikož je tento formát použit jako komunikace mezi Espressem a CUDD, detailní popis pla formátu se nachází v příloze A.

5 Minimalizace logických funkcí

Minimalizace logických funkcí náročná úloha, především z hlediska nejednoznačnosti výsledku. Pro zadanou funkci lze určit minimální počet termů a minimální počet literálů každého termu, ale pro složitější funkce neexistuje pouze jediný výsledek, odpovídající těmto požadavkům. Minimalizovat se dá za účelem dosažení minimálního počtu literálů v termech nebo za účelem dosažení minimálního počtu termů.

5.1 Minimalizace logických funkcí

Pokud hovoříme o minimalizaci logických funkcí máme, na mysli dvouúrovňovou minimalizaci (Disjunktivní normální forma).

Minimalizace je proces, při kterém se snažíme snížit počet termů, zjednodušit termy tak, aby se v nich vyskytovalo co nejméně literálů, a zároveň zachovat tautologický ekvivalent s původní funkcí.

5.2 Skupinová minimalizace logických funkcí

Jde o autentický problém jako v předešlém případě. Opět se snažíme snížit počet termů, zjednodušit termy tak, aby se v nich vyskytovalo co nejméně literálů, a zároveň zachovat tautologický ekvivalent původních funkcí s minimalizovanými. Při skupinové minimalizaci se berou na zřetel všechny výstupní funkce. Proto většinou dosahuje pro jednotlivé výstupní funkce horší úrovně minimalizace, nežli když funkce minimalizujeme každou zvlášť. Vhodným využitím skupinové minimalizace je například pokud chceme realizovat jedním obvodem více logických funkcí.

5.3 Způsoby minimalizace

Dělí se na tradiční a heuristické metody.

5.3.1 Tradiční metody minimalizace

- Pomocí zákonů Booleovy algebry
  Jedná se o intuitivní metodu. Snažíme se opětovnou aplikací zákonů Booleovy algebry dosáhnout výsledku. Tato metoda není vhodná pro skupinovou minimalizaci logických funkcí ani pro minimalizaci neúplně zadaných logických funkcí.
- Pomocí Karnaughových map
  Intuitivní metoda vhodná pro minimalizaci logických funkcí o málo vstupech. Vhodná pro minimalizaci neúplně zadaných funkcí. Lze jí provést skupinovou minimalizaci logických funkcí. Metoda spočívá v zakreslení výsledků logické funkce do mapy velikosti 2^n polí, kde n je počet vstupních proměnných, a následném vyčítání co největších smyček. Metoda je velmi nepřehledná pro více než 4 vstupní proměnné, smyčky pak lze dělat i přes přímo nesousední políčka mapy čímž se ztrácí přehlednost.
- Pomocí Quine-McCluskey
  Metoda spočívá:
  1. v nalezení množiny přímých implikantů
  2. ve výběru dvojic implikantů lišících pouze negací jedné proměnné a jejich minimalizaci
  3. v opakování kroku dva na výsledek minimalizace dokud nezískáme množinu přímých implikantů
  4. v sestavení tabulky pokrytí a výběru minimálního množství přímých implikantů popisující ONset

5.3.2 Heuristické metody

Hlavní využití minimalizátorů založených na heuristice se projevuje na úlohách špatně řešitelných tradičními metodami minimalizace. Mezi tento typ minimalizátorů patří například Espresso, FC-Min a BOOM.

Pro tuto práci byl zvolen minimalizátor Espreso, jehož popis je v následující kapitole.

6 Minimalizátor Espreso

Jedná se o heuristický dvouúrovňový minimalizátor logických funkcí pracující s pla formátem souboru, který je detailně popsán v předešlé kapitole. Jde o jeden z nejúspěšnějších heuristických minimalizátorů vyvinutý ve spoluprací university v Berkeley s firmou IBM. U heuristických minimalizátorů se nesnažíme o nalezení nesporně nejlepšího řešení, ale nalezení optimálního řešení v rozumném časovém horizontu.

6.1 Princip činnosti

Espresso pracuje ve dvou cyklech a to vnitřní a vnější.

Vnitřní cyklus:

1. Redukce – redukuje pokrytí množiny ONset. Rozdělí množinu přímých implikantů ONset na množinu ne nutně přímých implikantů

2. Expand – nahradí každý implikant množiny ONset v získané množině přímým implikantem

3. Irredundant – z nalezené množiny hledá nejlepší pokrytí množiny ONsetu. Minimální počet přímých implikantů popisující množinu ONset

Vnitřní cyklus se provádí, dokud dochází k minimalizaci pokrytí množiny ONset. Poté se program dostane opět do vnějšího cyklu, který zkouší jiné pokrytí množiny ONset. Tento cyklus kontroluje, zdali jsou nová řešení lepší, nežli předešlá nalezená. Před voláním vnějšího cyklu je nutné získat alespoň nějaké pokrytí množiny ONset.
Detailní popis algoritmu lze nalézt v disertační práci „Multiple-Valued Logic Minimization for PLA Synthesi“. [9]

6.2 Úloha v projektu

Espresso je použité jako externí minimalizátor volaný na zjednodušený problém v každém uzlu stromu. Jelikož rychlost provádění minimalizace silně závisí na počtu termů a počtu vstupních proměnných, jedná se o pokus urychlit zpracování minimalizace dělením problému na menší části. Tento přístup také umožňuje paralelizaci řešení problému po sestavení stromu, protože jednotlivé vrstvy stromu jsou na sobě nezávislé.

6.3 Funkce –Dverify

Zápis Espreso.exe –Dverify originál.pla výsledek.pla

Tato funkce dokáže ověřit, zdali je možné minimalizací originálu dosáhnout zadaného výsledku. Tato funkce Espressa je velmi důležitá pro ověření správnosti výsledku.

7 Realizace

Zde je obsažen postup činnosti na projektu.

7.1 Seznámení se s projektem

Během zjišťování stavu softwaru bylo odhaleno, že neodpovídá popisu Pavla Černého. Nebere v úvahu množinu DCset a chybně s ní operuje. Ani nedosahuje srovnatelných výsledků úrovně minimalizace co do počtu termů i do rychlosti s Espressem.

Minimalizace probíhá tak, že je volán externí minimalizátor Espresso přes příkaz System(arg) a výsledky minimalizace uzlů jsou předávány přes soubory, což je značně neefektivní přístup. Původní implementace minimalizátoru neumožňuje předávání dat mezi projektem a externím minimalizátorem jinak, nežli přes soubory a neumožňuje volání knihovního externího minimalizátoru.

7.2 Práce na projektu

Nezbytně nutné bylo zjistit skutečný stav projektu na počátku, provést srovnávací benchmarky a zjistit co projekt doopravdy umí.

Byla oprava funkce projektu. Stávající verze již bere DCset při minimalizaci v potaz tudíž umožňuje větší úroveň minimalizace neúplně zadaných logických funkcí.

Ze zdrojových kódů Espressa v předešlé verzi projektu byla vytvořena minimalizovaná verze Espressa a ta použita jako dynamická knihovna (dll).

Další krok byla úprava kódu nacházející se v CUDD balíčku tak, aby umožňoval volání interního minimalizátoru.

Na závěr byly provedeny testy týkající se pořadí vstupních proměnných, jejich vlivu na kvalitu minimalizace a srovnávací testy s předešlou verzí projektu.

8 Testování

Obsahuje srovnávací testy Espressa, původní varianty projektu a nově získané verze.

8.1 Porovnání s předešlou verzí

Úroveň optimalizace se shoduje. Dle zadání práce byl ponechán původní algoritmus minimalizace upravený o rozeznávání DCset.

	stará verze projektu[ms]		nová verze projektu[ms]		
Soubor	externí volání	celkem	externí volání	celkem	zrychleni
100i-1o-100t-20dc-alg.pla	54075	65844	2148	12375	5,3
bb_50x5x50_20_9.pla	202001	212313	7253	19719	10,8
test2.pla	597599	625906	26935	55016	11,4
bb_250x5x200_20_9.pla	4177030	4380313	170509	372437	11,8
bb_50x5x300_20_9.pla	1379122	1450047	48716	117593	12,3
10i-1o-100t-30dc.pla	6843	7250	310	578	12,5
bb_150x5x150_20_9.pla	1907622	2000046	64673	156296	12,8
test3.pla	306111	321438	9670	24437	13,2
soar.pla	45601	48140	1781	3157	15,2
bb_100x5x50_20_9.pla	432294	454234	11292	27547	16,5
20i-1o-200t-20dc.pla	130330	136594	3117	8078	16,9
15i-1o-200t-25dc.pla	102597	107516	2700	6172	17,4
50i-1o-500t-10dc.pla	1146297	1190344	29105	64359	18,5
ibm.pla	48043	50297	1182	2422	20,8
ex4.pla	81138	84688	2047	4078	20,8
pdc.pla	54334	57187	1145	2703	21,2

Tabulka 8.1: Srovnávací testy původní a nové verze

8.2 Porovnání s Espressem

8.2.1 Jednovýstupové logické funkce

Pro jednovýstupové funkce je úroveň optimalizace srovnatelná s Espressem. Navržený minimalizátor ovšem dává přednost minimalizaci počtu literálů v termech na úkor počtu termů. Tedy pro jednovýstupové funkce úplně i neúplně zadané je tento algoritmus výhodný pro dosažení minimálního počtu literálů v termech.

Soubor	Espresso	stavající
10i-1o-100t-0dc-50odc.pla	39	41
16i-1o-100t-5dc-20odc.pla	78	78
16i-1o-100t-5dc-50odc.pla	51	51
20i-1o-200t-5dc-10odc.pla	177	177
20i-1o-200t-5dc-50odc.pla	107	107
20i-1o-300t-15dc-20odc.pla	253	254
30i-1o-300t-0dc-70odc.pla	101	101
40i-1o-200t-0dc-20odc.pla	161	161
40i-1o-300t-0dc-20odc.pla	242	242

Tabulka 8.2: Srovnání počtu termů neúplně určených logických funkcí s jedním výstupem

8.2.2 Skupinová minimalizace

Testy ukázaly, že minimalizace vícevýstupové funkce tímto algoritmem nemá využití. Počet termů vzniklých minimalizací byl v extrémních případech přibližně roven násobku počtu vstupních termů a počtu výstupních funkcí.

8.3 Variace vstupů

Dle testování nemá pořadí vstupních proměnných zásadní vliv na úroveň a minimalizace. U funkcí jedné proměnné je stávající volba pořadí prováděná CUDD balíčkem dostatečná, i když orientovaná na počet literálů v termech, nikoli na počet termů. Daná orientace minimalizace plyne z užití binárních stromů.

Pro skupinovou minimalizaci nelze určit výhodné pořadí vstupních proměnných. Stávající pořadí se snaží o dosažení minimálního stromu a tím minimalizuje čas potřebný pro zpracování úlohy. Minimální počet vnitřních uzlů ovšem nesouvisí s počtem společných termů. Pořadím vstupních proměnných nelze přímo ovlivňovat počet společných termů výstupních funkcí, čímž by se dosáhlo větší úrovně minimalizace pro více výstupních proměnných, avšak na úkor rychlosti zpracovávání.

9 Závěr

Cílem práce bylo urychlit předešlí projekt a tento cíl byl splněn. V zadané úloze bylo dosaženo zrychlení zpracovávání na testovaných úlohách přibližně 10 až 20 krát s dosažením shodných výsledků. Pro tyto účely byla vyrobena knihovna z aplikace Espresso. Knihovna Espressa byla použita jako externí minimalizátor volaný přímo knihovnou CUDD. Volání probíhá přes předání ukazatele na funkci tak, aby bylo co nejvíce univerzální.

Úroveň minimalizace modifikované aplikace je shodná s původním projektem, avšak byla opravena chyba práce s množinou DCset, která zapříčiňovala malou úroveň minimalizace funkcí obsahujících DCset.

Navržený algoritmus dosahuje vhodné úrovně minimalizace jednovýstupových funkcí zaměřené na redukci literálů výstupních termů, nikoli na jejich počet. Těchto výsledků dosahuje za delší čas v porovnání s Espressem. Espreso i přes své stáří prokazuje velmi dobré výsledky minimalizace za velmi krátký čas.

Stávající implementace algoritmu, jež byla ponechána, prokazuje nízkou efektivitu na vícevýstupové funkce. Pro dosažení lepšího výsledku minimalizace by bylo nutné znovu zpracovat teoreticky vnitřní pochody, zaměřit pozornost na hledání možných průniků množin ONset všech výstupních funkcí a vést minimalizaci tímto směrem. Daný postup by docílil snížení počtu termů, což se u skupinové minimalizace projevilo jako výhodnější, nežli snížení počtu literálů jednotlivých termů.

Zdrojové kódy, jejich komentáře, testovací úlohy a texty se nachází na přiloženém CD. Náhled struktury obsahu přiloženého CD je v příloze B.

Použitá literatura

[1] Černý P.: Skupinová minimalizace neúplně určených logických funkcí pomocí modifikovaných rozhodovacích diagramů. Diplomová práce, ČVUT FEL, 2008.

[2] Bílek J.: Minimalizace neúplně určených logických funkcí pomocí modifikovaných binárních rozhodovacích diagramů. Diplomová práce, ČVUT FEL, 2007.

[3] Kološ O.: Port balíku CUDD pod Windows. Bakalářská práce, ČVUT FEL, 2006.

[4] Lee C. Y.: Representation of Switching Circuits by Binary-Decision Programs. 1959

[5] Akers S. B.: Binary Decision Diagrams. 1978

[6] Bryant R.E.: Graph-based algorithms for boolean functions manipulation. 1986.

[7] Bryant R. E.: Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. 1992

[8] Somenzi F.: http://vlsi.colorado.edu/~fabio/CUDD/

[9] Rudell R.L.: Multiple-Valued Minimization for PLA Optimization. Dissertation thesis, University of California at Berkeley, 1987

[10] University of California at Berkeley: http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/ 1993-1994

[11] Schindler J.: Detekce tautologie. Diplomová práce, ČVUT FEL, 2009.

Příloha A

Detailní popis souboru pla

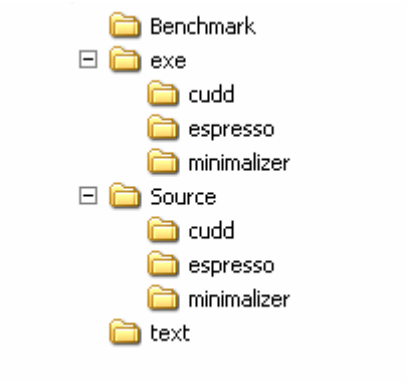
Povinné položky jsou tučně označené
.i argument argument určuje počet vstupních proměnných
.o argument argument určuje počet výstupních proměnných
.p argument argument určuje počet termů
.ilb s1 s2 .. sn pojmenování vstupních proměnných
.ob s1 s2 .. sn pojmenování výstupních proměnných
.type arg může být jeden z následujících
- f – následuje tabulka popisující množinu ONset, množina DCset je prázdná
- r – následuje tabulka popisující množinu OFFset, množina DCset je prázdná
- fd – následuje tabulka množiny ONset a DCset
- fr – následuje tabulka množiny ONset a OFFset
- dr – následuje tabulka množiny DCset a OFFset
- fdr – následuje tabulka množiny ONset, OFFset i DCset (jedná se o úplný výčet), pokud není vyplněn odpovídá fd

tabulka termů například -01 010 říká, že pro vstupy ¬x2*x3 jsou výstupy 010
.e ukončení souboru pla

znak – znamená, že proměnná na této pozici hodnotu termu neovlivní, u výstupů je tímto symbolem definován prvek DCset
znak ~ říká, že na hodnotě daného výstupu nezáleží

Příloha B

Obsah přiloženého CD



B.1 Struktura obsahu CD

Složka Benchmark obsahuje testovací příklady
Složka exe obsahuje přeložené .lib, .dll a .exe projektu
Složka Source obsahuje zdrojové kódy použitých částí projektu
Složka text obsahuje dokument této práce ve formátu doc a pdf