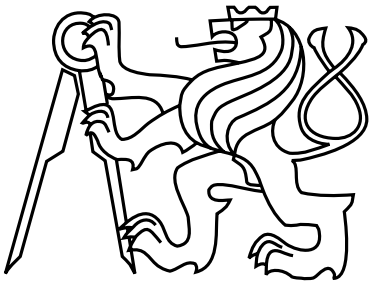
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Diplomová práce

Ortogonalizace logických funkcí

Bc. Lukáš Höger

Vedoucí práce: Ing. Petr Fišer, Ph.D.

Studijní program: Elektrotechnika a informatika, strukturovaný, Navazující magisterský

Obor: Výpočetní technika

5. ledna 2010

Poděkování

Rád bych na tomto místě poděkoval Ing. Petru Fišerovi, Ph.D. za vedení diplomové práce, za jeho cenné rady, podklady a připomínky.

Dále děkuji mé rodině za poskytnuté zázemí při psaní této práce. Zejména patří dík otci Vítězslavovi za výpomoc při stylizaci textu a mé matce Věře za gramatickou korekturu textu. Chtěl bych také poděkovat své přítelkyni Janě Steblové za psychickou podporu a bratru Tomášovi za technické konzultace.

Děkuji také svému Bohu za to, že mohu studovat technickou školu a že se mohu učit trpělivosti a systematičnosti při každé nové práci.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod 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 4. 1. 2010 .....

Abstract

This thesis pursues orthogonalization of a logic function - decomposition of logic function to disjoint sum of products. The thesis proposes implementation of a algorithm for orthogonalization with optimalization of results. A product of this work is a tool which is able to load logic function from PLA file, minimize it and save it back. Implementation of the algorithm is compared on MCNC benchmarks with DSOP and a subprogram of Espresso for orthogonalization.

Abstrakt

Tato diplomová práce se zabývá ortogonalizací logické funkce - rozklad logické funkce na součet vzájemně nepřekrývajících se součinů termů. Práce předkládá návrh vhodného algoritmu pro ortogonalizaci s optimalizaci výsledného řešení. Výsledkem je nástroj, který dokáže načíst logickou funkci z PLA souboru, ortogonalizovat ji a uložit zpět. Implementace algoritmu je porovnána na MCNC benchmarcích s cizími dostupnými nástroji DSOP a Espresso s podprogramem pro ortogonalizaci.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

1.1 Úvod do problematiky

Elektronická digitální zařízení v dnešní době hluboce pronikly do života každého z nás. Můžeme je najít v mnoha oblastech, od dětských hraček až po automobily. S rychlým tempem rozvoje elektroniky přichází potřeba její efektivní implementace. Návrh logických obvodů se stal klíčovým faktorem ekonomické úspěšnosti převážné většiny současných průmyslových výrobků.

Oblast logických funkcí přirozeně zavádí početní algebru v dvouhodnotovém světě jedniček a nul, která modeluje vlastnosti množinových a logických operací. Proces ortogonalizace, kterým se v této práci budeme zabývat, je algebraický způsob převodu zadané logické funkce do tvaru, kdy se jednotlivé součty součinů vstupních proměnných vzájemně neprotínají. Úkolem této práce je nejen nalézt způsob transformace logické funkce do ortogonalizované podoby, ale také optimalizace výsledku hledáním co nejúspornějšího vyjádření řešení.

S ortogonalizaci logických funkcí se dnes setkáváme v mnoha oblastech moderních návrhů logických obvodů. Používá se jako startovní bod pro syntézu ESOPs [1] nebo pro výpočet spektra logických funkcí [2].

1.2 Cíle práce

- Seznámení se stávajícími (konkurenčními) nástroji pro ortogonalizaci logické funkce.
- Návrh a implementace algoritmu pro ortogonalizaci.
- Otestování vytvořeného řešení na zkušebních úlohách.
- Srovnání s konkurenčními nástroji.

1.3 Implementační požadavky na řešení

- Implementace algoritmu v programovacím jazyce C++.
- Realizace nástroje postavena na programu BOOM (BOOlean Minimizer).

1.4 Rozdělení textu na kapitoly

Diplomová práce je rozdělena do osmi kapitol. První kapitola popisuje úvod do problematiky a shrnuje cíle práce. V druhé kapitole jsou zavedeny pojmy a definice z oblastí logických obvodů. V třetí kapitole je přiblížen proces ortogonalizace logické funkce. Ve čtvrté kapitole jsou nastíněny cizí dostupné nástroje na ortogonalizaci. V páté kapitole je analyzován problém ortogonalizace logické funkce s návrhem co nejoptimálnějšího vlastního řešení. V šesté kapitole je popsána realizace samotného řešení. Sedmá kapitola se zabývá testováním jednotlivých verzí navrženého algoritmu a srovnání s konkurenčními implementacemi. Osmá kapitola zhodnocuje splnění cílů práce, shrnuje výsledky testování a popisuje možné budoucí rozšíření a vylepšení navrženého algoritmu.

Kapitola 2

Definice

2.1 Opakování pojmů

V této sekci budou nastíněny základní termíny a definice z oblastí booleovy algebry a logických funkcí použitých v textu této práce. Záměrem této části je připomenout tyto pojmy, ne však dostačujícím způsobem vysvětlit.

2.1.1 Booleova algebra

Logické funkce využívající dvouhodnotovou logiku se označují jako Booleovy funkce. Matematickým nástrojem na práci s Booleovými funkcemi je Booleova algebra, která je definována jako distributivní a komplementární svaz skládající se z:

- Konečné množiny logických proměnných (a, b, c, d, e, ...z).
- Binárních operací logického součtu (označovaného $+$ nebo $\vee$) a logického součinu (. nebo $\wedge$).
- Unární operace negace ($^-$ nebo $\neg$).
- Dvou nulárních operací v podobě logických konstant 0 a 1.

Pro Booleovou algebru platí tyto základní zákony:

- 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: $x + (\overline{x} * y) = x + y$, $x * (\overline{x} + y) = x * y$
- Agresivita nuly a jedničky: $x * 0 = 0$, $x + 1 = 1$

- Komplementarita: $x + \overline{x} = 1$, $x * \overline{x} = 0$
- Dvojitá negace: $\overline{(\overline{x})} = x$
- De Morganovy zákony: $\overline{x} * \overline{y} = \overline{(x + y)}$, $\overline{x} + \overline{y} = \overline{(x * y)}$

2.1.2 Vyjádření logických funkcí

Logická (Booleova) funkce n proměnných je definována jako $f : B^n \rightarrow B$, je-li B Booleova algebra. Logickou funkci lze zapsat několika způsoby:

- **Pravdivostní tabulka** Tabulka obsahující všechny možné kombinace vstupních proměnných a pro ně definovaných výstupů.
- **Logický výraz** Vyjádření logické funkce v algebraické formě.
- **Mapa** Karnaughova mapa využívající se pro minimalizaci logické funkce s malým počtem proměnných.
- **Vícerozměrné jednotkové krychle** Popis logické funkce pomocí vícerozměrné krychle, kde souřadnice jsou dány hodnotami vstupních proměnných.
- **Modifikované binární diagramy** Reprezentace logické funkce vhodná pro strojovou minimalizaci.

2.1.3 Používané důležité termíny

Literál Proměnná nebo její negace. Literál může nabývat pouze dvou hodnot "0" nebo "1".

Term Logický výraz složený z literálů mezi sebou navzájem vázaných logickým operátorem. Termy dělíme na součinový term = krychle (neobsahuje operátory součtu) a součtový term (neobsahuje operátory součinu). **Upozornění**: V textu této práce bude pod pojmem term myšlen term součinový nebude-li uvedeno jinak!

ONset Množina termů logické funkce, jejichž výstupem je logická "1".

OFFset Množina termů logické funkce, jejichž výstupem je logická "0".

DCset (Don't care set) Množina termů logické funkce, jejichž výstup není určen.

Implikant logické funkce Součinový term, který definuje logickou funkci. Každý term z množiny ONset je implikantem logické funkce.

Přímý implikant Implikant logické funkce, který už není možné více zjednodušit.

Minterm Součinový term, který obsahuje všechny vstupní proměnné (mohou být v přímém nebo inverzním tvaru)

Maxterm Součtový term, který obsahje všechny vstupní proměnné (mohou být v přímém nebo inverzním tvaru)

Součet termů (SOP) Množina termů zahrnující všechny jedničky "1".

Disjunktivní normální forma (DNF) Logický výraz ve tvaru součtu součinových termů, který má funkční hodnotu logickou "1".

Konjunktivní normální forma (CNF) Logický výraz ve tvaru součinu součtových termů, který má funkční hodnotu logickou "0".

Pokrytí logické funkce Množina termů reprezentující danou logickou funkci (nejmenší počet termů).

Booleova minimalizace Dvouúrovňová minimalizace, která hledá pokrytí o minimální velikosti (nejmenší počet termů).

2.2 Operace pro dekompozici termů

Úloha ortogonalizace logické funkce je postavena na postupném rozkladu termů. V této sekci si uvedeme dvě operace pro dekompozici. Jsou to Sharp a Disjoint sharp.

Operace Sharp

Operaci Sharp označíme #; α a β jsou implikanty logické funkce.
Pro $\alpha\#\beta$ vrací operace Sharp součet implikantů, kdy α bude zachován a β pokrývají implikanty, které nesmí pokrýt žádnou "1" z α.

Příklad použití operace Sharp:

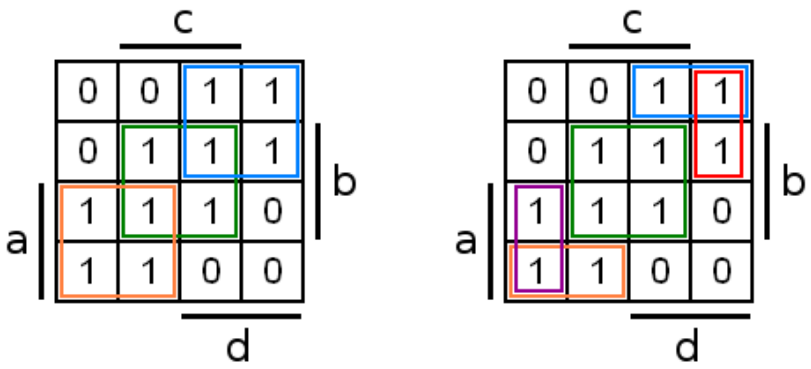
$\alpha = bc$
$\beta = a\overline{d} + \overline{a}d$
$\alpha\#\beta = bc + \overline{b}a\overline{d} + \overline{c}a\overline{d} + \overline{a}d\overline{b} + \overline{a}d\overline{c}$

Zápis příkladu použití operace Sharp je v Karnaughové mapě na obrázku 2.1.

Operace Disjoint sharp

Operace Disjoint sharp vychází z operace Sharp. Operaci Disjoint sharp označíme $\odot$.
Pro $\alpha \odot \beta$ vrací operace Disjoint sharp součet implikantů, kdy α bude zachován a β pokrývají implikanty, které nesmí pokrýt žádnou "1" z α. Zároveň všechny výsledné implikanty jsou vzájemně disjunktní (vzájemně se neprotínají).

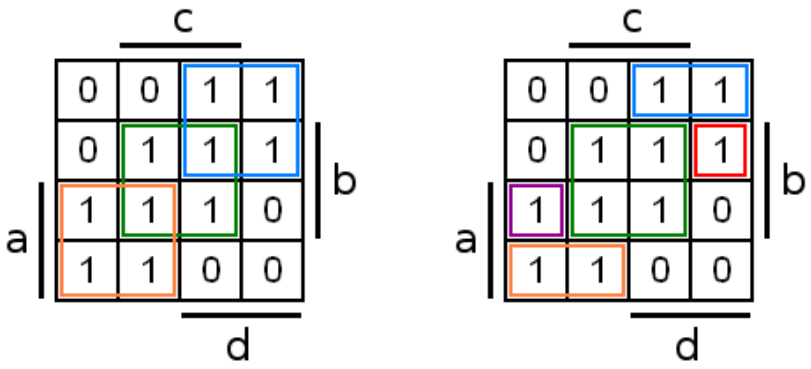
Příklad použití operace Disjoint sharp:



Obrázek 2.1: Příklad použití operace Sharp

$\alpha = bc$
$\beta = a\overline{d} + \overline{a}d$
$\alpha \bigodot \beta = bc + \overline{b}ac\overline{d} + \overline{c}a\overline{d} + \overline{a}d\overline{b} + \overline{a}bd\overline{c}$

Zápis příkladu použití operace Disjoint sharp je v Karnaughové mapě na obrázku 2.2.



Obrázek 2.2: Příklad použití operace Disjoint sharp

2.3 Formát PLA

Formát PLA je navržen University of California Berkeley v USA. Používá se pro načítání a ukládání logické funkce. PLA formát je způsob zápisu reprezentace dvouúrovňové logické funkce. V této části budou popsány jeho nejdůležitější vlastnosti, více informací je možné najít v [16].

PLA formát se zapisuje v textovém souboru s příponou *.pla*. Soubor se načítá po jednotlivých řádcích a každý řádek udává parametr nebo vlastnost logické funkce.

Základní nejdůležitější parametry jsou:

.i [číslo] Parametr udává počet vstupních proměnných.
.o [číslo] Parametr udává počet výstupů logické funkce.
.p [číslo] Parametr udává počet termů.
.type [typ] Typ způsobu definice logické funkce.
e. Příznak označující konec definice logické funkce.

Logická funkce je definována pomocí seznamu termů (jeden term = jeden řádek). Každý term je v součinovém tvaru a má podobu:

1100-1-1 11-0

Levá část po mezeru určuje vstupní proměnné funkce a pravá část za mezerou odpovídá výstupním hodnotám, které definují, zda je term na levé straně pravdivý. Znak "-" označuje neurčený vstup / výstup. Ve vstupní části označuje znak "1" literál a znak "0" označuje negovaný literál. Ve výstupní části znak "1" říká, že term patří do ONsetu pro daný výstup a znak "0" říká, že term patří do OFFsetu daného výstupu.

Příklad PLA souboru je na obrázku 2.3.

2.4 Nástroj BOOM

Program BOOM (BOOlean Minimizer) byl použit jako základ při vytváření algoritmu ortogonalizace logické funkce. BOOM je heuristický nástroj pro minimalizaci (booleovských) logických funkcí, vyvíjený katedrou výpočetní techniky fakulty elektrotechnické ČVUT v Praze. Běží jako Win32 console aplikace a obsahuje škálu funkcí a operací pro práci z logickými funkcemi. Je to např. již zmiňovaná minimalizace logické funkce, přidávání a odebírání termů, informativní funkce vracející vlastnosti termů, funkce porovnávající termy aj.

BOOM podporuje formát PLA. Na programu se stále pracuje a vylepšují se jeho schopnosti, nejnovější verze je BOOM 2.7.

```
.i 5
.o 3
00000  000
0--11  001
00101  010
00110  011
010-1  001
-1010  010
01100  011
01110  100
1-000  010
10011  011
10100  100
10111  101
110-1  011
11011  100
11101  101
11-11  110
1--11  100
111-1  001
11-10  111
e.
```

Obrázek 2.3: Příklad výpisu souboru s logickou funkcí v PLA formátu.

Kapitola 3

Popis problému ortogonalizace

V následujích řádcích této kapitoly je popsána činnost ortogonalizace logických funkcí a úskalí, která jsou s ní spojené.

3.1 Ortogonalizace

V lineární algebře se ortogonalizací označuje postup hledání souboru ortogonálních (vzájemně kolmých) vektorů, které pokrývají vymezený podprostor [14]. Formálně popsáno, ortogonalizace je proces nalezení v lineárně nezávislém souboru vektorů $(v_1, v_2, ..., v_k)$ soubor vektorů $(u_1, u_2, ..., u_k)$, který generuje stejný podprostor jako vektory $(v_1, v_2, ..., v_k)$. Přitom každý vektor z nového souboru je kolmý ke všem ostatním vektorům v novém souboru.

Jednoduše řečeno, ortogonalizace je proces rozdělení problému nebo systému na jednotlivé odlišné prvky, které po složení dají stejný výsledek.

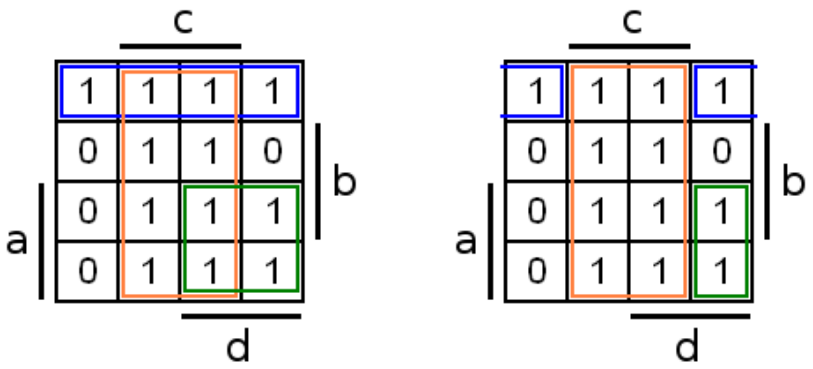
3.2 Ortogonalizace logické funkce

Ortogonalizací je v tomto projektu myšlen rozklad logické funkce na součet vzájemně nepřekrývajících se součinů termů. Tento tvar logické funkce se označuje DSOP (Disjoint Sum of Product).

Názorná ukázka ortogonalizace logické funkce je na obrázku 3.1. Je patrné, že výsledek ortogonalizace má úzkou spojitost s operaci $\odot$ (viz. kap 2.2). Postupný rozklad termů pomocí této operace zaručí ortogonalitu logické funkce.

Pro ortogonalizovanou logickou funkci $f2$, která vznikla rozkladem logické funkce $f1$, musí platit tyto dvě vlastnosti:

- Každý term z $f2$ musí být disjunktní (nesmí se protínat) s ostatními termy z $f2$.
- $f1$ a $f2$ musí být vzájemně ekvivalentní logické funkce, tj. termy z $f1$ musí pokrýt všechny termy z $f2$ a naopak.



Obrázek 3.1: Logická funkce a její ortogonalizovaný tvar zapsaný v Karnaughové mapě.

Elementární rozklad, který zaručí vždy ortogonalitu logické funkce, je dekompozice na jednotlivé mintermy.

3.3 Rozložení dvou termů

Pro dva termy t_1 a t_2, které se vzájemně překrývají (příklad na obrázku 3.2), existují vždy dvě efektivní možnosti rozkladu. Zachová se buď term t_1 (zůstane nezměněn) a podle něho se rozdělí term t_2 na součet podtermů ($t_1 \odot t_2$; levá Karnaughová mapa na obrázku 3.3).

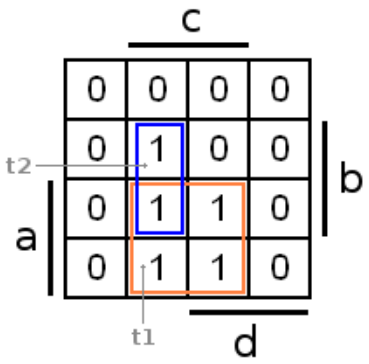
V druhém případě naopak term t_2 zůstane nezměněn a rozdělí term t_1 na součet podtermů ($t_2 \odot t_1$; pravá Karnaughová mapa na obrázku 3.3).

Různými možnostmi dekompozice logické funkce můžeme ve výsledku dosáhnout odlišného počtu termů. Přichází tím první důležité zamyšlení. Jakým způsobem provést rozklad aby počet termů byl co nejmenší a tím bylo dosáženo co nejlepšího možného výsledku? Toto můžeme označit jako stěžejní otázku problému ortogonalizace, na kterou se snažím v této práci najít odpověď.

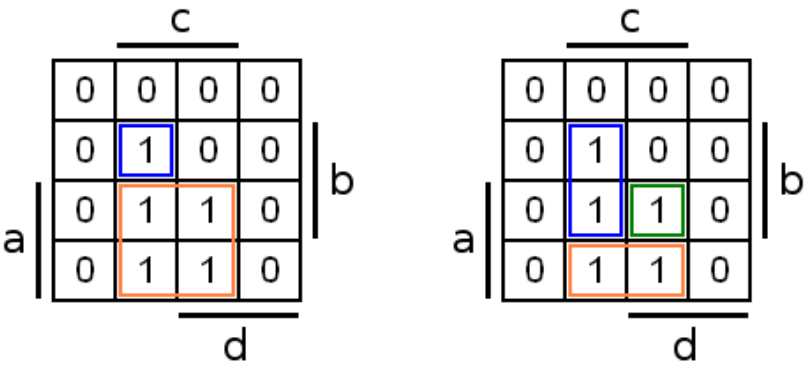
3.4 Časová složitost algoritmu

Cena za algoritmus, který hledá nejlepší výsledek, je obvykle penalizován časem, tzv. je časově náročnější. Algoritmus, který dokáže rozložit logickou funkci na co nejmenší počet vzájemně disjunktních termů, vede k řešení postavit časově složitější programovou konstrukci.

A jak moc nás zajímá v úloze tohoto projektu časová složitost? Z využití ortogonalizace logických funkcí v praxi můžeme usoudit, že jediným kritériem pro časovou náročnost je vytvoření algoritmu, který bude schopný dokončit úlohy v akceptovatelném čase.



Obrázek 3.2: Logická funkce zapsaná v Karnaughové mapě.



Obrázek 3.3: Ortogonalizovaná logická funkce. Ukázka dvou způsobů rozkladu.

3.4.1 Optimální řešení

Nalezení optimálního řešení ortogonalizace logické funkce lze převést na úlohu minimálního pokrytí. Algoritmus minimálního pokrytí je NP těžká úloha [8]. Z časové složitosti algoritmu vyplývá, že optimální řešení ortogonalizace logické funkce nelze vždy najít v polynomiálním čase.

Pro úlohu, která je zadaná logickou funkcí o stovkách až tisících termech, je ale důležité, aby její řešení bylo nalezeno v polynomiálním čase, jinak by doba čekání na výsledek byla neúnosná. Proto v této práci je vytvářen algortimus, který pracuje v polynomiálním čase, ale s ryzikem, že nenajde vždy optimální řešení.

Kapitola 4

Seznámení s existujícími implementacemi

V této kapitole si přiblížíme existující implementace ortogonalizace logických funkcí. Programů, které řeší tento problém, není mnoho. Z dostupných, které jsem měl možnost vyzkoušet, to byly tyto tři: Espresso, DSOP, DSOP využívající maximální binate proměnné.

V následujících podkapitolách se s těmito nástroji podrobněji seznámíme. Testování a srovnávání jednotlivých nástrojů se nachází v kapitole 7.

4.1 Espresso

Program Espresso [5] je heuristický nástroj pro dvouúrovňovou minimalizaci logických funkcí. Autorem základního algoritmu je Richard Rudell působící na University of California Berkeley. Na jeho základě byl vyvinut samotný nástroj Espresso, který byl navržen Robert Braytonem z IBM. Jedná se o volně kopírovatelný program, který je v oblasti minimalizace široce rozšířen.

Espresso je implementováno v K & R C [10], ale existuje také i v mnoha jiných podobách, včetně knihovny do C++. Podporovaný formát pro načtení a uložení logické funkce je PLA.

Přestože hlavní přínos Espressa je v oblasti optimalizace heuristické minimalizace, nástroj obsahuje také i podprogram na rozklad termů na disjunktní části. Spuštění programu se provádí pomocí parametru -Ddisjoint.

Rozložení termů je založeno na operaci Disjoint sharp. Samotný proces ortogonalizace je implementován velice jednoduše, bez žádných složitějších sofistikovaných metod.

4.2 DSOP

Nejsilnější dostupný nástroj na ortogonalizaci logických funkcí, s kterým jsem měl možnost se seznámit, je DSOP [11] vzniklý na Katedře informačních technologií na universitě v Pise v Itálii. Jejími autory jsou A. Bernasconi, V. Ciriani, F. Luccio, L. Pagli.

DSOP představuje implementaci heuristického rozhodování pro nalezení disjunktního součtu součinů termů logické funkce. Nástroj je implementován v programovacím jazyku

Java a podporuje PLA formát.

Algoritmus pracuje ve čtyřech fázích:

- V první fázi se spustí Espresso pro minimalizaci zadané logické funkce.
- V druhé fázi se přiřadí termům váha w, která je závislá na počtu protnutí s ostatními termy.
- V třetí fázi se setřídí pořadí termů podle váhy w.
- Ve čtvrté fázi se provede rozložení termů na disjunktní celky pomocí metody postavené na operaci Disjoint sharp.

Složitost celého algoritmu je polynominální ve velikosti výstupu, tj. počtem termů vzniklých při rozkladu.

DSOP je vytvořena ve třech dostupných verzích:

- DSOP-1 Jednoduchá nejrychlejší verze, založený na základním algoritmu bez průběžného uspořádání termů během procesu rozkladu.
- DSOP-2 Zdokonalený algoritmus rozšířen o průběžné uspořádání termů během procesu rozkladu.
- DSOP-3 Algoritmus DSOP-2 s vylepšeným výběrem termů pro rozklad.

Pro každou variantu algoritmu je možnost volby mezi dvěmi typy uspořádání termů podle váhy a to vzestupně nebo sestupně.

4.3 DSOP využívající maximální binate proměnné

Algortimus ortogonalizace logické funkce, který je postaven na heuristice nalezení největší binate proměné [6]. Jeho autoři jsou L. Shivakumaraiah and A. Thornton z Department of Electrical and Computer Engineering v Mississippi State University.

Algoritmus v prvním kroku hledá proměnnou (tzv. největší binate proměnnou), která má nejčastější výskyt ve všech termech logické funkce v přímé nebo negované podobě. Termy jsou rozděleny podle této největší binate proměnné na dvě kategorie. Na termy, které ji obsahují v přímé podobě (A) a termy , které ji obsahují v negované podobě (B). Termy, které mají na pozici binate proměnné don't care, se přepíšou v dané proměnné na "1" a uloží se do A a zároveň na "0" a uloží se do B. Algoritmus se opakuje tak dlouho, dokud se postupně neprojdou všechny proměnné dané logické funkce.

Nástroj, postavený na popisovaném algoritmu, využívá pro načtení a uložení logické funkce formát PLA.

Program DSOP využívající maximální binate proměnné není volně dostupný k použití. Měl

jsem pouze možnost se s ním seznámit teoreticky a to prostřednictvím informací, které lze nalézt na internetu [6]. Jsou tam k vidění i srovnávací testy s jinými algoritmy. Podle mého názoru, ale nejsou použitelné, protože počet úloh, na kterém byla tato implementace testovaná, je velice malý. Je jich pouze 7. Z tohoto důvodu algoritmus DSOP využívající maximální binate proměnné nebude začleněn do srovnávacích testů této diplomové práce.

Kapitola 5

Analýza problému a návrh implementace

Záměrem této kapitoly je analyzovat problém ortogonalizace a návrh co nejoptimálnějšího vlastního řešení.

5.1 Rozklad termů

Při dekompozici, jak už u procesu ortogonalizace víme, musíme rozdělit logickou funkci na nezávislé nepřekrvající se termy. Pro tento účel nám poslouží již popisovaná operace $\odot$.

5.1.1 Postupný rozklad

Abychom rozkladem získali vzájemně disjunktní množinu termů, musíme rozložit všechny překrývající se termy. Základní postup pro rozklad s použitím $\odot$ vypadá následovně: Pro logickou funkci F, která se skládá z množiny termů $T = \{t_1, t_2, t_3, t_4, ...t_n\}$, provedeme pro každý term z T operaci $\odot$ ze všemi ostatními termy z T.

5.2 Řízení rozkladu

Už víme, že hledáme způsob, kterým rozložíme zadanou logickou funkcí na co nejmenší počet termů. Průběh procesu dekompozice je závislý na významném parametru a tím je pořadí, ve kterém se budou termy postupně rozkládat. Jedna z nejvyužitelnějších metod, které se nabízí, jak zajistit vhodné pořadí pro rozklad, je setřízení termů.

Je důležité termy řadit průběžně. Při běhu operace $\odot$ vznikají nové termy, které jsou různých vlastností a je nezbytné je porovnat a začlenit do již uspořádané sekvence termů.

Pro úlohu ortogonalizace logické funkce lze použít tyto řadící metody:

- Seřazení podle velikosti termů
- Seřazení termů podle počtu protnutí s ostatními termy

- Seřazení termů podle možnosti sloučení (viz. kap 5.3)

Popis jednotlivých setřízení se nachází v částech 5.2.1, 5.2.2 a 5.3.3.

Každou metodu řazení můžeme navíc realizovat dvěma způsoby a to vzestupně nebo sestupně. Třídící metody lze také kombinovat - umístit více typů uspořádání za sebou (viz. kap 5.2.3).

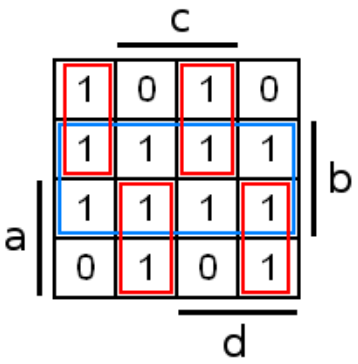
Jednotlivé zmiňované varianty řazení a jejich kombinace jsou v určitých případech užitečné a mohou dopomoci k dosážení optimálnějšího výsledku ortogonalizace.

5.2.1 Seřazení podle velikosti termů

Kritériem řazení je v této metodě velikost termů (počtu pokrytých "1"). Výhody použití sestupného a vzestupného řazení jsou ukázány na dvou příkladech.

Příklad 5.1 Logická funkce F_1 je zadaná Karnaughovou mapou (obrázek 5.1). Výsledky rozkladu s použitím řazení podle velikosti termů jsou na obrázku 5.2.

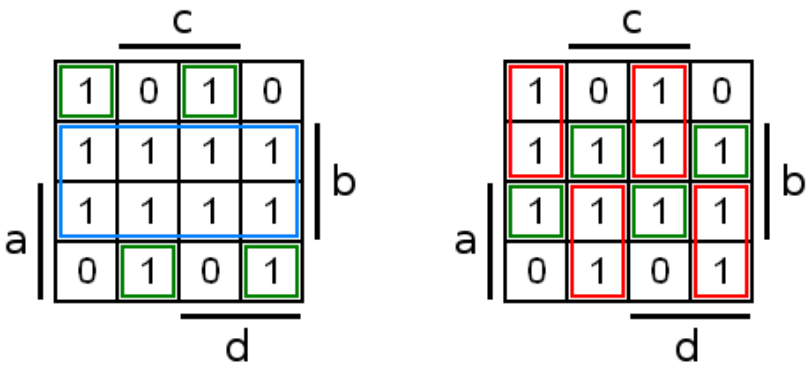
Příklad 5.2 Logické funkce F_2 je zadaná Karnaughovou mapou (obrázek 5.3). Výsledky rozkladu s použitím setřízení podle velikosti termů jsou znázorněny na obrázku 5.4.



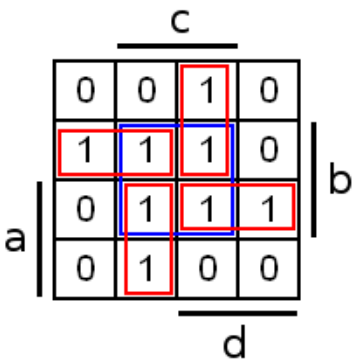
Obrázek 5.1: Logická funkce F_1 zapsaná v Karnaughové mapě.

V prvním příkladu 5.1 s logickou funkcí F_1 při rozkladu s metodou vzestupného třizení (levá Karnaughová mapa na obrázku 5.2) malé červené termy nevhodně rozdělí velký term na izolované malé kousky termů. Na výsledku se to projeví téměř v dvojnásobném počtu termů oproti rozkladu s metodou sestupného řazení (pravá Karnaughová mapa na obrázku 5.2).

V druhém příkladu 5.2 s logickou funkcí F_2, která není minimalizovaná, je výhodnější rozklad s metodou vzestupného uspořádání. Červené malé termy zde plně pokrývají modrý term a tím ho zcela nahradí (obrázek. 5.4).



Obrázek 5.2: Rozklad F_1 s použitím metod řazení podle velikosti termů sestupně a vzestupně.



Obrázek 5.3: Logická funkce F_2 zapsaná v Karnaughové mapě.

5.2.2 Seřazení termů podle počtu protnutí s ostatními termy

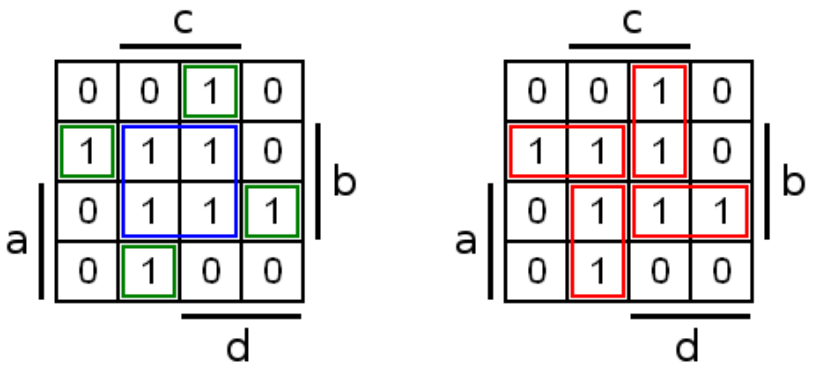
Kritériem setřízení je v této metodě počet protnutí s ostatními termy. Výhody použití sestupného a vzestupného řazení jsou ukázány na dvou příkladech.

Příklad 5.3 Logická funkce F_3 zadaná Karnaughovou mapou je (obrázek 5.5). Výsledky rozkladu s použitím seřazení podle počtu protnutí s ostatními termy jsou na obrázku 5.6.

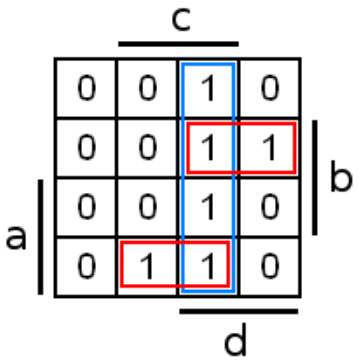
Příklad 5.4 Logická funkce F_4 je zadaná Karnaughovou mapou (obrázek 5.7). Výsledky rozkladu s použitím setřízení podle počtu protnutí s ostatními termy jsou na obrázku 5.8.

První příklad 5.3 s logickou funkcí F_3 poukazuje na výhodu použití sestupného řazení (pravá Karnaughova mapa na obrázku 5.6). Nejvíce protínaný modrý term s dominantní velikostí zde vhodně rozdělí ostatní menší termy.

Ve druhém příkladu 5.4 u logické funkce F_4 se třemi stejně velikými termy vychází naopak výhodněji použití vzestupného uspořádání (levá Karnaughova mapa na obrázku 5.8), kdy nejprotínanější modrý term je rozdělen zbylými dvěmi termy.



Obrázek 5.4: Rozklad F_2 s použitím metod řazení podle velikosti termů sestupně a vzestupně.



Obrázek 5.5: Logická funkce F_3 zapsaná v Karnaughové mapě.

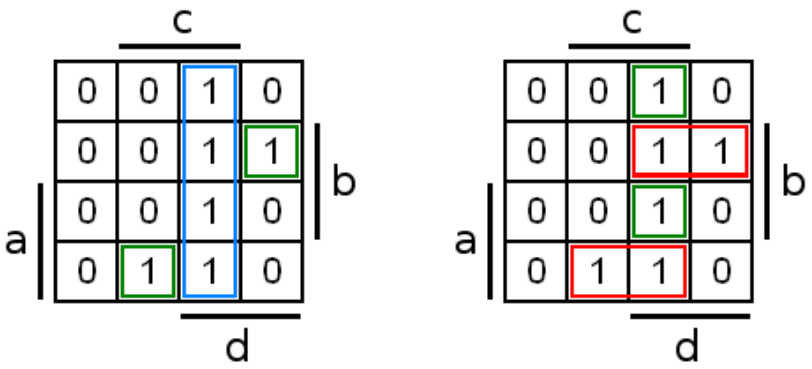
5.2.3 Sériové použití více třídících metod

Odlišné řadící metody můžeme umístit sériově za sebou a tím dosáhnout dvojitého uspořádání. Můžeme tak v určitých situacích využít částečně výhody obou řazení. Použití si vysvětlíme na následujícím příkladu.

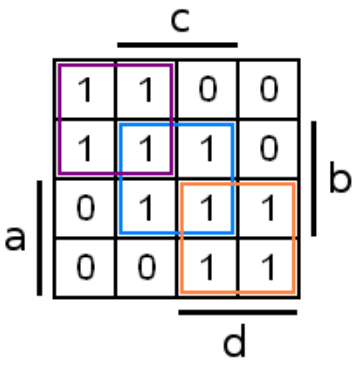
Příklad 5.5 Logická funkce f_5 je zadaná pravdivostní tabulkou a Karnaughovou mapou na obrázku 5.9. Termy nejdříve uspořádáme sestupným řazením podle počtu protnutí a následně setřídíme sestupným řazením podle velikosti termů.

Uspořádání termů v prvním kroku rozkladu je znázorněno na obrázku 5.10. Sériové použití těchto dvou řazení s porovnáním s třízením podle počtu protnutí s ostatními termy lze vidět po rozkladu na obrázku 5.11.

Dominantní vliv na výsledek uspořádání má druhé řazení. První setřízení navíc uspořádá, ale ještě i ty termy, které mají stejnou velikost.



Obrázek 5.6: Rozklad F_3 s použitím metod řazení podle velikosti termů sestupně a vzestupně.



Obrázek 5.7: Logická funkce F_4 zapsaná v Karnaughové mapě.

5.3 Složení termů

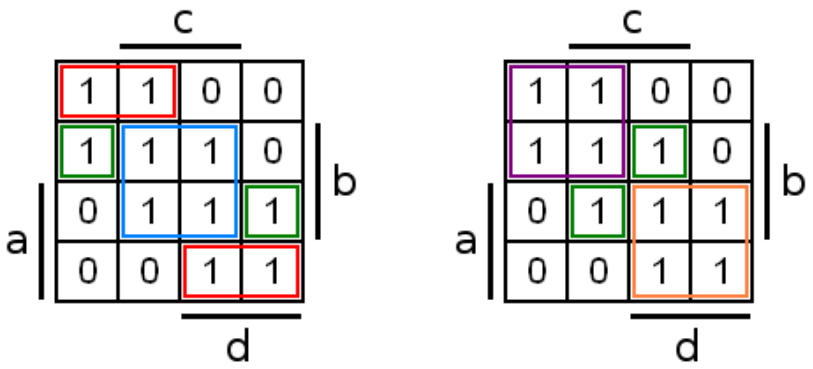
5.3.1 Pohlcení termů

Zadaná logická funkce nemusí být vždy v minimalizované podobě. Termy se mohou zcela překrývat a tím být nadbytečné v procesu ortogonalizace. Jejich eliminaci nazveme pohlcení termů neboli absorpci.

Formálně zapsáno: t_1, t_2 patří do množiny termů logické funkce F. Pohlcení termu t_1 nastane, když term t_2 plně pokrývá term t_1. Příklad pohlcení nadbytečných termů logické funkce je znázorněn na obrázku 5.12.

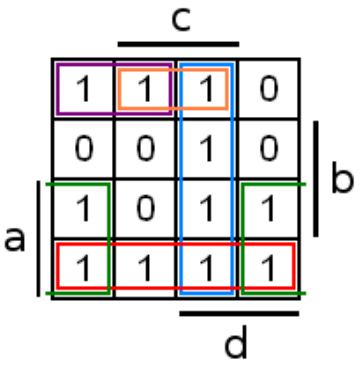
5.3.2 Sloučení termů

Cíl procesu ortogonalizace není pouze rozložit logickou funkci na disjunktní termy, ale také pokusit se získat minimální počet těchto termů. Zavedeme si novou funkci, která pomáhá tento požadavek splnit, a tou je sloučení termů.



Obrázek 5.8: Rozklad F_4 s použitím metod řazení podle počtu protnutí s ostatními termy sestupně a vzestupně.

	a	b	c	d
t1	1	–	0	–
t2	0	0	1	–
t3	0	0	–	0
t4	–	–	1	1
t5	1	0	–	–



Obrázek 5.9: Logická funkce f_5 zadaná pravdivostní tabulkou a Karnaughovou mapou.

Podmínky, které musí současně platit, pro sloučení dvou termů t_1 a t_2 jsou:

1. Stejná velikost termů t_1 a t_2.
2. Termy t_1 a t_2 se liší v polaritě jedné proměnné.

Možnost sloučení termů nastává ve dvou případech když:

1. Procesem ortogonalizace vznikají nové termy, které lze spojit s ostatními termy.
2. Zadaná logická funkce není minimalizovaná.

Příklad sloučení dvou termů po procesu ortogonalizace je znázorněno na obrázku 5.13. Modré termy rozdělí červené termy na mintermy a ty se poté spojí.

Term	t1	t2	t3	t4	t5
Velikost termu	4	2	2	4	4
Počet protnutí	1	2	1	2	2

⇓ sestupné řazení podle počtu protnutí s ostatními termy

Term	t2	t4	t5	t1	t3
Velikost termu	2	4	4	4	2
Počet protnutí	2	2	2	1	1

⇓ sestupné řazení podle velikosti termů

Term	t4	t5	t1	t2	t3
Velikost termu	4	4	4	2	2
Počet protnutí	2	2	1	2	1

Obrázek 5.10: Sériové použití dvou třídících metod během prvního kroku rozkladu.

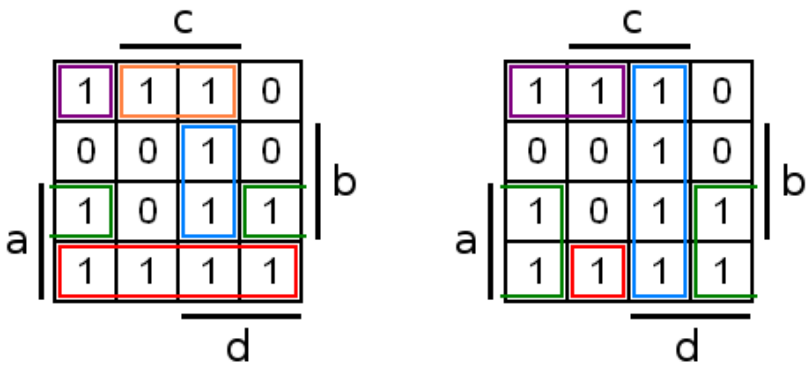
Sloučení termů lze efektivně použít ve dvou variantách:

- Dopředná - sloučení vybraného zpracovávaného termu s ostatními ještě neortogonalními termy.
- Zpětná - sloučení vybraného zpracovávaného termu s již ortogonálními termy.

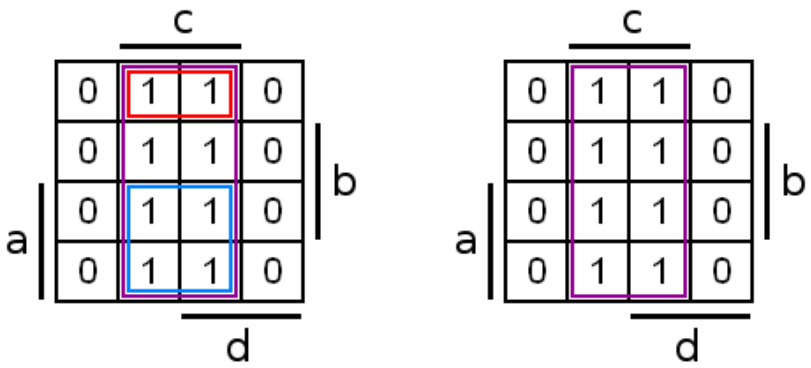
5.3.3 Seřazení termů podle počtu možností sloučení

Na funkci spojení termů můžeme postavit další druh řazení pořadí termů během procesu ortogonalizace. Termy se uspořádají podle statistiky, kde kritériem je z kolika možnými sousedy se mohou eventuálně sloučit. Toto řazení může být výhodné při zpracování logické funkce z malými termy, především mintermy.

Využití tohoto řazení je znázorněno na následujících ilustracích. Na obrázku 5.14 je proveden proces ortogonalizace bez setřízení, na obrázku 5.15 s metodou seřazení termů podle počtu možností sloučení. Červenou barvou jsou označeny termy, které nemají možnost spojení s jiným termem, zelenou barvou termy, které mají jednu možnost spojení s jiným termem a fialovou barvou je označen výsledek.



Obrázek 5.11: Rozklad logické funkce f_5 s použitím sestupného seřazení podle počtu protnutí s ostatními termy - levá Karnaughová mapa, s použitím dvou sériových řadících metod za sebou - pravá Karnaughová mapa.



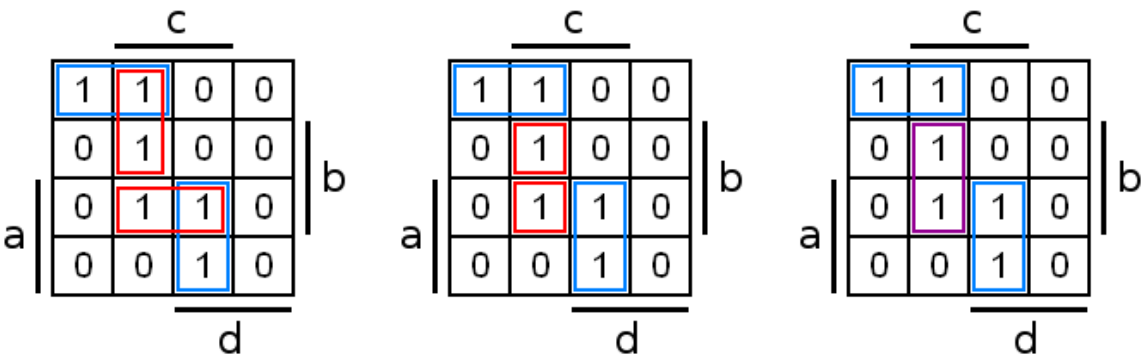
Obrázek 5.12: Pohlcení nadbytečných termů.

5.4 Předzpracování vstupního seznamu termů

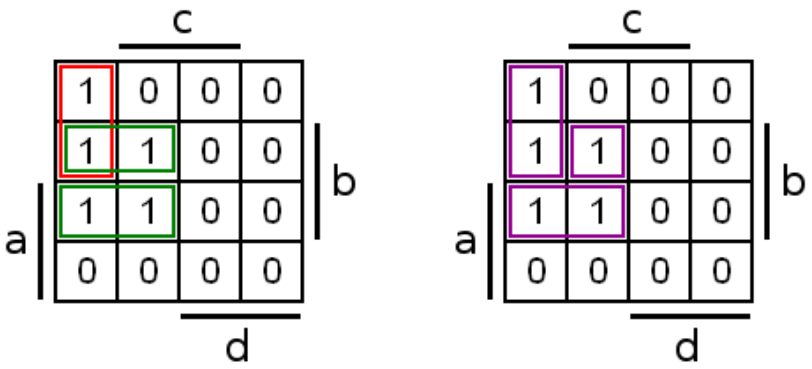
5.4.1 Nastavení výstupů termů

Jak již víme, každý zpracovávaný term logické funkce ve formátu PLA má dvě části. Vstupní část určující vstupní proměnné funkce a výstupní část určující zda term patří do ONset nebo OFFset množiny.

Zaměříme se teď na termy, které nemají ve výstupní části definovány hodnoty (obsahují výstupní hodnoty, které jsou nastaveny na "-"). Můžeme si u nich vybrat, zda je překlopíme do ONsetu nebo OFFsetu. Jak ale poznáme to, jak termy správně nastavit? V tomto projektu nedostaneme odpověď na tuto otázku. Výzkum optimálního nastavení neurčených hodnot na výstupu při procesu ortogonalizace je problém, který přesahuje rozsah této diplomové práce.



Obrázek 5.13: Logická funkce, její ortogonalizovaná podoba a její tvar po sloučení dvou termů.



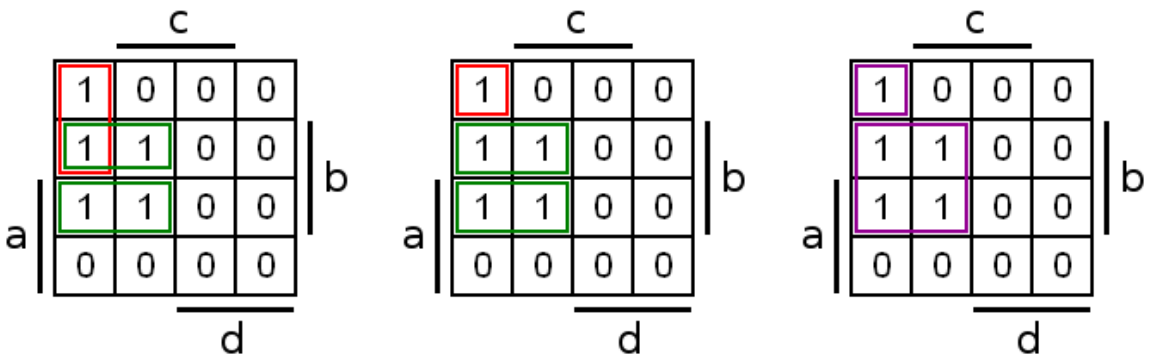
Obrázek 5.14: Logická funkce a jeji ortogonalizovaná podoba bez použití seřazení termů podle počtu možností sloučení.

Existuje ale specifický případ, kdy je nevyhnutelné udělat rozhodnutí, jak se k těmto termům postavit. A to tehdy, když výstup termu není určen žádnou ONset hodnotou. Tento term z pohledu definice logické funkce je zanedbatelný a lze jej vynechat.

5.4.2 Minimalizace termů

Před procesem ortogonalizace se nabízí možnost zkusit vstupní termy minimalizovat. Pro tuto funkci lze použít např. osvědčený nástroj Espresso [13].

Přinese tento krok po rozkladu optimálnější ortogonalizované řešení? Výsledek tohoto experimentu není zřejmý. Zda bude tato možnost prospěšná, určí až testy (viz. kap 7).



Obrázek 5.15: Logická funkce, jeji ortogonalizovaná podoba a podoba s použitím ortogonalizace s řazením termů podle počtu možností sloučení.

5.5 Návrh algoritmu

Záměrem této části je navrhnout základní strukturu algoritmu pro ortogonalizaci logické funkce.

5.5.1 Schéma algoritmu

Schéma koncepce algoritmu je znázorněna na obrázku 5.16. Algoritmus je rozdělen do pěti fází: A, B, C, D, E.

5.5.2 Popis fází algoritmu

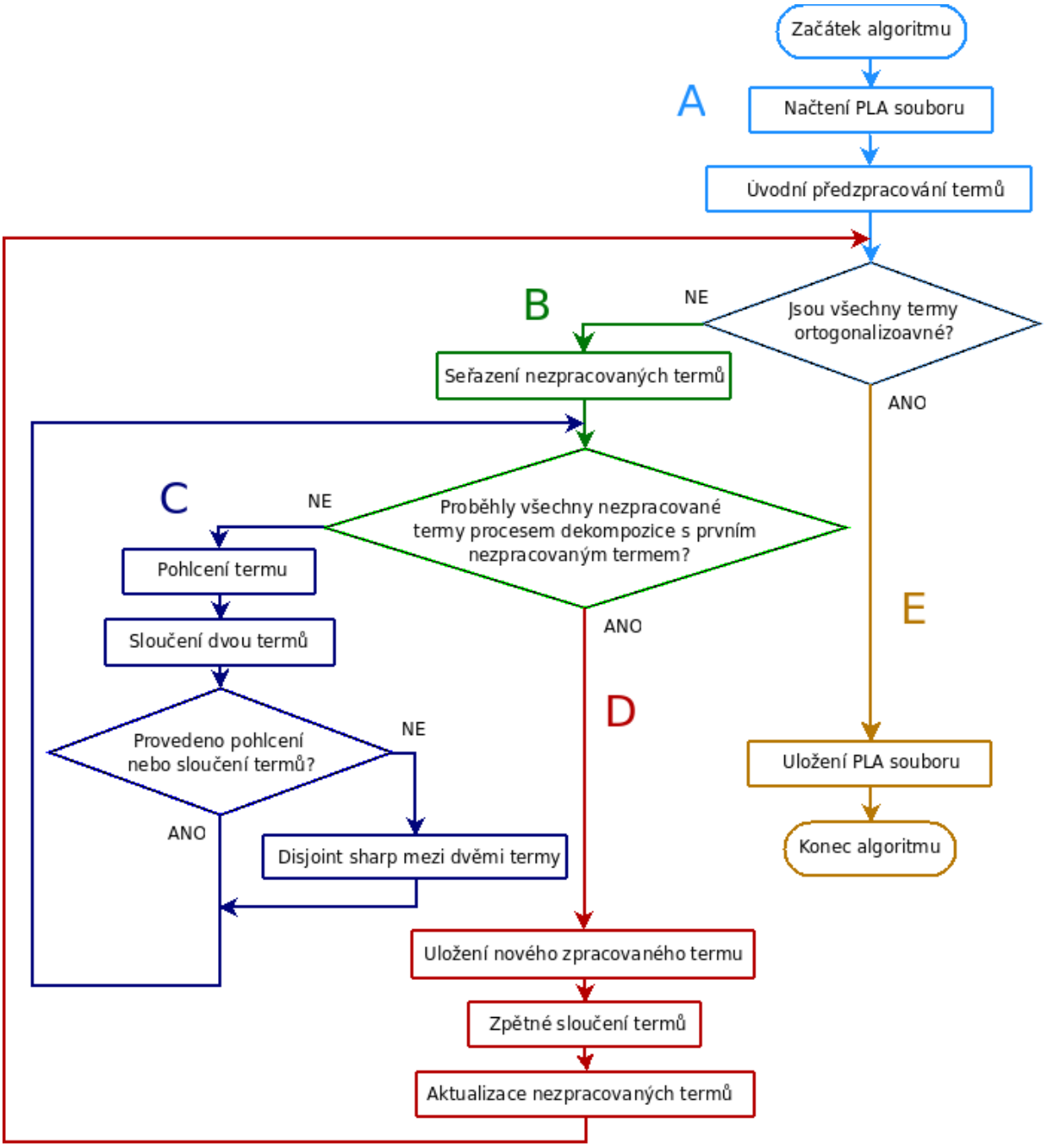
A - 1. fáze Načte se logická funkce z PLA souboru a uloží se do seznamu termů. Dále se provede úvodní předzpracování tohoto seznamu (viz. kap 5.4). V dalším kroku se spustí cyklus, který probíhá tak dlouho, dokud všechny termy nebudou vzájemně ortogonální.

B - 2. fáze V těle cyklu se provede setřízení termů (viz. kap 5.2). Vybere se první term z množiny nezpracovaných termů t_f. Pomocí vnořeného cyklu se postupně prochází všechny nezpracované termy.

C - 3. fáze Na každém aktuálně procházeném termu t_t se nejprve zkusí vzájemná absorpce s t_f a poté možnost slučitelnosti s t_f (viz. kap 5.3). Nedojde-li k pohlcení ani k sloučení provede se operace $t_f \bigodot t_t$.

D - 4. fáze Po ukončení vnořeného cyklu se nově ortogonální term t_f zkusí spojit pomocí metody zpětného sloučení (viz. kap 5.3.2) s předešlými již hotovými termy. Nedojde-li ke sloučení, term se uloží na poslední pozici do seznamu hotových termů. V dalším kroku se aktualizuje množina nezpracovaných termů, odebráním zpracovaného termu t_f a přidáním nových termů vzniklých po operaci $\bigodot$ s aktuálním termem t_f.

E - 5. fáze Po dokončení procesu ortogonalizace se výsledek uloží do nového PLA souboru.



Obrázek 5.16: Schéma základní koncepce algoritmu ortogonalizace logické funkce

Kapitola 6

Realizace řešení

6.1 Programové vybavení pro implementaci algoritmu

Nástroj ortogonalizace logické funkce staví na programu BOOM (viz. kap 2.4), který je vyvíjen v programovacím jazyce C++. Výhody tohoto jazyka jsou rychlost, výkonnost, podpora objektů a snadná platformová přenositelnost. Vytvářený algoritmus má vysoké nároky na paměť, proto je volba C++ pro implementaci vhodná. Je to navíc také součásti zadání.

Pro napsání zdrojových kódů a odladění programu bylo zvoleno prostředí DevCpp verze 4.9.9.2. pod operačním systémem Microsoft Windows XP.

6.2 Pseudokód zjednodušeného algoritmu

(1) $w \leftarrow$ LoadPLA()
(2) $w \leftarrow$ EliminateUndefined(w) nebo $w \leftarrow$ EspressoMinimization(w)
(3) $t_1 \leftarrow$ první term v seznamu w
(4) While (t_1 není poslední term v seznamu w) proveď
(5) Seřadit seznam termů w
(6) $t_2 \leftarrow$ následující term v seznamu za t_1
(7) While (t_2 není poslední term v seznamu w) proveď
(8) $t_3 \leftarrow$ MergeNeighbors(t_1, t_2)
(9) If (nedošlo k sloučení t_1 a t_2) proveď
(10) $z \leftarrow$ TermDisjointSharp(t_1, t_2)
(11) If (došlo k sloučení termů t_1 a t_2) proveď
(12) $t_2 \leftarrow$ následující term za t_1
(13) $t_1 \leftarrow t_3$
(14) Vyprázdnit seznam termů z
(15) $t_2 \leftarrow$ následující term v seznamu, který je za t_2
(16) Přidání t_1 do seznamu *výsledek*
(17) *výsledek* $\leftarrow$ BackMerge(*výsledek*)
(18) $t_1 \leftarrow$ následující term v seznamu, který je za t_1
(19) SavePLA(*výsledek*)

Popis použitých příkazů:

LoadPLA Načtení logické funkce ze souboru formátu PLA
SavePLA Uložení logické funkce do souboru formátu PLA
TermDisjointSharp Operace Disjoint sharp mezi dvěmi termy
MergeNeighbors Spojení dvou slučitelných termů
BackMerge Zpětné spojení dvou slučitelných termů
EliminateUndefined Eliminace termů nepatřící do ONsetu
EspressoMinimization Minimalizace termů pomocí Espressa

6.3 Proces dekompozice termů

Základní programovou konstrukcí rozkladu jsou dva vnořené cykly. V prvním se postupně prochází aktualizovaný seznam nezpracovaných termů (fáze B) a v druhém cyklu se provádí samotný rozklad termů (fáze C). Dekompozici termů provádí funkce TermDisjointSharp.

6.3.1 Funkce TermDisjointSharp

Ústřední funkce celé ortogonalizace. Vstupem jsou dva termy t_1 a t_2, výstupem je seznam termů.

Funkce je rozdělena na tyto 3 části:

1. Otestuje se, zda se termy t_1 a t_2 protínají. Jestliže ne, proces se ukončí, rozklad není nutný. Test protnutí se provede pomocí funkce Intersects [17].

2. Provedení operace $\odot$ mezi termy t_1 a t_2, výsledné termy jsou uloženy do nového seznamu. Rozložení probíhá v cyklu, prochází se po bitech vstupní proměnné termů t_1 a t_2. Když na pozici vstupní hodnoty termu t_1 je DCset a na pozici t_2 logická "0" nebo "1" vytvoří se nový term, který se nastaví podle předurčených podmínek (viz. obrázek 6.1).

3. Vytvoření doplňujícího termu. Term vznikne za předpokladu, že term t_2 má ve výstupní části alespoň jednu "1" na pozici, kde ho term t_2 nemá.

6.4 Setřízení termů

Uspořádání termů probíhá ve fázi B. Vstupem a výstupem je datová struktura PLA.

Proces setřízení je rozdělen na 2 části:

```
for (int j = 0; j <= t2.GetIMSize(); j++ ) {

    if ((t2.GetIM(j) == '-') && (t1.GetIM(j) != '-')) {
        newterm = t2;

        for (int i = 0; i < j; i++ ) {
            if ((newterm.GetIM(i) == '-'))
                newterm.SetIM(i,t1.GetIM(i));
        }

        if (t1.GetIM(j) == '0')
            newterm.SetIM(j,'1');
        if (t1.GetIM(j) == '1')
            newterm.SetIM(j,'0');

        /* uložení termu do seznamu */
        res.push_back(newterm);
    }
}
```

Obrázek 6.1: Výpis zdrojového kódu operace $\odot$ mezi termy t_1 a t_2 (2. část funkce TermDisjointSharp)

1. Podle typu zadaného třídícího kriteria se vytvoří statistika požadovaných vlastností jednotlivých termů, která se uloží do pole. Pořadí položek v poli odpovídá posloupnosti pořadí termů v seznamu.

2. Postupným procházením seznamu termů, který se porovnává se statistikou, se vytváří nový seznam setřízených termů. Vzestupné a sestupné třízení termů závisí na tom, zda se jednotlivé údaje ze statistiky porovnávají od největších hodnot nebo od nejmenších.

Vytvoření statistiky vlastností termů v první části je závislá na metodě řazení. Jsou to tyto implementace:

- Setřízení podle velikosti termů (funkce SizeSortInc a SizeSortDec). Sečte se počet DC-setů ve vstupní části, suma těchto hodnot = velikost daného termu.

- Setřízení podle počtu protnutí s ostatními termy (funkce IntersectionSortInc a IntersectionSortDec). Využívá funkci Intersects [17] pro nalezení společného průniku mezi dvěmi termy.

- Setřízení termů podle počtu možností sloučení (funkce SortByMergeInc a SortByMergeDec). Test slučitelnosti termů je postaven na podmínce odlišnosti právě jednoho literálu "1" - "0" nebo "0" - "1".

6.5 Sloučení termů a absorbce

Dopředné i zpětné složení slučitelných termů je postaveno na funkci MergeNeighbors [17]. Funkce MergeNeighbors plně zastoupí také i funkci pro absorpci termů, kterou není proto nutné samostatně implementovat.

Funkce zpětného sloučení BackMerge je popsána v následujícím odstavci.

6.5.1 Funkce BackMerge

Vstupem a výstupem je datová struktura PLA. Funkce v cyklu testuje možnost spojení posledního vloženého termu v seznamu s předešlými. Po úspěšném sloučení se term uloží na poslední pozici v seznamu a cyklus se opakuje od začátku. Pro sloučení dvou termů se využívá obdobně funkce MergeNeighbors.

6.6 Eliminace termů nepatřící do ONsetu

Termy, které nemají výstup definován žádnou hodnotou nastavenou na "1", odstraní funkce EliminateUndefined. Funkce postupně prochází seznam vstupních termů, termy nepatřící do ONsetu vynechá a ostatní uloží do nového seznamu.

6.7 Minimalizace vstupních termů pomocí Espressa

Program Espresso je v algoritmu ortogonalizace logické funkce spuštěn externě pomocí příkazu system().

Kapitola 7

Testování

V této kapitole se zabýváme experimentálním vyhodnocením navrženého algoritmu, jeho jednotlivých variant a srovnání s cizími dostupnými implementacemi.

7.1 Označení testovaných algoritmů

Vyvíjený algoritmus ortogonalizace logické funkce je implementován v 12 testovacích verzích.

Varianty základního navrženého alogritmu postavené na rozdílném průbežném řazaní termů během procesu ortogonalizace (viz. kap 5.2) jsou:

WithoutSort Bez řazení.

SizeSortInc Podle velikosti termů (vzestupně).

SizeSortDec Podle velikosti termů (sestupně).

IntersectionSortInc Podle počtu protnutí s ostatními termy (vzestupně).

IntersectionSortDec Podle počtu protnutí s ostatními termy (sestupně).

SizeIntersectionSortInc Podle počtu protnutí s ostatními termy (vzestupně) a podle velikosti termů (sestupně).

SizeIntersectionSortDec Podle počtu protnutí s ostatními termy (sestupně) a podle velikosti termů (sestupně).

SortByMergeInc Podle počtu možností sloučení s ostatními termy (vzestupně) .

SortByMergeDec Podle počtu možností sloučení s ostatními termy (sestupně).

Rozšířené verze navrženého algoritmu:

A_BackMerge Varianta algoritmu *SizeSortDec* s použitím zpětného sloučení termů (viz. kap 5.3.2).

A_EliminateUndefined Varianta algoritmu *A_BackMerge* s eliminací termů, které nemají určen výstup ani jednou hodnotou definovanou na ONset (viz.kap 5.4.1).

A_Espresso Varianta algoritmu *A_EliminateUndefined* s minimalizací vstupních termů pomocí Espressa 6.7) .

Cizí testované algoritmy:

Espresso -Ddisjoint Podprogram Espressa pro ortogonalizaci logické funkce (viz. kap 4.1).

DSOP Implementace heuristického algoritmu pro ortogonalizaci logické funkce (viz. kap 4.2).

7.2 Postup testování

Implementovaný nástroj pro ortogonalizaci logické funkce byl testovaný na MCNC benchmarkcích [7], sadě testovacích souborů formátu PLA.

Všechny testy byly prováděny na notebooku Lenovo 3000 N100 s procesorem Intel Centrino Duo 1.8 GHz, s operační pamětí 1024MB RAM a pevným diskem s rychlostí 5400ot/min. Programy pro testování byly kompilovány v DevCpp 4.9.9.2. pod systémem Microsoft Windows XP SP2.

Byly provedeny tyto dva typy testů:

- Test funkčnosti - testování správné činnosti algoritmu.
- Srovnavací test - porovnání výsledků algoritmů

7.3 Test funkčnosti

Záměrem testu funkčnosti bylo ověřit, zda výsledek algoritmu je ortogonální logická funkce (viz. kap 3.2). Pro testování byly použity tyto dva nástroje:

- Podprogram Espressa [13] spouštějící se s parametrem "Dverify" testující, zda nová ortogonalizovaná podoba logické funkce je ekvivalentní s původní logickou funkcí ze které vychází.

- Program Ortho [15] spouštějící se s parametrem "Check" testující, zda termy v logické funkci jsou vzájemně disjunktní.

V následujících srovnávacích testech byly všechny výsledky ověřeny testováním funkčnosti. Každá použitá varianta algoritmu prošla tímto testem úspěšně a jejich výsledek byl ortogonální.

7.4 Srovnávací testy

Při testech byly porovnány tyto experimentální měření:

- srovnání metod průběžného uspořádání termů
- srovnání časů výpočtů jednotlivých algoritmů
- srovnání nejefektivnějších variant navrženého algoritmu s ostatními konkurenčními implementacemi

Výsledky měření se nachází v přílohách A, B, C a D. Symbol "-", který se nachází v některých buňkách tabulky znázorňuje, že ani po několika hodinách algoritmus nespočítal výsledek.

7.5 Porovnání průběžného uspořádaní termů

Byly provedeny testy řadících metod *WithoutSort*, *SizeSortInc*, *SizeSortDec*, *IntersectionSortInc*, *IntersectionSortDec*, *SizeIntersectionSortInc*, *SizeIntersectionSortDec*, *SortByMergeInc* a *SortByMergeDec* na 101 testovacích úlohách. Naměřené hodnoty se nachází v příloze A. Čas, za který jednotlivé varianty algoritmu vypočítaly výsledek jsou v příloze B.

7.5.1 Porovnání výsledků metod řazení

Po sečtení všech hodnot výsledků získaných z testovaných benchmarků můžeme metody seřadit od nejlepší:

1. 14300 termů *SizeSortDec*
2. 14363 termů *SizeIntersectionSortInc*
3. 14360 termů *SizeIntersectionSortInc*
4. 16266 termů *IntersectionSortDec*
5. 20127 termů *WithoutSort*
6. 21925 termů *SortByMergeDec*
7. 44236 termů *IntersectionSortInc*

Metody, které nedokázaly spočítat všechny testované úlohy:

- 22255 termů *SizeSortInc*
- 15945 termů *SortByMergeInc*

7.5.2 Porovnání času metod řazení

Po sečtení všech výsledných časů získaných z testovaných benchmarků můžeme metody seřadit od nejlepší:

1. 110 s *WithoutSort*
2. 241 s *SizeSortDec*
3. 1011 s *SizeIntersectionSortInc*
4. 1069 s *SizeIntersectionSortInc*
5. 1418 s *IntersectionSortDec*
6. 6884 s *SortByMergeDec*
7. 9707 s *IntersectionSortInc*

Metody, které nedokázaly spočítat všechny testované úlohy:

- 1051 s *SizeSortInc*
- 2875 s *SortByMergeInc*

Pozn. časy v tomto meření mají pouze orientační charakter.

7.5.3 Srovnání algoritmu *WithoutSort* s algoritmy s průběžným řazením

Z měření je patrné, že vhodná řazení dopomáhají získat ve výsledcích ortogonalizace menší počet termů než implementace bez řazení, což potvrdilo základní předpoklad použití třídících metod.

Dalším důležitým poznatkem, který můžeme z toho měření odvodit je, že lepším řešením je použití algoritmu bez řazení *WithoutSort* než s použitím nevhodně navržených třídících metod, např. jako jsou *SizeSortInc*, *SizeIntersectionSortInc*, *SortByMergeInc* a *SortByMergeDec*.

7.5.4 Srovnání řazení *SizeSortInc* a *SizeSortDec*

Algoritmus s použitím řazení *SizeSortInc* generoval oproti řešení *SizeSortDec* téměř dvojnásobný počet termů v čtyřnásobném čase. Test tedy prokázal, že užitečnějším řešením je termy řadit sestupným způsobem, což znamená logickou funkci rozkládat od největších termů po nejmenší. Je nutno také zmínit, že algoritmus s *SizeSortInc* uspořádáním nedokázal některé logické funkce z testovacího benchmarku spočítat v přijatelném čase v délce několika hodin.

7.5.5 Srovnání řazení *IntersectionSortInc* a *IntersectionSortDec*

Porovnání algoritmu podle počtu protnutí s ostatními termy ukázalo, že je jednoznačně výhodnější řešení využívající řazení *IntersectionSortDec* oproti *IntersectionSortInc* . Metoda *IntersectionSortInc* ve výsledku vytvořila více než dvojnásobný počet termů než *IntersectionSortDec*. Tzn. prokázalo se, že je výhodnější začít dekomponovat logickou funkci od co nejvíce protínatějších termů, které svým rozkladem ovlivní největší počet ostatních termů. Řešení *IntersectionSortDec* bylo také oproti řešení *IntersectionSortInc* podstatně rychlejší (téměř šestkrát).

7.5.6 Srovnání řazení *SizeSortDec* se sériovým řazením *SizeIntersectionSortInc* a *SizeIntersectionSortDec*

Podle provedených testů jsou tyto tři varianty uspořádání nejúčinnější. Při sériovém řazení dvou třídících metod podle počtu protnutí s ostatními termy a sestupného uspořádání podle velikosti termů, se intuitivně dalo předpokládat, že toto řešení by mohlo dopadnout lépe než samotné řazení *SizeSortDec* z kterého vychází. Výsledek je překvapivý, protože jejich "síla" je vzájemně srovnatelná. Při porovnání času je implementace *SizeSortDec* oproti *SizeIntersectionSortInc*, *SizeIntersectionSortDec* téměr čtyřikrát rychlejší.

Během tohoto testu se neprokázalo viditelné zlepšení, které by sériové řazení třídících metod mohlo přinést.

7.5.7 Srovnání řazení *SortByMergeInc* a *SortByMergeDec* s metodou *WithoutSort*

SortByMergeInc a *SortByMergeDec* jsou třídící metody, které obě ve výsledku zaostávají za variantou algoritmu bez použití řazení. Test tedy ukázal, že jejich použití není v praxi příliš užitečné.

7.5.8 Zhodnocení testování řadících metod algoritmu

Řazení *SizeSortDec* můžeme z hlediska kvality řešení a časové rychlosti definitivně považovat za nejefektivnější třídící metodu algoritmu, která byla odzkoušena. Na tomto algoritmu, proto stavíme následující vylepšené varianty ortogonalizace *A_BackMerge*, *A_EliminateUndefined* a *A_Espresso*.

7.6 Porovnání algoritmu s cizími implementacemi

Vybrané nejsilnější varianty vytvářeného algoritmu *A_BackMerge*, *A_Eli-minateUndefined* a *A_Espresso* jsou v tomto testu porovnány s cizími implementacemi *Espresso -Ddisjoint* a *DSOP* na 86 testovacích úlohách. Pro srovnání je do testování zahrnuta i metoda algoritmu *SizeSortDec*. Naměřené hodnoty se nachází v příloze C. Čas, za který jednotlivé varianty algoritmu vypočítaly výsledek se nachází v příloze D.

7.6.1 Porovnání výsledků algoritmů

Po sečtení všech hodnot výsledků získaných z testovaných benchmarků můžeme algoritmy seřadit od nejlepšího:

1. 11394 termů *A_Espresso*
2. 11613 termů *A_EliminateUndefined*
3. 12490 termů *A_BackMerge*
4. 13702 termů *DSOP*
5. 13799 termů *SizeSortDec*
6. 24901 termů *Espresso -Ddisjoint*

7.6.2 Porovnání časů algoritmů

Po sečtení všech výsledných časů získaných z testovaných benchmarků můžeme algoritmy seřadit od nejlepšího:

1. 7,1 s *DSOP*
2. 25,2 s *Espresso -Ddisjoint*
3. 253,3 s *A_Espresso*
4. 260,7 s *SizeSortDec*
5. 302,2 s *A_EliminateUndefined*
6. 339,1 s *A_BackMerge*

7.6.3 Srovnání algoritmů *SizeSortDec*, *A_BackMerge*, *A_EliminateUndefined* a *A_Espresso*

Algoritmus se zpětným sloučením *A_BackMerge* přináší ve výsledcích kladnou odezvu, kdy můžeme vidět zdokonalení oproti metodě *SizeSortDec* z které vychází.

Algoritmus *A_EliminateUndefined* s eliminací termů s výstupem neurčeným ani jednou hodnotou definovanou na ONset, přináší očekávané vylepšení, které je zřetelné už na první pohled hned ze zadání testovaných úloh.

Algoritmus *A_Espresso* přináší díky novému přístupu značně odlišné výsledky oproti předchozím navrženým variantám. Lze vypozorovat, že minimalizace logické funkce před orotogonalizací může být v některých případech prospěšná, někdy však může navést algoritmus špatným směrem a být tak ke škodě.

V testech byl algoritmus *A_EliminateUndefined* lepší v 54% úloh než *A_Espresso*. Při konečné sumě všech výsledných termů byl ale naopak algoritmus *A_Espresso* úspěšnější než *A_EliminateUndefined* .

7.6.4 Srovnání algoritmu *Espresso -Ddisjoint* s ostatními implementacemi

Espresso -Ddisjoint je jednoduchý algoritmus, který podle testů očekávaně zaostává za ostatními implementacemi.

7.6.5 Srovnání algoritmů *A_EliminateUndefined* , *A_Espresso* a *DSOP*

Algoritmus *A_EliminateUndefined* s porovnáním s *DSOP* byl u 60% testovaných úloh lepší. Algoritmus *A_Espresso* byl ve srovnání s *DSOP* také lepší a to v 54% úloh.

V čem se skrývá výhoda *A_EliminateUndefined* a *A_Espresso* oproti *DSOP*? V první řadě to může být důsledek použité metody průběžného řazení termů. *DSOP* používá setřízení termů podle počtu protnutí s ostatními termy, oproti tomu *A_EliminateUndefined* a *A_Espresso* jsou postaveny na uspořádání termů podle jejich velikosti, které se osvědčily v testech řazení lépe. Dalším důvodem, který může mít vliv na výsledek, jsou metody dopředného a zpětného sloučení termů které, *A_EliminateUndefined* a *A_Espresso* využívají.

Srovnávací testy ukázaly, že *A_EliminateUndefined* a *A_Espresso* jsou nejúčinnější navržené algoritmy této práce.

Kapitola 8

Závěr

Hlavním záměrem projektu bylo seznámení se stávajícími konkurenčními nástroji pro ortogonalizaci logické funkce, návrh a implementace vlastního algoritmu, jeho otestování a srovnání s konkurenčními nástroji. Je možné konstatovat, že všechny tyto cíle diplomové práce byly úspěšně splněny.

Vývoj projektu byl rozdělen do čtyř částí. V první fázi byly formulovány požadavky na zadání práce a byly připomenuty základní definice a termíny z oblasti logických funkcí. Obsahem druhé fáze byl analytický rozbor procesu ortogonalizace, seznámení se s existujícími dostupnými nástroji a návrh vlastního algoritmu. V třetí fázi byla provedena programová realizace algoritmu, při které byl vytvořen funkční nástroj pro ortogonalizaci logické funkce. Ve čtvrté fázi bylo provedeno testování navržené implementace, jejich jednotlivých variant a srovnání s konkurenčními algoritmy.

Nástroj na ortogonalizaci byl implementován v programovacím jazyce C++ a je postaven na programu BOOM. Základním kamenem algoritmu je operace Disjoint sharp, která rozloží logickou funkci na disjunktní termy. Výsledky testu funkčnosti prokázaly, že vytvořený nástroj pracuje správně a dokáže zadanou logickou funkci spolehlivě ortogonalizovat. Experimentálním testováním bylo potvrzeno, že vhodné uspořádání termů během procesu ortogonalizace má zásadní vliv na výsledný počet termů. Z třídících metod se nejvíce osvědčilo průběžné sestupné řazení termů podle velikosti. Další podstatnou funkcí, která dokáže vylepšit výsledek algoritmu, je možnost složení slučitelných termů při rozkladu. Algoritmus s minimalizací vstupních termů pomocí Espressa přinesl v testech zlepšení ve výsledku ortogonalizace, avšak u některých logických funkcí byl naopak k neprospěchu.

Na testovaných úlohách z MCNC benchmarků vytvořený algoritmus generoval lepší výsledky než dostupné konkurenční implementace DSOP a Espresso s podprogramem pro ortogonalizaci.

Budoucí zdokonalování algoritmu je v pokračování hledání nejvýhodnější dekompozice logické funkce, která ji rozloží na co nejmenší počet termů. Případný vývoj vidím v heuristickém rozhodování nastavení vstupních termů logické funkce před ortogonalizací na místech, kde nemají termy ve výstupní hodnotě definovaný ONset ani OFFset. Správná konfigurace těchto hodnot může pomoci během procesu rozkladu dosáhnout výhodnějšího výsledku.

Druhou oblastí, na které by se dalo dál pracovat, je spojení dílčích kroků ortogonalizace s vhodnými algoritmy na minimalizaci.

Zdrojové kódy, jejich komentáře, testovací úlohy a další náležitosti týkající se této práce se nachází na přiloženém CD. Jeho struktura je popsána v příloze F.

Kapitola 9

Literatura a použité zdroje

[1] N. Song, M. Perkowski, Minimization of Exclusive Sum of Products Expressions for Multi-Output Multiple-Valued Input, IEEE Trans. on CAD, Vol. 15, 1996, pp. 385-395.

[2] G. McGuire, Spectra of Boolean Functions, School of Mathematical Sciences University College Dublin, 2007

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

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

[5] R. K. Brayton, G. D. Hachtel, C. T. McMullen, A. L. Sangiovanni-Vincentelli, Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, 1984.

[6] L. Shivakumaraiah, M. A. Thornton, Computation of disjoint cube representations using a maximal binate variable heuristic, System Theory, 2002.

[7] S. Yang, Synthesis on Optimization Benchmarks, User guide, Microelectronic Center, 1991.

[8] R M. Karp, Reducibility among combinatorial problems, Plenum, 1972, pp. 85–103.

[9] M. Miklánek, Úprava minimalizačního nástroje ESPRESSO, Bakalářská práce, FEL ČVUT, 2008.

[10] B. Kernighan, D. Ritchie, The C Programming Language, Prentice Hall, 1978.

[11] A. Bernasconi, V. Ciriani, F. Luccio, L. Pagli, A New Heuristic for DSOP Minimization, Proc. 8th Int. Workshop on Boolean Problems, Freiberg, Germany, 18.-19.9.2008, pp. 169-174.

[12] J. Rybička, Latex pro začátečníky, 3. vydání, Konvoj, 2003.

[13] Espresso - list of the command, http://www.ece.duke.edu/~dwyer/courses/ece52/espresso.1.html

[14] Wikipedia - Orthogonalization, http://en.wikipedia.org/wiki/Orthogonalization

[15] Nástroj Ortho, http://cs.felk.cvut.cz/~fiserp

[16] Formát PLA, http://service.felk.cvut.cz/vlsi/prj/BOOM/pla_c.html

[17] BOOM - the Boolean Minimizer, http://service.felk.cvut.cz/vlsi/prj/BOOM

[18] Wikipedia - Espresso heuristic logic minimizer, http://en.wikipedia.org/wiki/Espresso_heuristic_logic_minimizer

Dodatek A

Porovnání metod průběžného řazení termů

Označení metod průběžného řazení termů:

WS WithoutSort (bez řazení)

SSI SizeSortInc (Podle velikosti termů - vzestupně)

SSD SizeSortDec (Podle velikosti termů - sestupně)

ISI IntersectionSortInc (Podle počtu protnutí s ostatními termy - vzestupně)

ISD IntersectionSortDec (Podle počtu protnutí s ostatními termy - sestupně)

SISI SizeIntersectionSortInc (Podle počtu protnutí s ostatními termy - vzestupně a podle velikosti termů - sestupně)

SISD SizeIntersectionSortDec (Podle počtu protnutí s ostatními termy - sestupně a podle velikosti termů - sestupně)

MI SortByMergeInc (Podle počtu možností sloučení s ostatními termy - vzestupně)

MD SortByMergeDec (Podle počtu možností sloučení s ostatními termy - sestupně)

Jednotlivé naměřené hodnoty metod algoritmů jsou v tabulce uvedeny v termech.

	PLA	**SOP**	**WS**	**SSI**	**SSD**	**ISI**	**ISD**	**SISI**	**SISD**	**MI**	**MD**
1	al2.pla	103	101	109	105	101	109	105	105	109	101
2	alcom.pla	47	46	47	46	46	46	46	46	46	46
3	alu1.pla	19	23	23	19	23	19	19	19	23	23
4	alu2.pla	91	126	195	102	104	122	93	122	122	132
5	alu3.pla	72	159	254	72	199	144	72	72	159	159
6	apla.pla	134	100	100	100	100	100	100	100	100	100

7	b10.pla	138	121	122	121	121	121	121	121	121	120
8	b11.pla	74	31	41	31	41	30	31	31	31	33
9	b12.pla	431	177	698	96	292	120	74	66	188	161
10	b2.pla	110	127	198	111	151	134	111	111	125	127
11	b3.pla	234	697	2215	392	765	382	381	370	752	580
12	b4.pla	54	80	-	63	5826	81	116	63	80	80
13	b7.pla	74	31	41	31	41	30	31	31	31	33
14	b9.pla	123	261	530	181	301	269	181	181	275	244
15	bc0.pla	479	383	504	377	396	382	369	381	385	375
16	bca.pla	301	202	202	202	202	202	202	202	202	207
17	bcb.pla	299	188	188	188	188	188	188	188	187	194
18	bcc.pla	245	163	163	163	163	163	163	163	163	166
19	bcd.pla	243	149	149	149	149	149	149	149	149	156
20	br1.pla	34	20	20	20	20	20	20	20	20	20
21	br2.pla	35	19	19	19	19	19	19	19	19	19
22	check.pla	16	14	14	14	14	14	14	14	14	14
23	check2.pla	16	14	14	14	14	14	14	14	14	14
24	chkn.pla	153	933	-	174	3906	252	171	187	-	1009
25	clpl.pla	20	20	231	20	20	20	20	20	20	20
26	dc1.pla	15	19	20	19	19	19	19	19	19	19
27	dc2.pla	58	45	45	45	45	45	45	45	45	46
28	dist.pla	256	179	179	179	179	179	179	179	179	179
29	dk17.pla	93	76	76	76	76	76	76	76	76	76
30	dk27.pla	52	52	52	52	52	52	52	52	52	52
31	dk48.pla	148	142	142	142	142	142	142	142	142	142
32	ex1010.pla	1024	1020	1020	1020	1020	1020	1020	1020	-	1020
33	ex5.pla	256	179	179	179	179	179	179	179	179	182
34	ex7.pla	123	261	530	181	301	269	181	181	275	244
35	example.pla	153	933	-	174	3906	252	171	187	-	1009
36	exp.pla	89	74	74	74	74	74	74	74	74	74
37	exps.pla	196	165	165	165	166	165	166	165	167	165
38	f51m.pla	256	256	256	256	256	256	256	256	256	256
39	gary.pla	214	122	765	116	421	157	139	112	122	131
40	ibm.pla	173	785	2455	376	865	486	375	381	866	918
41	in0.pla	138	137	137	137	137	137	137	137	137	136
42	in1.pla	110	127	198	111	151	134	111	111	125	127
43	in2.pla	137	405	1640	167	683	286	165	169	479	556
44	in3.pla	75	253	615	121	407	133	120	121	254	276
45	in5.pla	62	250	502	172	455	183	176	173	268	242
46	in6.pla	54	80	-	63	5826	81	116	63	80	80
47	in7.pla	84	591	-	76	2632	352	74	93	1559	869
48	inc.pla	34	33	33	33	33	33	33	33	33	33
49	intb.pla	664	1979	5570	830	3301	1134	838	866	-	2508

50	luc.pla	27	42	42	29	38	30	29	29	42	42
51	m1.pla	32	23	23	23	23	23	23	23	23	24
52	m2.pla	96	64	64	64	64	64	64	64	64	67
53	m3.pla	128	90	90	90	90	90	90	90	90	91
54	m4.pla	256	165	165	165	165	165	165	165	165	164
55	mark1.pla	129	128	143	128	128	128	128	128	128	128
56	max1024.pla	1024	569	569	569	569	569	569	569	568	580
57	max128.pla	128	89	89	89	89	89	89	89	89	89
58	max46.pla	46	47	47	46	47	47	46	46	47	47
59	max512.pla	512	284	284	284	284	284	284	284	284	291
60	misj.pla	48	48	51	48	48	48	48	48	48	48
61	mlp4.pla	256	256	256	256	256	256	256	256	256	256
62	mp2d.pla	123	146	248	166	184	177	158	172	145	155
63	mytest.pla	4	4	4	4	4	4	4	4	4	4
64	mytest3.pla	4	4	4	4	4	4	4	4	4	4
65	newapla.pla	17	40	105	35	106	35	42	35	48	40
66	newapla1.pla	10	10	10	10	10	10	10	10	10	10
67	newapla2.pla	7	7	7	7	7	7	7	7	7	7
68	newbyte.pla	8	8	8	8	8	8	8	8	8	8
69	newcond.pla	31	59	125	37	101	44	36	37	63	60
70	newcpla1.pla	38	65	202	49	202	58	50	53	66	92
71	newcpla2.pla	19	29	42	19	29	24	19	19	29	29
72	newcwp.pla	11	13	13	13	13	13	13	13	13	13
73	newill.pla	8	19	32	8	19	8	8	8	21	19
74	newtag.pla	8	14	21	8	14	14	8	8	14	14
75	newtpla.pla	23	34	44	23	34	31	23	23	34	34
76	newtpla1.pla	4	4	4	4	4	4	4	4	4	4
77	newtpla2.pla	9	9	9	9	9	9	9	9	9	9
78	p82.pla	24	22	22	22	22	22	22	22	22	22
79	pope.pla	64	61	61	61	61	61	61	61	61	61
80	prom1.pla	502	478	478	478	478	478	478	478	478	479
81	prom2.pla	287	287	287	287	287	287	287	287	287	287
82	risc.pla	74	33	43	33	43	32	33	33	33	35
83	root.pla	256	143	143	143	143	143	143	143	141	153
84	ryy6.pla	112	112	2218	112	148	112	112	112	112	214
85	sec.pla	23	28	38	26	28	28	26	26	32	28
86	shift.pla	100	100	100	100	100	100	100	100	100	100
87	soar.pla	529	733	3687	552	1512	742	545	555	-	923
88	sqn.pla	96	69	69	69	69	69	69	69	69	75
89	sqr6.pla	64	64	64	64	64	64	64	64	64	64
90	t1.pla	865	567	572	558	561	563	553	556	571	575
91	t2.pla	301	263	265	262	263	263	262	262	263	266
92	t3.pla	152	86	86	86	86	86	86	86	86	86

93	t4.pla	516	516	516	516	516	516	516	516	516	516
94	tcheck.pla	3	3	3	3	3	3	3	3	3	3
95	ts10.pla	128	128	128	128	128	128	128	128	128	128
96	vg2.pla	110	236	1172	207	374	246	204	210	230	312
97	vtx1.pla	110	516	1121	213	642	260	212	211	538	489
98	wim.pla	16	16	16	16	16	16	16	16	16	16
99	x1dn.pla	112	391	1115	213	595	284	212	211	454	487
100	x6dn.pla	121	220	377	153	255	230	153	153	226	225
101	x9dn.pla	120	437	1370	229	765	316	227	227	510	579
Σ		15700	20127	22255	14300	44236	16266	14360	14363	15945	21925

Tabulka A.1: Srovnávací test - porovnání metod průběžného uspořádání termů

Dodatek B

Porovnání časů metod průběžného řazení termů

Označení metod průběžného řazení termů:

WS WithoutSort (bez řazení)

SSI SizeSortInc (Podle velikosti termů - vzestupně)

SSD SizeSortDec (Podle velikosti termů - sestupně)

ISI IntersectionSortInc (Podle počtu protnutí s ostatními termy - vzestupně)

ISD IntersectionSortDec (Podle počtu protnutí s ostatními termy - sestupně)

SISI SizeIntersectionSortInc (Podle počtu protnutí s ostatními termy - vzestupně a podle velikosti termů - sestupně)

SISD SizeIntersectionSortDec (Podle počtu protnutí s ostatními termy - sestupně a podle velikosti termů - sestupně)

MI SortByMergeInc (Podle počtu možností sloučení s ostatními termy - vzestupně)

MD SortByMergeDec (Podle počtu možností sloučení s ostatními termy - sestupně)

Jednotlivé naměřené hodnoty časů metod algoritmů jsou v tabulce uvedeny v sekundách.

	PLA	**WS**	**SSI**	**SSD**	**ISI**	**ISD**	**SISI**	**SISD**	**MI**	**MD**
1	al2.pla	0	0	0	1	2	2	1	2	1
2	alcom.pla	0	0	0	0	0	0	0	0	0
3	alu1.pla	0	0	0	0	0	0	0	0	0
4	alu2.pla	0	1	1	1	1	1	1	1	1
5	alu3.pla	0	0	0	1	2	0	0	2	3
6	apla.pla	1	0	0	0	0	1	1	1	2

7	b10.pla	0	1	1	1	1	2	1	1	2
8	b11.pla	0	0	0	1	0	0	0	0	0
9	b12.pla	1	39	1	159	6	10	2	142	19
10	b2.pla	0	1	1	0	1	1	1	1	1
11	b3.pla	5	35	11	8	19	16	19	621	60
12	b4.pla	0	-	0	747	1	1	1	1	1
13	b7.pla	0	0	0	0	0	0	0	0	0
14	b9.pla	0	5	0	2	6	4	2	8	2
15	bc0.pla	2	7	5	15	11	V	23	70	50
16	bca.pla	0	2	2	6	8	11	7	24	12
17	bcb.pla	1	1	1	6	10	11	7	17	16
18	bcc.pla	0	1	1	3	4	4	4	14	6
19	bcd.pla	0	1	1	3	2	2	4	9	12
20	br1.pla	0	0	0	0	0	0	0	0	0
21	br2.pla	0	0	0	0	0	0	0	0	0
22	check.pla	0	0	0	0	0	0	0	0	0
23	check2.pla	0	0	0	0	0	0	0	0	0
24	chkn.pla	13	-	1	2067	5	3	4	-	696
25	clpl.pla	0	1	0	0	0	0	0	0	0
26	dc1.pla	0	0	0	0	0	0	0	0	0
27	dc2.pla	0	0	0	0	0	0	0	0	0
28	dist.pla	1	1	1	2	1	1	3	12	7
29	dk17.pla	0	0	0	0	0	0	0	1	1
30	dk27.pla	0	0	0	0	0	1	0	0	0
31	dk48.pla	0	0	0	1	1	0	1	2	2
32	ex1010.pla	9	51	46	160	179	206	182	-	600
33	ex5.pla	0	1	1	4	5	3	4	14	15
34	ex7.pla	1	1	2	2	5	2	2	13	2
35	example.pla	12		2	2063	10	2	5	-	689
36	exp.pla	1	1	0	0	0	0	0	0	0
37	exps.pla	0	1	2	2	1	3	2	5	4
38	f51m.pla	0	1	2	3	2	6	4	6	14
39	gary.pla	1	5	1	11	2	5	2	6	6
40	ibm.pla	1	35	8	8	36	12	15	378	49
41	in0.pla	1	1	0	1	1	1	1	3	2
42	in1.pla	0	0	0	1	1	0	1	3	3
43	in2.pla	1	29	1	13	8	1	2	71	51
44	in3.pla	1	4	0	1	2	1	0	18	13
45	in5.pla	0	3	1	2	2	2	3	8	3
46	in6.pla	0	-	0	712	0	1	0	1	0
47	in7.pla	8	-	0	2569	37	1	1	-	990
48	inc.pla	0	0	0	1	0	0	0	0	0
49	intb.pla	18	353	46	553	435	228	254	-	1925

50	luc.pla	0	0	0	0	0	0	0	0	1
51	m1.pla	0	0	0	0	1	0	0	0	0
52	m2.pla	0	1	0	0	0	1	1	0	0
53	m3.pla	0	0	1	0	0	1	0	1	2
54	m4.pla	0	2	0	2	1	4	2	15	9
55	mark1.pla	1	1	1	0	1	2	1	2	5
56	max1024.pla	8	22	23	89	87	95	99	308	276
57	max128.pla	0	0	1	1	1	1	1	3	2
58	max46.pla	1	0	0	0	0	0	0	0	0
59	max512.pla	2	3	4	7	13	19	19	44	38
60	misj.pla	1	0	0	0	0	0	1	0	0
61	mlp4.pla	1	1	1	3	5	3	2	6	8
62	mp2d.pla	0	1	0	0	3	1	1	2	1
63	mytest.pla	0	0	0	0	0	0	0	0	0
64	mytest3.pla	0	0	0	0	0	0	0	0	0
65	newapla.pla	0	1	0	0	0	0	0	0	0
66	newapla1.pla	0	0	0	0	0	0	0	0	0
67	newapla2.pla	0	0	0	0	0	0	0	0	0
68	newbyte.pla	0	0	0	0	0	0	0	0	0
69	newcond.pla	0	0	0	1	0	0	1	0	0
70	newcpla1.pla	0	1	0	0	0	0	0	0	0
71	newcpla2.pla	0	0	0	0	0	0	0	0	0
72	newcwp.pla	0	0	0	0	0	0	0	0	0
73	newill.pla	0	0	0	0	0	0	0	0	0
74	newtag.pla	0	0	0	0	0	0	0	0	0
75	newtpla.pla	0	0	0	0	0	0	0	0	0
76	newtpla1.pla	0	0	0	0	0	0	0	0	0
77	newtpla2.pla	0	0	0	0	0	0	0	0	0
78	p82.pla	0	0	0	0	0	0	0	0	0
79	pope.pla	0	0	0	0	1	0	0	1	0
80	prom1.pla	2	10	5	24	12	22	22	124	115
81	prom2.pla	1	3	1	3	7	6	7	14	14
82	risc.pla	0	1	0	1	0	0	0	0	0
83	root.pla	0	1	1	1	3	1	2	9	4
84	ryy6.pla	0	30	1	3	6	4	3	5	28
85	sec.pla	0	0	0	0	0	0	0	0	0
86	shift.pla	1	1	0	0	1	0	0	1	1
87	soar.pla	3	320	29	311	320	164	197	-	679
88	sqn.pla	0	1	0	1	1	0	0	0	0
89	sqr6.pla	0	0	0	0	0	0	0	0	1
90	t1.pla	4	18	18	89	105	103	107	364	267
91	t2.pla	1	3	4	5	3	4	3	15	17
92	t3.pla	0	1	0	1	0	0	1	1	1

93	t4.pla	3	13	7	18	21	27	27	107	91
94	tcheck.pla	0	0	0	0	0	0	0	0	0
95	ts10.pla	0	1	0	1	1	0	0	1	2
96	vg2.pla	0	9	1	2	5	1	2	3	6
97	vtx1.pla	1	4	1	3	3	2	3	191	15
98	wim.pla	0	0	0	0	0	0	0	0	0
99	x1dn.pla	1	4	1	4	3	2	2	119	17
100	x6dn.pla	0	3	1	1	2	1	1	5	4
101	x9dn.pla	1	13	1	6	5	2	4	79	20
Σ		110	1051	241	9707	1418	1011	1069	2875	6884

Tabulka B.1: Srovnávací test - Porovnání časů metod průběžného uspořádání termů

Dodatek C

Porovnání navrženého algoritmu s cizími implementacemi

Označení algoritmů:

SSD SizeSortDec (Algoritmus s řazením termů podle velikosti - sestupně)

ABM A_BackMerge (Algoritmus *SizeSortDec* s použitím zpětného sloučení termů)

AEU A_EliminateUndefined (Algoritmus *A_BackMerge* s eliminací termů nepatřící do ONsetu)

AE A_Espresso (Algoritmus *A_EliminateUndefined* s minimalizací vstupních termů pomocí Espressa)

Cizí algoritmy:

DSOP (Implementace heuristického algoritmu pro ortogonalizaci logické funkce)

Espresso -Ddisjoint (Podprogram Espressa pro ortogonalizaci logické funkce)

Jednotlivé naměřené hodnoty algoritmů jsou v tabulce uvedeny v termech.

	PLA	**SOP**	**SSD**	**ABM**	**AEU**	**AE**	**DSOP**	**Espresso -Ddisjoint**
0	al2.pla	103	105	105	105	106	86	121
1	alcom.pla	47	46	46	46	49	47	47
2	alu1.pla	19	19	19	19	19	19	23
3	alu2.pla	91	102	102	98	98	87	138
4	alu3.pla	72	72	72	68	77	68	196
5	apla.pla	134	100	100	78	33	56	112
6	b10.pla	138	121	120	117	116	135	134
7	b11.pla	74	31	30	30	29	35	42
8	b12.pla	431	96	60	60	78	52	302

9	b2.pla	110	111	111	111	119	118	134
10	b3.pla	234	392	343	343	340	288	1043
11	b4.pla	54	63	63	63	116	62	86
12	b7.pla	74	31	30	30	29	35	42
13	b9.pla	123	181	181	181	177	181	288
14	bc0.pla	479	377	335	276	207	365	430
15	bca.pla	301	202	181	181	189	301	301
16	bcb.pla	299	188	160	160	164	299	299
17	bcc.pla	245	163	140	140	146	245	245
18	bcd.pla	243	149	126	126	123	243	243
19	br1.pla	34	20	20	20	21	33	33
20	br2.pla	35	19	15	15	14	35	35
21	check.pla	16	14	12	1	1	4	4
22	check2.pla	16	14	11	1	1	4	4
23	chkn.pla	153	174	174	174	190	168	1511
24	clpl.pla	20	20	20	20	20	20	20
25	dc1.pla	15	19	19	19	12	15	19
26	dc2.pla	58	45	43	43	47	58	58
27	dist.pla	256	179	166	165	134	255	255
28	dk17.pla	93	76	76	40	24	32	57
29	dk27.pla	52	52	52	20	11	14	20
30	dk48.pla	148	142	142	36	26	28	42
31	ex1010.pla	1024	1020	1020	806	899	810	810
32	ex5.pla	256	179	165	165	125	256	256
33	ex7.pla	123	181	181	181	177	181	288
34	example.pla	153	174	174	174	190	168	1511
35	exp.pla	89	74	70	70	76	89	89
36	exps.pla	196	165	143	143	155	193	196
37	f51m.pla	256	256	256	255	76	255	255
38	gary.pla	214	116	116	116	122	121	123
39	ibm.pla	173	376	376	376	394	361	1046
40	in0.pla	138	137	136	133	122	135	135
41	in1.pla	110	111	111	111	119	118	134
42	in2.pla	137	167	167	167	181	179	615
43	in3.pla	75	121	121	121	133	94	308
44	in5.pla	62	172	165	165	171	125	348
45	in6.pla	54	63	63	63	116	61	86
46	in7.pla	84	76	76	76	63	56	1384
47	inc.pla	34	33	33	33	36	34	34
48	intb.pla	664	830	821	821	841	791	2538
49	luc.pla	27	29	29	29	35	28	42
50	m1.pla	32	23	21	21	20	32	32
51	m2.pla	96	64	53	53	58	96	96

52	m3.pla	128	90	73	73	88	128	128
53	m4.pla	256	165	123	123	150	248	256
54	max1024.pla	1024	569	416	416	363	1024	1024
55	max128.pla	128	89	85	85	113	128	128
56	max46.pla	46	46	46	46	46	46	47
57	max512.pla	512	284	208	208	176	512	512
58	misj.pla	48	48	48	48	82	47	48
59	mlp4.pla	256	256	238	225	151	225	225
60	mp2d.pla	123	166	166	166	140	137	166
61	mytest.pla	4	4	4	4	2	4	4
62	mytest3.pla	4	4	4	3	4	3	3
63	p82.pla	24	22	22	22	22	24	24
64	pope.pla	64	61	61	61	91	64	64
65	prom1.pla	502	478	475	475	485	465	502
66	prom2.pla	287	287	287	287	287	287	287
67	risc.pla	74	33	32	32	31	35	42
68	ryy6.pla	112	112	112	112	128	112	112
69	pla	23	26	26	26	27	26	30
70	shift.pla	100	100	100	100	105	100	100
71	soar.pla	529	552	552	552	547	454	805
72	sqn.pla	96	69	65	54	48	72	84
73	sqr6.pla	64	64	64	63	52	63	63
74	t1.pla	865	558	290	256	144	542	860
75	t2.pla	301	262	164	76	63	76	130
76	t3.pla	152	86	46	42	34	148	148
77	t4.pla	516	516	252	38	22	38	38
78	tcheck.pla	3	3	3	3	3	3	3
79	ts10.pla	128	128	128	128	128	128	128
80	vg2.pla	110	207	207	207	207	204	236
81	vtx1.pla	110	213	213	213	213	204	710
82	wim.pla	16	16	15	10	17	10	10
83	x1dn.pla	112	213	213	213	213	204	517
84	x6dn.pla	121	153	152	152	158	142	262
85	x9dn.pla	120	229	229	229	229	218	595
Σ		15122	13799	12490	11613	11394	13702	24901

Tabulka C.1: Srovnávací test - porovnání vybraných metod navrženého algoritmu s cizími dostupnými implementacemi

Dodatek D

Porovnání časů navrženého algoritmu s cizími implementacemi

Označení algoritmů:

SSD SizeSortDec (Algoritmus s řazením termů podle velikosti - sestupně)

ABM A_BackMerge (Algoritmus *SizeSortDec* s použitím zpětného sloučení termů)

AEU A_EliminateUndefined (Algoritmus *A_BackMerge* s eliminací termů nepatřící do ONsetu)

AE A_Espresso (Algoritmus *A_EliminateUndefined* s minimalizací vstupních termů pomocí Espressa)

Cizí algoritmy:

DSOP (Implementace heuristického algoritmu pro ortogonalizaci logické funkce)

Espresso -Ddisjoint (Podprogram Espressa pro ortogonalizaci logické funkce)

Jednotlivé naměřené hodnoty časů algoritmů jsou v tabulce uvedeny v sekundách.

	PLA	**SSD**	**ABM**	**AEU**	**AE**	**DSOP**	**Espresso -Ddisjoint**
0	al2.pla	0,2	0,4	0,4	0,4	0,1	0,2
1	alcom.pla	0,5	0,5	0,2	0,6	0,0	0,7
2	alu1.pla	0,1	0,1	0,1	0,3	0,0	0,1
3	alu2.pla	0,2	0,2	0,2	1,1	0,1	0,1
4	alu3.pla	0,0	0,0	0,0	0,2	0,0	0,2
5	apla.pla	1,0	1,0	1,0	0,9	0,0	0,2
6	b10.pla	0,5	0,0	0,0	1,9	0,0	0,1
7	b11.pla	0,2	0,3	0,3	0,7	0,0	0,2
8	b12.pla	0,7	0,4	0,3	0,5	0,1	0,1

9	b2.pla	0,6	0,5	0,5	0,7	0,2	0,4
10	b3.pla	11,1	6,2	5,0	6,6	0,1	0,5
11	b4.pla	0,3	0,9	0,9	1,3	0,0	0,5
12	b7.pla	0,2	0,2	0,2	0,6	0,0	0,1
13	b9.pla	0,4	1,6	1,6	1,4	0,1	0,1
14	bc0.pla	5,1	6,2	10,9	1,1	0,2	0,3
15	bca.pla	2,0	2,6	4,1	1,3	0,1	0,8
16	bcb.pla	0,9	1,1	3,0	1,3	0,1	0,3
17	bcc.pla	1,1	2,2	1,5	1,4	0,1	0,2
18	bcd.pla	1,0	1,1	1,1	1,8	0,1	0,2
19	br1.pla	0,1	0,1	0,0	0,0	0,0	0,2
20	br2.pla	0,0	0,0	0,0	0,1	0,0	0,1
21	check.pla	0,1	0,1	0,1	0,6	0,0	0,1
22	check2.pla	0,0	0,0	0,0	0,8	0,0	0,1
23	chkn.pla	1,2	1,3	1,6	2,6	0,2	0,1
24	clpl.pla	0,7	0,8	0,5	0,8	0,0	0,8
25	dc1.pla	0,0	0,0	0,0	0,1	0,0	0,1
26	dc2.pla	0,0	0,0	0,0	0,1	0,0	0,1
27	dist.pla	1,2	1,4	1,4	1,4	0,0	0,1
28	dk17.pla	0,3	0,5	0,4	0,4	0,0	0,2
29	dk27.pla	0,0	0,0	0,0	0,1	0,0	0,0
30	dk48.pla	0,4	1,3	1,3	0,9	0,0	0,1
31	ex1010.pla	45,9	65,7	46,7	45,8	0,1	0,0
32	ex5.pla	1,1	2,9	1,5	1,0	0,5	0,3
33	ex7.pla	2,4	2,6	1,2	1,1	0,1	1,8
34	example.pla	2,3	3,2	1,3	1,3	0,2	0,3
35	exp.pla	0,2	0,3	0,2	1,1	0,0	0,8
36	exps.pla	2,1	1,6	0,9	0,9	0,1	0,1
37	f51m.pla	2,2	3,6	1,6	0,8	0,0	0,3
38	gary.pla	1,2	2,0	1,3	1,4	0,1	0,2
39	ibm.pla	7,6	5,5	5,5	13,4	0,2	0,2
40	in0.pla	0,3	1,1	1,1	0,9	0,0	0,8
41	in1.pla	0,4	0,5	0,6	0,5	0,1	0,2
42	in2.pla	1,0	1,2	1,2	1,1	0,1	0,3
43	in3.pla	0,4	0,9	1,3	0,9	0,1	0,3
44	in5.pla	1,1	1,1	2,2	1,7	0,0	0,2
45	in6.pla	0,3	0,8	1,0	0,6	0,0	0,2
46	in7.pla	0,4	1,5	0,3	0,6	0,0	0,2
47	inc.pla	0,2	0,2	0,1	0,3	0,0	0,1
48	intb.pla	45,6	63,4	73,5	69,2	1,1	0,1
49	luc.pla	0,1	0,2	0,3	0,2	0,0	3,8
50	m1.pla	0,4	0,4	0,2	0,6	0,0	0,1
51	m2.pla	0,4	0,5	1,5	0,7	0,0	0,1

52	m3.pla	1,2	1,9	0,7	0,8	0,1	0,2
53	m4.pla	0,4	2,6	1,4	1,4	0,1	0,2
54	max1024.pla	22,8	31,1	33,8	8,1	0,2	0,4
55	max128.pla	0,9	1,2	0,4	0,8	0,1	0,8
56	max46.pla	0,4	0,7	0,5	0,5	0,0	0,2
57	max512.pla	3,8	3,3	3,3	1,2	0,1	0,1
58	misj.pla	0,1	0,1	0,1	0,1	0,0	0,3
59	mlp4.pla	1,4	2,8	1,3	1,7	0,0	0,2
60	mp2d.pla	0,4	1,4	1,0	1,4	0,0	0,2
61	mytest.pla	0,2	0,5	0,1	0,4	0,0	0,2
62	mytest3.pla	0,0	0,0	0,0	0,0	0,0	0,0
63	p82.pla	0,0	0,0	0,0	0,0	0,0	0,0
64	pope.pla	0,1	0,2	0,2	0,8	0,1	0,1
65	prom1.pla	5,3	14,6	6,0	6,3	0,0	0,3
66	prom2.pla	1,2	2,1	2,7	2,7	0,2	0,0
67	risc.pla	0,4	0,5	0,8	0,3	0,0	0,6
68	ryy6.pla	1,4	1,6	2,1	2,3	0,1	0,1
69	pla	0,3	0,3	0,3	0,5	0,0	0,3
70	shift.pla	0,4	1,0	0,9	1,6	0,0	0,0
71	soar.pla	28,3	33,2	33,2	27,2	0,5	0,1
72	sqn.pla	0,9	0,9	0,9	0,9	0,0	1,8
73	sqr6.pla	0,2	0,7	0,5	0,7	0,0	0,1
74	t1.pla	17,6	16,5	20,4	2,9	0,1	0,1
75	t2.pla	4,2	5,3	1,4	0,8	0,0	0,3
76	t3.pla	1,7	1,7	0,9	0,3	0,0	0,1
77	t4.pla	14,7	14,7	0,7	0,7	0,0	0,1
78	tcheck.pla	0,1	0,1	0,1	0,6	0,0	0,1
79	ts10.pla	0,3	1,2	0,4	1,3	0,0	0,0
80	vg2.pla	1,4	1,7	2,2	3,6	0,1	0,1
81	vtx1.pla	1,2	1,4	1,4	2,6	0,1	0,5
82	wim.pla	0,4	0,8	0,6	0,6	0,0	0,4
83	x1dn.pla	1,4	2,0	1,4	1,0	0,1	0,0
84	x6dn.pla	0,7	0,2	0,6	0,6	0,1	0,3
85	x9dn.pla	1,3	2,3	2,0	2,5	0,1	0,2
Σ		260,7	339,1	302,2	253,3	7,1	25,2

Tabulka D.1: Srovnávací test - porovnání časů vybraných metod navrženého algoritmu s cizími dostupnými implementacemi

Dodatek E

Ovládání navrženého programu

Program pro ortogonalizaci logické funkce je určen pro platformu Windows.

Spuštění programu v příkazové řádce:

Orthogonalization.exe [*číslo_algoritmu*] [*vstupní_PLA_soubor*] [*výstupní_PLA_soubor*]

Parametr "číslo_algoritmu":

"**0**" Základní algoritmus bez průběžného řazení.

"**1**" Základní algoritmus s řazením podle velikosti termů (vzestupně).

"**2**" Základní algoritmus s řazením podle velikosti termů (sestupně).

"**3**" Základní algoritmus s řazením podle počtu protnutí s ostatními termy (vzestupně).

"**4**" Základní algoritmus s řazením podle počtu protnutí s ostatními termy (sestupně).

"**5**" Základní algoritmus s řazením podle počtu protnutí s ostatními termy (vzestupně) a podle velikosti termů (sestupně).

"**6**" Základní algoritmus s řazením podle počtu protnutí s ostatními termy (sestupně) a podle velikosti termů (sestupně).

"**7**" Základní algoritmus s řazením podle počtu možností sloučení s ostatními termy. (vzestupně).

"**8**" Základní algoritmus s řazením podle počtu možností sloučení s ostatními termy. (sestupně).

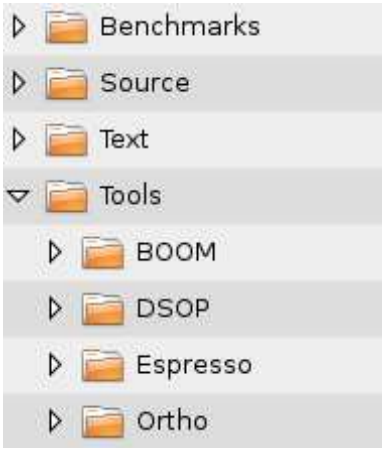
"**9**" Algoritmus **2** s použitím zpětného sloučení termů.

"**10**" Algoritmus **9** s eliminací termů nepatřící do ONsetu.

"**11**" Algoritmus **10** s minimalizací vstupních termů pomocí Espressa.

Dodatek F

Obsah přiloženého CD



Obrázek F.1: Obsah přiloženého CD

- **Benchmarks** MCNC testovací úlohy formátu PLA
- **Source** Zdrojové soubory navrženého nástroje na ortogonalizaci logické funkce.
- **Text** Zdrojové soubory tohoto textu a tištěný formát PDF.
- **Tools** Použité nástroje
  - BOOM Zdrojové soubory minimalizátoru BOOM 2.7
  - DSOP Spustitelný program DSOP [11] na ortogonalizaci logické funkce.
  - Espresso Spustitelný program a manuál ESPRESSA 2.3
  - Ortho Spustitelný program Ortho pro testování ortogonalizace logické funkce.